High-Order Schemes for the Shallow Water Equations on GPUs

Espen Graff Berglie

Master of Science in Physics and Mathematics

Submission date:	Januar 2013
Supervisor:	Helge Holden, MATH
Co-supervisor:	Knut-Andreas Lie, SINTEF ICT
	André Brodtkorb, SINTEF ICT

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

In this thesis, well-balanced, central-upwind high-resolution methods of high order are developed for the two-dimensional shallow water equations, on the graphics processing unit (GPU). The methods are based on a fifth-order Weighted Essentially Non-Oscillating (WENO) reconstruction technique and a fourth-order Gaussian quadrature for the one-sided interface fluxes. Two schemes are implemented, one with bilinear interpolation of the bottom topography and a second-order quadrature for the bed slope source term, and one with a fourth-order source term quadrature and a fifth-order hydrostatic WENO reconstruction of the water height. The high-order schemes are compared to a second-order scheme by Kurganov and Petrova, which recently has been implemented on the GPU by Brodtkorb.

The schemes are shown to be well-balanced and are capable of outperforming the second-order scheme on smooth problems provided that dry states and discontinuities do not occur. The performance gain is larger after a low number of time steps, where speed-ups by a factor between 3 and 8 are documented. As the system evolves, the accuracy of the high-order methods drops, and the performance gain is reduced to a factor around 1.5. The schemes do, however, not support outflow and inflow boundary conditions, and are not yet tested on real-world problems.

The high-order scheme using the hydrostatic reconstruction and a fourth-order source term quadrature is implemented in such a way that an extension to quadratures and reconstructions of even higher order, should be fairly straight forward.

Norsk sammendrag

I denne masteroppgaven utvikles balanserte høyoppløsningsmetoder av høy orden for gruntvannslikningene på grafiske prosesseringsenheter (GPU-er). Metodene er basert på en femte ordens vektet essensielt ikke-oscillerende (WENO) rekonstruksjonsteknikk og et fjerde ordens gaussisk kvadratur for de ensidige kantfluksene. To skjemaer er implementert. Det ene har en bilineær interpolasjon av bunnen og andre ordens kildeleddskvadratur. Det andre har et fjerde ordens kildeleddskvadratur og en femte ordens hydrostatisk WENO-rekonstruksjon av vannhøyden. Skjemaene av høy orden er sammenliknet med et skjema av Kurganov og Petrova av orden to, som nylig er implementert på GPU av Brodtkorb.

Skjemaene vises å være balanserte og er i stand til å utkonkurrere skjemaet av orden to på glatte problemer så fremt tørre soner og diskontinuiteter ikke oppstår. Effekten er størst etter få tidssteg, der kjøretiden vises å være rundt 3-8 ganger raskere. Etter hvert som systemet utvikles, minker skjemaenes nøyaktighet og kjøretiden vil kun være omtrent 1,5 ganger raskere. Skjemaene støtter dog ikke ut- og innstrøms grensebetingelser og er heller ikke testet på virkelige problemer.

Skjemaet som bruker en hydrostatisk rekonstruksjon og fjerde ordens kildeleddskvaratur, er implementert på en slik måte at det vil være rimelig ukomplisert å utvide det til å bruke kvadraturer og rekonstruksjoner av enda høyere orden.

Preface

This document is my thesis for the Master's degree program Industrial Mathematics at the Norwegian University of Science and Technology (NTNU), in the field of numerical mathematics. It is a joint project with NTNU and the independent research organization SINTEF, carried out in the period September 2012 - January 2013.

First and foremost I would like to thank my supervisors at SINTEF , André Brodtkorb and Knut Andreas Lie, for numerous helpful e-mail correspondences throughout the autumn and winter, and my supervisor at NTNU, Helge Holden, for making the cooperation with SINTEF possible. I will also thank Martin L. Sætra, PhD candidate at SINTEF, who, alongside with André, was most helpful during my one week stay at their office in Oslo.

I would also like to thank the IT support crew of the Department of Mathematical Sciences at NTNU, and in particular senior engineer Per Kristian Hove, for setting up the software correctly in the computer lab. Without you it would not have been possible to do the thesis work in Trondheim. Also associate professor Anne Cathrine Elster and PhD candidate Rune E. Jensen, in the Department of Computer and Information Science at NTNU, should be honoured for warmly welcoming me in their computer lab, giving me access to better hardware.

Lastly, a big thank you goes to my fellow students accompanying me in the computer lab. Our Master's thesis related discussions, but mostly our extensive coffee breaks, have been of great importance.

Espen Graff Berglie
Trondheim
27th January 2013

Contents

Preface v

1 Introduction 1

2 Hyperbolic conservation laws 3
- 2.1 The general conservation law 3
- 2.2 The shallow water equations 5
- 2.3 Finite-volume methods 7

3 High-resolution methods 11
- 3.1 Reconstruction from cell averages 12
- 3.2 The Riemann problem and flux integral 15
- 3.3 Total variation diminishing Runge-Kutta methods 17
- 3.4 Source term quadrature 18
- 3.5 High-order schemes 19
- 3.6 Weighted Essentially Non-Oscillating reconstructions 20

4 Graphics processor units in heterogeneous computing 29
- 4.1 GPU architecture 29
- 4.2 GPU computing using CUDA 31
  - 4.2.1 Stencil computations on the GPU 32

5 Numerical aspects of the shallow water equations 37
- 5.1 Fourth-order well-balanced source terms 37
- 5.2 Hydrostatic reconstruction 41
- 5.3 Dry states 43

6 Implementation on the GPU **45**
6.1 Outline of the scheme 46
6.2 Reconstruction of the conserved variables 47
6.3 The flux-source kernel 49
6.3.1 Implementation of second-order source term 54
6.3.2 Shared memory usage 55
6.3.3 Block size 57
6.3.3.1 Optimal block size 58
6.3.4 Note on the intrinsic *powf()* function 59
6.4 The time step kernel 59
6.5 Boundary conditions 60

7 Numerical results **61**
7.1 Code performance 61
7.2 The idealised circular dam break problem 64
7.3 Accuracy and verification of order 67
7.4 Test of well-balanced property 82
7.5 Unwanted oscillations 86

8 Concluding remarks **89**
8.1 Hardware impact 90
8.2 Accuracy and speed-ups 90
8.3 Further research 91

1. Introduction

Developing numerical methods for hyperbolic conservation laws is a challenging research field, subject to significant improvements over the past decades. Hyperbolic problems typically includes discontinuous phenomena, which regular finite-volume and finite-difference methods might fail to capture correctly, in that they either introduce spurious oscillations or too much numerical diffusion. In 1983, Harten presented high-resolution methods [10] as a remedy for most of these numerical issues. The principles of high-resolution methods have been developed further in the new millennium, and much of the work here is based on Shu, Kurganov, Levy and Audusse, to name a few[1].

High-resolution methods is the basis for this thesis. We will implement high-order high-resolution finite-volume methods for the shallow water equations, based on a Weighted Essentially Non-Oscillating (WENO) interpolation technique, and the idea that analytical stationary states should be captured exactly also numerically, so-called well-balanced methods.

High-resolution methods are rather computationally expensive, even for low-order accuracy, including several numerical components, such as interpolation, quadrature rules and possibly also Riemann solvers. However, they are essentially just slightly advanced examples of stencil computation, and thus of a highly parallel nature, which we will exploit by implementing the solvers on the Graphics Processing Unit (GPU).

Using the GPU as a computational unit is a rather fresh research area, and over the last decade, we have seen a great improvement both in terms of the hardware being more compatible with non-graphics computation, and also with the introduction of programming environments aimed at easing the transition into developing code for the GPU, rather than for the CPU. A good introduction to solving conservation laws on the GPU would be to read [9],

[1]See e.g. `http://www.cscamm.umd.edu/centpack/publications/`

followed by e.g. [4,6,7]. We will restrict ourselves to GPUs from NVIDIA, using the CUDA framework for programming. The GPU used for initial development, was the outdated NVIDIA Quadro FX 380, which was replaced by the Quadro 5000 towards the end of the testing process.

The main concern in this research, has been whether a high-order scheme on the GPU, due to its increased accuracy, would be comparable to the second-order scheme implemented by Brodtkorb [7] at SINTEF ICT. Clearly, in a highly serialized code, this would not be the case, as the computational cost of a WENO scheme is massive compared to simpler interpolation techniques. However, on the GPU, the parallel potential of the second-order scheme is in fact restricted by GPU memory, which suggests that it could be beneficial to apply a more complex stencil to the elements loaded into memory. We will investigate the possibilities of a speed-up by the means of running a more accurate method on a coarser grid. The natural follow-up question is in what circumstances a high-order scheme would prove to be as good as, or even better, than a second-order scheme.

In chapters 2 and 3, we establish the theoretical framework of the thesis, by introducing hyperbolic conservation laws, finite-volume methods, the shallow water equations and high-resolution methods, as well as the WENO reconstruction technique, which is essential to this thesis. Chapter 4 gives a brief walk-through of GPU hardware and GPU programming in the context of stencil computing. Chapter 5 is devoted to mathematical aspects particular to numerically solving the shallow water equations. The actual GPU implementation is described in chapter 6. In chapter 7 we present and discuss the results, culminating in the summary and concluding remarks in chapter 8.

2. Hyperbolic conservation laws

2.1. The general conservation law

Several physical phenomena are governed by a principle of conservation. In essence, a conservation law states that the rate of change of some conserved variable within a domain Ω, is equal to the sum of inflow and production minus the outflow and removal. For example a population could be modeled by a conservation law, where the inflow and outflow would correspond to immigration and emigration, whereas the production and removal would correspond to births and deaths. Another typical example is water flow, which is the cornerstone of this thesis.

Mathematically the conservation law for Q on the domain $x \in \Omega \subset \mathbb{R}^n$, is described as the initial value problem

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} Q(x,t)\mathrm{d}x + \int_{\partial\Omega} F(x,t,Q)\cdot n\,\mathrm{d}s = \int_{\Omega} S(x,t,Q)\mathrm{d}x, \quad t \geq 0, \tag{2.1}$$

meaning that the rate of change of some conserved variable, Q, after the initial time, $t = 0$, is governed by the function, F, defined on the domain boundary and commonly referred to as the flux function, as well as the source term, S, which can also be dependent on other variables than the three mentioned here. The conserved variables might be a vector valued function, meaning that $Q, F, S : \mathbb{R}^m \to \mathbb{R}^m$.

Consider for example the conservation of mass for a fluid flowing over a control area Ω, with no source terms. The mass is given by $\rho V = \rho h A$. The flux is a product of the flow speed, u, and the area, A, of the boundary, Γ. Provided that the mass density is constant, the conservation of mass is therefore equivalent to a conservation law of volume, which, using (2.1), reads

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega h\,\mathrm{d}x\mathrm{d}y + \int_\Gamma hu\cdot n\,\mathrm{d}s = 0.$$

This equation, we will see, is the first component of the shallow water equations.

The integral form (2.1) of the conservation law can, provided that the solution is sufficiently smooth, be written as a partial differential equation (PDE), by applying the divergence theorem to the flux integral, obtaining

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega Q(x,t)\mathrm{d}x + \int_\Omega \nabla\cdot F(x,t,Q)\,\mathrm{d}x = \int_\Omega S(x,t,Q)\,\mathrm{d}x,$$

$$\int_\Omega \left(Q_t(x,t,Q) + \nabla\cdot F(x,t,Q) - S(x,t,Q)\right)\,\mathrm{d}x = 0,$$

which yields the PDE

$$Q_t(x,t) + \nabla\cdot F(x,t,Q) = S(x,t,Q). \tag{2.2}$$

The PDE (2.2) is therefore a special case of the integral form (2.1).

In the problems relevant to this thesis we only deal with conservation laws in which the flux function, F, only depends on Q. Let us for time being also restrict ourselves to the homogenous system,

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega Q(x,t)\mathrm{d}x + \int_{\partial\Omega} F(Q)\cdot n\,\mathrm{d}s = 0, \quad \Omega\subset\mathbb{R}^n,\ t\geq 0. \tag{2.3}$$

The term hyperbolicity is concerned with the eigenvalues of the system (2.3), more precisely the eigenvalues of $J_F(Q)$, the Jacobian of F. We denote these eigenvalues $\lambda_k^F(Q)$. When $\lambda_k^F(Q)$ are real for all k, we call the system *hyperbolic*, and if they in addition are distinct, we say that the system is *strictly hyberbolic.*

In this thesis we will concentrate on problems in two spatial dimensions. It is customary to decompose the flux, F, into one component in each spatial dimension, F and G for the x and y direction, respectively. This leads to the notation

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega Q(x,t)\mathrm{d}x + \int_{\partial\Omega} \big(F,G\big)\cdot\big(n_x,n_y\big)\,\mathrm{d}s = 0, \quad \Omega\subset\mathbb{R}^n,\ t\geq 0, \tag{2.4}$$

where the flux integral is expressed in terms of an inner product. We will stick to this notation, even though in the general case, it would have been confusing to name the flux F in the x direction.

2.2. The shallow water equations

The shallow water equations are derived from the Navier-Stokes equation under the assumption that the water elevation is small compared to the horizontal length of the domain, so that the vertical velocity component can be ignored. The non-homogeneous shallow water equations, including a bed slope source term, are on its differential form (2.2), given by

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghB_x \\ -ghB_y \end{bmatrix}, \tag{2.5}$$

or, in short,

$$Q_t + F(Q)_x + G(Q)_y = S_B(Q, \nabla B),$$

which describes the conservation of mass and momentum. Here h is the water elevation, u and v are the velocities in the x and y direction, respectively, g is the gravitational acceleration and the bottom topography is given by the function $B(x, y)$. The bed slope source term, $S_B(Q, \nabla B)$, accounts for water flow induced by the varying bottom topography. Consequently, we are left with a homogeneous system if the bottom topography is flat.

Other source terms of relevance include the bed shear stress friction, $S_F(Q)$, and the Coriolis force, $S_C(Q)$. The bed shear stress friction reads

$$S_F(Q) = \begin{bmatrix} 0 \\ -gu\sqrt{u^2+v^2}/C_z^2 \\ -gv\sqrt{u^2+v^2}/C_z^2 \end{bmatrix}, \tag{2.6}$$

where C_z is the Chézy friction coefficient, which can be written $C_z = h^{1/6}/n$, using Manning's roughness coefficient, n.

In medium-scale computations, where the domain is big enough for the geometry, rotation and curvature of the earth to have an impact, we also add a Coriolis force source term,

$$S_C(Q) = \begin{bmatrix} 0 \\ fv \\ -fu \end{bmatrix}, \tag{2.7}$$

where f is the Coriolis parameter.

The general non-homogeneous shallow water equations then read

$$Q_t + F(Q)_x + G(Q)_y = S_B(Q, \nabla B) + S_F(Q) + S_C(Q).$$

The Jacobians of the flux functions, J_F and J_G, have the eigenvalues $u, u \pm \sqrt{gh}$ and $v, v \pm \sqrt{gh}$, respectively. Thus the shallow water equations in two spatial dimensions is a hyperbolic system, with strict hyperbolicity when $\sqrt{gh} > 0 \Leftrightarrow h > 0$, called a *wet bed.* See [17, 29] for a more thorough analysis.

In this work, most of the attention will be given to the bed slope source term, and one particular field of interest is when $Q_t = 0$ and (2.5) admits the stationary solution

$$\begin{aligned} u(x,y,t) &= 0 \\ v(x,y,t) &= 0 \\ h(x,y,t) + B(x,y) &= \text{const}, \end{aligned} \tag{2.8}$$

at which point, the water is at rest and the bed slope source is balancing out the flux terms.

When the system is in the stationary state (2.8), it should be realized that the water *surface*, $h + B$, is constant, not the water *height*, h, which motivates the variable change

$$w(x,y,t) = h(x,y,t) + B(x,y),$$

after which, the shallow water equations, on the difference form, read

$$Q_t + F_x + G_y = S,$$

with

$$\begin{aligned} Q &= \begin{bmatrix} w \\ hu \\ hv \end{bmatrix}, \\ F &= \begin{bmatrix} hu \\ \frac{(hu)^2}{w-B} + \frac{1}{2}g(w-B)^2 \\ \frac{(hu)(hv)}{w-B} \end{bmatrix}, \quad G = \begin{bmatrix} hv \\ \frac{(hu)(hv)}{w-B} \\ \frac{(hv)^2}{w-B} + \frac{1}{2}g(w-B)^2 \end{bmatrix} \\ S &= \begin{bmatrix} 0 \\ -g(w-B)B_x \\ -g(w-B)B_y \end{bmatrix}. \end{aligned} \tag{2.9}$$

Note that with this variable change, the Jacobians of the system have the eigenvalues u and $u \pm \sqrt{g(w-B)}$ for J_F and v and $v \pm \sqrt{g(w-B)}$ for J_G.

2.3. Finite-volume methods

Non-linear hyperbolic conservation laws can be challenging to treat numerically, particularly since the equations often admit discontinuous solutions, even from smooth initial conditions. The numerical methods need to behave nicely around discontinuities, yet at the same time converge as rapidly as possible in smooth areas. In particular we want to avoid spurious oscillations near discontinuities. We will consider a class of methods which is called *shock-capturing*, seeing as they adapt automatically to discontinuous behaviour, as opposed to *shock-fitting*, also called front-tracking, methods, which tracks the position of the discontinuities and explicitly introduce jumps in the solution.

Conservation laws are commonly solved by finite-volume methods [17,29]. Instead of operating with Taylor expansions using function values at the grid points, as we would have done in a finite-differences setting, we divide the grid into small control 'volumes', or control cells, and then apply the actual conservation law (2.1) to each cell. By this, we are able to solve the more general integral form of the conservation law, rather than the resulting PDE, thus obtaining the weak solution of the problem. Since (2.1) holds for any subset, $K \subset \Omega$, it holds for any cell in some defined grid.

The concepts of the finite-volume method, are usually explained in one spatial dimension, knowing that they are easily extended to $\mathbb{R}^n$. Let us, however, do the introduction in two spatial dimensions, seeing as that is the framework needed for the shallow water equations. In two dimensions, for a uniform grid with cells

$$I_{ij} = \left[x_i - \frac{\Delta x}{2}\,,\, x_i + \frac{\Delta x}{2}\right] \times \left[y_j - \frac{\Delta y}{2}\,,\, y_j + \frac{\Delta y}{2}\right],$$

the cell averages of a variable Q, are given by

$$Q_{ij}^n = \frac{1}{\Delta x \Delta y} \int_{x-\Delta x/2}^{x+\Delta x/2} \int_{y-\Delta y/2}^{y+\Delta y/2} Q(x, y, t_n)\, \mathrm{d}y\, \mathrm{d}x,$$

at the given time $t = t_n$.

The concept is to evolve cell averages of the conserved variables, Q, by applying (2.1) to each grid cell. A semi-discrete finite-volume discretization of the general hyperbolic conservation law in two spatial dimensions reads

$$\frac{\mathrm{d}}{\mathrm{d}t} Q_{i,j}(t) = -\frac{H^x_{i+\frac{1}{2},j}(t) - H^x_{i-\frac{1}{2},j}(t)}{\Delta x} - \frac{H^y_{i,j+\frac{1}{2}}(t) - H^y_{i,j-\frac{1}{2}}(t)}{\Delta y} + S_{i,j}(t), \quad (2.10)$$

where $Q_{ij}(t)$ and $S_{ij}(t)$ are to be interpreted as cell averages. The notation $H^x_{i+1/2,j}(t)$ is taken to be some numerical approximation of the flux integral on the eastern cell interface,

$$H^x_{i+\frac{1}{2},j}(t) \approx \frac{1}{\Delta y}\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} F(Q(x_{i+\frac{1}{2}},y,t))\,\mathrm{d}y,$$

and likewise for the other three interfaces, with the assumption that the flux function only depends on the conserved variables, which indeed is the case in the shallow water equations.

To solve the semi-discrete problem (2.10), we need an appropriate ODE solver, see Section 3.3, restricted by a CFL condition - named after Richard Courant, Kurt Friedrichs, and Hans Lewy - which determines an upper bound for Δt and serves as a necessary, but not sufficient, condition for convergence. The CFL condition ensures that the information does not 'travel too fast'. In [17] it is defined as

Definition 2.1. ***(CFL condition).*** *A numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the PDE, at least in the limit as Δt and Δx go to zero.*

If we consider a cell interface in the uniform grid, at which the flux depends only on the two cells that share the interface, then if the time step is too large, we allow information from more cells to cross the interface, and thus the numerical flux formulas do not match the physical situation.

For hyperbolic problems in two spatial dimensions on the form (2.4), the CFL condition reads

$$\Delta t \leq C_{cfl}\min\left\{\Delta x/\max_{\Omega}\left|\lambda_k^F(Q)\right|,\, \Delta y/\max_{\Omega}\left|\lambda_k^G(Q)\right|\right\}, \tag{2.11}$$

where the constant C_{cfl} depends on the ODE solver.

By applying the ODE solver, we essentially end up with a fully discrete scheme. Exemplified by Euler's method, the scheme, which evolves the cell averages from $t = t_n$ to t_{n+1}, reads

$$Q^{n+1}_{i,j} = Q^n_{i,j} - \Delta t\frac{H^{x,n}_{i+\frac{1}{2},j} - H^{x,n}_{i-\frac{1}{2},j}}{\Delta x} - \Delta t\frac{H^{y,n}_{i,j+\frac{1}{2}} - H^{y,n}_{i,j-\frac{1}{2}}}{\Delta y} + S^n_{i,j}, \tag{2.12}$$

where the numerical fluxes are approximations of the respective interface fluxes integrated over Δt. If we again consider the eastern interface, we have

$$H^{x,n}_{i+\frac{1}{2},j} \approx \frac{1}{\Delta y}\int_{t_n}^{t_{n+1}}\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} F(Q(x_{i+\frac{1}{2}}, y, \tau))\,\mathrm{d}y\,\mathrm{d}\tau.$$

The integral over t is usually approximated by only considering function values at $t = t_n$. We will come back to the flux integrals in Section 3.2.

The Lax-Friedrich scheme

With this framework, we are able to establish the classical first-order Lax-Friedrich scheme. Given cell averages Q^n_{ij}, and if we only consider the homogeneous case, the evolution of Q, using (2.12) is fully determined by the numerical fluxes. The Lax-Friedrich scheme assumes $Q(x, y, t)$ to be piecewise constant, so that $Q(x, y, t) = Q^n_{ij}$ for all $x, y \in I_{ij}$ and $t \in [t^n, t^n + \Delta t)$. Consequently, at each cell interface, Q is discontinuous. We take the cell interface fluxes to be the arithmetic average of the fluxes evaluated using the cell averages in each of the two adjacent cells, e.g.

$$H^{x,n}_{i+\frac{1}{2},j} = \frac{1}{2}\Big(F(Q^n_{i,j}) + F(Q^n_{i+1,j})\Big)$$

on the eastern cell interface.

Inserting this into (2.12), we get

$$Q^{n+1}_{i,j} = Q^n_{i,j} - \frac{1}{2}\Delta t\frac{F(Q^n_{i+1,j}) - F(Q^n_{i,j})}{\Delta x} - \frac{1}{2}\Delta t\frac{G(Q^n_{i,j+1}) - G(Q^n_{i,j-1})}{\Delta y}.$$

As it turns out, to ensure stability, we need to add artificial diffusion, resulting in the Lax-Friedrich scheme in two dimensions

$$\begin{aligned} Q^{n+1}_{i,j} =&\frac{1}{4}\Big(Q^n_{i+1,j} + Q^n_{i-1,j} + Q^n_{i,j+1} + Q^n_{i,j-1}\Big) \\ &- \frac{1}{2}\Delta t\frac{F(Q^n_{i+1,j}) - F(Q^n_{i,j})}{\Delta x} - \frac{1}{2}\Delta t\frac{G(Q^n_{i,j+1}) - G(Q^n_{i,j-1})}{\Delta y}. \end{aligned} \tag{2.13}$$

3. High-resolution methods

The classical finite-volume schemes, such as the Lax-Friedrich scheme (2.13), are typically either highly dissipative or oscillating around discontinuities. High-resolution methods [9,10,17,29] are Godunov-type shock-capturing finite-volume methods, tailored to cope with these issues, by providing high-order accuracy in smooth parts of the solution, while still remaining accurate and non-oscillating around discontinuities. One way of building up a high-resolution method, is to apply the so-called REA algorithm - Reconstruct, Evolve, Average - which is a three-step algorithm to advance the solution of the semi-discrete conservation law (2.10) one time step. It is commonly defined as

1. *Reconstruct.* From cell averages at time $t = t_n$, Q_{ij}^n, reconstruct a piece-wise polynomial function

   $$\hat{Q}_{ij}(x, y, t_n) \in \mathbb{P}^k, \quad x, y \in I_{ij},$$

   satisfying

   $$Q_{ij}^n = \frac{1}{\Delta x \Delta y} \int_{I_{ij}} \hat{Q}_{ij}(x, y, t_n) \, \mathrm{d}x \, \mathrm{d}y.$$

2. *Evolve.* Evolve the reconstructed polynomials, $\hat{Q}_{ij}(x, y, t_n)$ according to the conservation law, obtaining $\hat{Q}_{ij}(x, y, t_{n+1})$.

3. *Average.* Average the polynomials over each cell,

   $$Q_{ij}^{n+1} = \frac{1}{\Delta x \Delta y} \int_{I_{ij}} \hat{Q}_{ij}(x, y, t_{n+1}) \, \mathrm{d}x \, \mathrm{d}y.$$

In the actual implementations, however, the averaging typically is a direct result of the evolution process, which is done by some Runge-Kutta method, and it would possibly be more instructive to define the REA algorithm as in Algorithm 1. The latter definition fits better with the implementations used in this work. Commonly the solution is evolved in time by the means of an explicit Runge-Kutta method, which means that we approximate the fluxes during one time step purely based on the function values at $t = t_n$, which means that we do not actually evolve the reconstructed polynomials, as suggested by the regular REA definition, but rather insert the flux and source approximations at $t = t_n$ into the right-hand side of the conservation law (2.10) and evolve the cell averages by the Runge-Kutta method. Hence the averaging part of the REA algorithm is somewhat disguised.

In the following sections we explain the numerical aspects of high-resolution methods more in detail.

Algorithm 1 The REA algorithm

```
Given cell averages, $Q^n_{ij}$.
Reconstruct a polynomial from cell averages, $Q^n_{ij} \longmapsto \hat{Q}(x, y, t_n)$.
Compute flux and source terms at $t = t_n$ using the reconstructed polynomial, $\hat{Q}(x, y, t_n) \longmapsto F^n, G^n, S^n$.
Evolve solution using a Runge-Kutta method, $Q^n, F^n, G^n, S^n \longmapsto Q^{n+1}$.
```

3.1. Reconstruction from cell averages

The first step of a high-resolution method is the procedure in which we use the given cell averages, Q_i, at each time step to obtain point values of the conserved variables. We refer to this step as the *reconstruction*. We seek a polynomial, $\hat{Q}_i$, defined in each cell, which is to be evaluated at the integration points used in the flux interface integrals. Note here that we refer to a one-dimensional cell for simplicity. The simplest, first-order, approach, would be to set $\hat{Q}_i = Q_i$, meaning that we approximate the conserved variables by a piecewise constant function. This is in fact what we did in the Lax-Friedrich scheme, and already in this simple setting, we are introduced to one of the key consequences of the reconstruction step. Since, in general, we have $\hat{Q}_i = Q_i \neq Q_{i+1} = \hat{Q}_{i+1}$, we realize that at the cell interfaces, the reconstructed function values will, in general, be different depending on which of the two adjacent cell reconstructions we are evaluating. Exemplified at the interface $x = x_{i+1/2}$, these two possible one-sided values of $Q(x_{i+1/2})$ have various notations in the literature, such as

$$\hat{Q}_i(x_{i+1/2}) = \hat{Q}(x_{i+1/2} - 0) = Q^-_{i+1/2} = Q^L_{i+1/2} = Q^E_i,$$

for the $\hat{Q}_i$ reconstruction and correspondingly

$$\hat{Q}_{i+1}(x_{i+1/2}) = \hat{Q}(x_{i+1/2} + 0) = Q^+_{i+1/2} = Q^R_{i+1/2} = Q^W_{i+1},$$

for the $\hat{Q}_{i+1}$ reconstruction. Here, $R/L\,, +/-$ means 'slightly to the right/left of the interface', whereas E and W refers to the eastern and western interface relative to the cell specified by the subscript. With that in mind these notation variants should be self-explanatory. Exploiting these one-sided reconstruction values at the cell interfaces, is a crucial part of the flux computation.

In the first-order setting, the reconstruction step is not particularly interesting. In a second-order reconstruction, however, the term gives more sense. Here we are approximating the conserved variables by piecewise linear functions, i.e. polynomials $p \in \mathbb{P}^1$, a reconstruction procedure which is quite popular due to its improved accuracy, without complicating the schemes too much. Several known schemes, such as [1, 2, 12, 14], just too name a few, are based on linear reconstruction.

Now, let us present one of the main attributes we embrace in a reconstruction method. Since hyberbolic problems often involves solutions with discontinuities, we want our reconstruction to minimize spurious oscillations near the discontinuous points and to be total variational diminishing (TVD). The first property is rather self-explanatory. The latter is defined as

$$\mathrm{TV}(Q(\cdot,\tau)) \leq \mathrm{TV}(Q(\cdot,t)), \quad \forall t \leq \tau, \tag{3.1}$$

where

$$\mathrm{TV}(v) = \limsup_{h\to 0} \frac{1}{h}\int |v(x) - v(x-h)\,\mathrm{d}x.$$

A high-order reconstruction should possess these properties and converge with maximal accuracy in smooth areas, but at the same time behave 'nicely' around discontinuities. This is done with some sort of a non-linear limiting function. To obtain second-order reconstructions we build up a polynomial $p_i \in \mathbb{P}^1$, for each cell I_i. Given the cell averages at time $t = t_n$, for some approximation, s_i, of the first derivative, we can build up the polynomial

$$p_i(x, t_n) = \hat{Q}_i(x, t_n) = Q^n_i + s_i(x - x_i), \quad x \in [x_{i-1/2}, x_{i+1/2}].$$

The crucial part of the reconstruction step is the choice of s_i. From basic theory of finite-differences [15], we know there are three obvious choices of two-point stencils for approximating the first derivative, namely

$$s_i^- = \frac{Q_i - Q_{i-1}}{\Delta x_i}, \quad s_i^c = \frac{Q_{i+1} - Q_{i-1}}{2\Delta x_i}, \quad s_i^+ = \frac{Q_{i+1} - Q_i}{\Delta x_i}.$$

Which one to choose will depend on the behaviour of the solution. We want the stencil that causes the least oscillations. This procedure has several possible solutions. One is the common minmod limiter

$$\operatorname{minmod}(a_1, a_2, ...) = \begin{cases} \min_k\{a_k\} & \text{if } a_k > 0\,\forall k \\ \max_k\{a_k\} & \text{if } a_k < 0\,\forall k \\ 0 & \text{else} \end{cases}. \tag{3.2}$$

We set

$$s_i = \operatorname{minmod}\left(\theta s_i^-, s_i^c, \theta s_i^+\right), \quad \theta \in [1, 2],$$

The minmod function is one example of a non-linear *limiter function*. Other examples include the *superbee* or the *modified minmod* limiter [9].

While the minmod function serves as an instructive example of a key element in the reconstruction step, it is of little relevance to this thesis, where the main objective is to implement schemes of higher order than 2, which will involve the use of a rather different reconstruction technique, with its corresponding non-linear limiting functions, to be presented in Section 3.6.

So far, the description of the reconstruction procedure has been purely general. For the shallow water equations (2.5) we encounter the issue of having a bottom topography, B, in addition to the conserved variables, Q. The gradient of B is present in the bed slope source term and needs to be approximated numerically, and when using the common variable change, $w = h + B$ (2.9), B itself is included in the flux integrals. The bottom topography is often known as point values at the cell corners, as opposed to cell averages, thus applying the reconstruction procedure to B would not make sense. However, to compute the derivatives, we need some interpolation procedure based on the cell corners. In a second-order scheme, this is best done using bilinear interpolation, and we obtain, for cell I_{ij},

$$\hat{B}_{ij}(x,y) = B_{i-\frac{1}{2},j-\frac{1}{2}} + \left(B_{i+\frac{1}{2},j-\frac{1}{2}} - B_{i-\frac{1}{2},j-\frac{1}{2}}\right)\frac{x - x_{i-\frac{1}{2}}}{\Delta x}$$
$$+ \left(B_{i-\frac{1}{2},j+\frac{1}{2}} - B_{i-\frac{1}{2},j-\frac{1}{2}}\right)\frac{y - y_{j-\frac{1}{2}}}{\Delta y}$$
$$+ \left(B_{i+\frac{1}{2},j+\frac{1}{2}} - B_{i+\frac{1}{2},j-\frac{1}{2}} - B_{i-\frac{1}{2},j+\frac{1}{2}} + B_{i-\frac{1}{2},j-\frac{1}{2}}\right)\frac{(x - x_{i-\frac{1}{2}})(y - y_{j-\frac{1}{2}})}{\Delta x\,\Delta y}. \tag{3.3}$$

3.2. The Riemann problem and flux integral

A central component of any high-resolution scheme is to determine the flux approximations, H, involved in (2.10).

At the cell interfaces, the two one-sided reconstructions of Q are in general not equal, and determining the flux integral is similar to locally solving the Riemann problem,

$$Q_t + F(Q)_x = 0, \quad Q(x,0) = \begin{cases} Q^L, \ x < 0 \\ Q^R, \ x > 0, \end{cases}$$

where Q^L and Q^R are the left and right-hand sided reconstructed values of Q at the cell interface. The Riemann problem has similarity solutions $q(x,t) = V(x/t)$, called the Riemann fan, and we call a method *upwind* if it aims to solve the Riemann problem, either exact or by an approximate solver, in order to compute the interface fluxes. In contrast, a *central* scheme, does not apply any Riemann solvers.

Examples of central fluxes include the classical Lax-Friedrich flux,

$$H^{LF} = \frac{1}{2}\left[F^L + F^R\right] - \frac{1}{2}\frac{\Delta t}{\Delta x}\left[Q^R - Q^L\right], \tag{3.4}$$

where the numerical fluxes are defined by $F^L = F(Q^L)$, $F^R = F(Q^R)$. Inserting (3.4) into (2.12), yields the Lax-Friedrich scheme (2.13).

The schemes developed in this work, belongs to a class called central-upwind methods, as they keep the simplicity of the central methods, while including local, one-sided, speeds of propagation, i.e. eigenvalues of J_F, J_G, in the flux computation, aiming to retrieve some of the accuracy of the upwind methods. The preferred choice of flux formula seems to be the central-upwind flux, which was introduced by Kurganov, Noelle and Petrova in [13]. This flux approximation incorporates estimates of the Riemann fan at the cell interface. To this end we define

$$\begin{aligned} a^+ &= \max_{Q\in\{Q^L,Q^R\}} \big(\lambda_m(Q),\, 0\big) \\ a^- &= \min_{Q\in\{Q^L,Q^R\}} \big(\lambda_1(Q),\, 0\big), \end{aligned} \tag{3.5}$$

where $\lambda_1 \leq ... \leq \lambda_m$ are the eigenvalues of F. Hence a^+ is the biggest and a^- the smallest eigenvalue at any side of the cell interface and corresponds to the one-sided wave speeds. The central-upwind flux is then defined by

$$H^{CUW} = \frac{a^+F^L - a^-F^R}{a^+ - a^-} + \frac{a^+a^-}{a^+ - a^-}(Q^R - Q^L). \tag{3.6}$$

In two dimensions, the one-sided values F^L, F^R, Q^L, Q^R, are computed using the one-sided reconstructed point values in some quadrature rule at the 'plus' and 'minus' cell interface. Using e.g. only the midpoint, we obtain a second-order approximation, which will integrate linear functions exactly. Having established the reconstructed polynomial, $\hat{Q}$, it is possible to choose a quadrature rule such that the reconstructed functions are integrated exactly, meaning that the only numerical approximation present is in the reconstruction step.

For a second-order reconstruction of Q it is customary to use the midpoint rule, which transformed to the interval $I = [-1, 1]$ reads

$$\int_{-1}^{1} f(x(\xi))\,\mathrm{d}\xi \approx f(x(0)), \tag{3.7}$$

which for the eastern boundary of cell (i, j), using the 'minus' reconstruction $\hat{Q}_{i,j}$, give $F^L_{i+1/2,j} = F(\hat{Q}_{i,j}(x_{i+1/2}, y_j))$, $Q^L_{i+1/2,j} = \hat{Q}_{i,j}(x_{i+1/2}, y_j)$.

For reconstructions of higher order than two, we would also like high-order approximations of the flux integrals. Using Simpson's method we obtain third-order accuracy, but we need three integrations points, which can be costly to calculate. A more preferable method would be the Gaussian quadrature,

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} f(x(\xi))\,\mathrm{d}\xi \approx \frac{1}{2}\left(f(x(-\alpha)) + f(x(\alpha))\right), \quad \alpha = \frac{1}{2\sqrt{3}}, \tag{3.8}$$

which gives a fourth order approximation using only two points. The quadrature rules (3.7) and (3.8) are the methods of choice for second-order and high-order high-resolution schemes respectively. Gaussian quadrature is cheaper and more accurate then e.g. Simpson's method, while the midpoint rule provide the same accuracy as the trapezoidal rule, but with only one integration point.

When applying quadratures to the one-sided flux integrals, these quadrature points should also be included in the calculation of the smallest and biggest eigenvalues (3.5), such that the domain in which we seek the minimum is extended to e.g. $Q \in \{Q^L_{-\alpha}j,\, Q^L_{+\alpha},\, Q^R_{-\alpha},\, Q^R_{+\alpha}\}$ in the case of Gaussian quadrature.

3.3. Total variation diminishing Runge-Kutta methods

A conservation law in its semi-discrete form (2.10), is an equation on the form

$$\frac{\mathrm{d}}{\mathrm{d}t}Q = f(Q),$$

where we have used the notation $f(Q)$ which is commonplace in the literature on first-order ODEs.

After computing the reconstruction and flux integrals, we are left with the discrete ODE formulation of the system, denoted

$$\frac{\mathrm{d}}{\mathrm{d}t}Q_{ij} = L_{ij}(Q).$$

The operator L_{ij} is given, for each cell, by the right-hand side of (2.10). With the notation $L_{ij}(Q)$, we have assumed that the problem is not explicitly dependant on time, which clearly is true for the shallow water equations. An explicit ODE solver then is on the form

$$\begin{aligned} Q^{(i)} &= Q^{(0)} + \Delta t \sum_{k=0}^{i-1} c_{ik} L(Q^{(k)}), \quad i = 1, 2, ..., m \\ Q^{(0)} &= Q^{(n)}, \quad Q^{(m)} = Q^{(n+1)} \end{aligned} \tag{3.9}$$

where L is defined by the right-hand side of (2.10). In (3.9) we have ignored the cell indices for better readability.

Typically we would also prefer that the ODE solver is TVD (3.1). An overview of TVD Runge-Kutta methods can be found in [8,25,26], where optimal TVD Runge-Kutta methods, with respect to the TVD property, are derived. We will state the main results here.

The suggested optimal first-order Runge-Kutta method corresponds to the regular Euler's method, and is TVD under CFL condition $C_{cfl} = 1$. The optimal second-order TVD Runge-Kutta reads

$$\begin{aligned} Q^{(1)} &= Q^{(n)} + \Delta L(Q^{(n)}) \\ Q^{(n+1)} &= \frac{1}{2}Q^{(n)} + \frac{1}{2}Q^{(1)} + \frac{1}{2}\Delta L(Q^{(1)}), \end{aligned} \tag{3.10}$$

which is TVD in one spatial dimension under CFL condition $C_{cfl} = 1$, and also equivalent to the regular second-order Runge-Kutta, i.e. Heun's method or modified Euler method. In two spatial dimensions the CFL condition is stricter, $C_{cfl} = 0.25$ [7].

The third-order TVD Runge-Kutta is

$$\begin{aligned} Q^{(1)} &= Q^{(n)} + \Delta L(Q^{(n)}) \\ Q^{(2)} &= \frac{3}{4}Q^{(n)} + \frac{1}{4}Q^{(1)} + \frac{1}{4}\Delta L(Q^{(1)}) \\ Q^{(n+1)} &= \frac{1}{3}Q^{(n)} + \frac{2}{3}Q^{(2)} + \frac{2}{3}\Delta L(Q^{(2)}), \end{aligned} \tag{3.11}$$

under CFL condition $C_{cfl} = 1$ in one dimension and $C_{cfl} = 0.5$ in two dimension [28]. Contrary to the second-order case, (3.11) is not equivalent to any classical Runge-Kutta method. Note how, for the Runge-Kutta methods (3.10) and (3.11), the global memory storage load is rather low, seeing as we, at each time sub step, $i \to i + 1$, need only store three values, $Q^{(n)}$, $Q^{(i)}$ and $L(Q^{(i)})$.

Fourth-order TVD Runge-Kutta methods, however, are not as easy to derive, as they include using the adjoint of the operator L [8,25,26], and will not be considered here. Some schemes are using the classical fourth-order Runge-Kutta method [18,24]. Another possibility is to use a so-called Strong Stability Preserving (SSP) Runge-Kutta method [27]. Unfortunately, for a fourth-order Runge-Kutta method, the global memory load is higher, seeing as we typically need to store more than three values at each time sub step, in contrast to the low-order methods.

3.4. Source term quadrature

In non-homogeneous conservation laws, also the source terms need to be approximated numerically. It is of course possible to use a standard quadrature such as Gaussian quadrature of any order, or Simpson's rule, but often we are particularly interested in the system's behaviour close to stationary state, i.e. when $Q_t = 0$. Analytically the flux integrals will in stationary state be canceled out by the bed slope source term, in (2.1). To model the system accurately in, or near, stationary state, we therefore want to ensure that also the discretized system (2.10) has this property, meaning that, in two spatial dimensions, we want

$$Q_t = 0 \implies S_{i,j} = \frac{H^x_{i+\frac{1}{2},j} - H^x_{i-\frac{1}{2},j}}{\Delta x} + \frac{H^y_{i,j+\frac{1}{2}} - H^y_{i,j-\frac{1}{2}}}{\Delta y}, \tag{3.12}$$

for each cell and each time step, which can be achieved by a special quadrature rule for the source term, which we will present in Section 5.1 for the shallow water equations. A numerical method satisfying (3.12), is said to be *well-balanced.*

As with the one-sided flux integrals, it is possible to choose a quadrature of any order for the source term as long as the reconstructed polynomial is defined for the entire cell. Special care should be given to make sure the reconstruction procedure and the flux and source quadratures all produce about the same level of accuracy.

3.5. High-order schemes

To obtain truly high-order high-resolution schemes, we need to increase the order of all the methods involved in the REA procedure, both the reconstruction, the one-sided flux integrals, the source term quadrature and the Runge-Kutta method. For the shallow water equations, also the interpolation of B need to be of higher order. We have already presented the fourth-order accurate Gaussian quadrature for the one-sided flux integrals, which is sufficient in most cases. Also high-order Runge-Kutta schemes are covered. A special fourth-order source term quadrature for the shallow water equations will be derived in Section 5.1.

Inspired by [18], let us first introduce a proper notation to state the order of the high-resolution schemes, formalized in the following definition:

Definition 3.1. ***(Numerical order)****. The formal order of a high-resolution finite-volume scheme, using the REA algorithm is given by the quadruplet*

$$(t, r, f, s),$$

where

t : *Order of the time integration*

r : *Order of the reconstruction*

f : *Order of the quadrature in the flux integral*

s : *Order of the quadrature in the source term integral,*

The formal total order of the scheme is given by

$$min\{t, r, f, s\}.$$

In the coming discussions, we will refer to a scheme as a 'scheme of order (t, r, f, s)' or simply a '(t, r, f, s) scheme'.

Typically, the time integration will be the limiting factor, $t \leq r, f, s$, which might lead to schemes that yield high order of accuracy close to the start time, while the accuracy level tends towards order t as the system develops in time.

3.6. Weighted Essentially Non-Oscillating reconstructions

Let us now again turn to the reconstruction step of the REA algorithm. In the second-order reconstruction we used a three-point stencil, (Q_{i-1}, Q_i, Q_{i+1}), from which we picked the least oscillating two-point stencil to compute the derivatives in the x direction, and likewise in the y direction. In total we used a five-point stencil to build up the complete reconstructed polynomial, $p_i \in \mathbb{P}^1$. In essence, we create reconstructions of higher order by adding more points to the stencil.

As with second-order schemes, there will be spurious oscillations near discontinuities of the solution if high-order schemes are applied naively. One remedy is to invoke a method whose concept is similar to the non-linear limiter function (e.g. the minmod function) for a second-order scheme. A widely cited paper by Harten [11] introduced the concept of essentially non-oscillating (ENO) reconstructions, where we choose the least oscillating stencil, of as high order as possible, at hand. This approach was later extended to the weighted essentially non-oscillatory (WENO) procedure, in which we use a convex combination of all feasible stencils, each weighted by non-linear weights. In the second-order setting that would mean using a weighted linear combination of s^-, s^c and s^+, as opposed to choosing only one.

In [9] there is a brief introduction to WENO reconstruction, including examples of two different approaches for a multidimensional WENO scheme, the *genuinely multidimensional* one, and the *dimension-by-dimension* approach. The genuinely multidimensional reconstruction follows the same idea as in the second-order case, where a complete polynomial, valid for an entire cell, is created. For a third-order reconstruction, in two spatial dimensions, a non-linear combination of a standard nine-point stencil for the second derivative and two-point stencils for the first derivatives, is used in order to pick a stencil of third-order when possible, and of lower order close to discontinuities. The approach is highly instructive, intuitive and worthwhile reading, but it is computationally costly to store all the coefficients for high-order polynomials. In the dimension-by-dimension approach, however, we are computing the function values at the integration points directly, without the use of a complete polynomial, valid for the whole cell, which is more appealing for reconstructions of more than order three. The dimension-by-dimension approach will therefore be the focus of this thesis.

A stand-alone theoretical background of WENO reconstructions is found in [25], and we will state the essentials here. The details of the two-dimensional scheme suited for solving the shallow water equations, are taken from [28], where an extension to three dimensions also is provided.

In the dimension-by-dimension WENO reconstruction, we perform one reconstruction, referred to as a 'sweep', in each spacial dimension, and design the method such that it is of any wanted order at the chosen integration points. In general, in a reconstruction of order $2r - 1$, each sweep computes a weighted convex combination of r polynomials, $p_i \in \mathbb{P}^{r-1}$. Each polynomial is then of order r. Note that the literature is not consistent in the choice of indexes and denotations, so to avoid confusion we emphasize that here, and in the following, we will denote the order by r. Given that we wish to apply a fourth-order Gaussian quadrature to the one-sided flux integrals, it is plausible to choose $r = 3$, which yields a fifth-order WENO reconstruction. Increasing the accuracy to, say seventh-order, $r = 4$, would not make sense without also adding an integration point, resulting in a sixth-order Gaussian quadrature.

In two dimensions, the first sweep is designed to reconstruct line averages from the given cell averages. A sweep along the x axis produces the line average on a line along the y axis, and vice versa. By doing one more sweep, in the spatial direction along the current line, we obtain the function values at the integration points situated on the line.

In the fourth-order Gaussian quadrature, we seek the function values of Q at the eight points

$$(x_{i+\frac{1}{2}}, y_{j\pm\alpha}), (x_{i-\frac{1}{2}}, y_{j\pm\alpha}), (x_{i\pm\alpha}, y_{j+\frac{1}{2}}), (x_{i\pm\alpha}, y_{j-\frac{1}{2}}), \ \alpha = \frac{1}{2\sqrt{3}} \tag{3.13}$$

at the cell interfaces. Additionally, in order to apply the fourth-order accurate bed slope source term quadrature (5.6)-(5.7), we will also need the four interior points

$$(x_i, y_{j\pm\alpha}), (x_{i\pm\alpha}, y_j), \ \alpha = \frac{1}{2\sqrt{3}}, \tag{3.14}$$

which we will come back to in Section 5.1. This leaves us with a choice, as illustrated in Figure 3.1, either to reconstruct six line averages with two integration points each, or to reconstruct four line averages with three integration points each. We will shortly see that the actual reconstructions are oblivious to whether they compute line averages or point values. If the input is a cell average, the output will be a line average, and if the input is a line average, the output will be a point value. The only mathematical difference between the different reconstruction sweeps, is if the offset from the cell centre of the line or point is $\pm\alpha$, 0 or $\pm 1/2$. Thus, whether we choose the approach in Figure 3.1a or the one in Figure 3.1b, has no impact on the formulas presented below.

For each cell, I_{ij}, at a time step, $t = t_s$ we are given the cell average

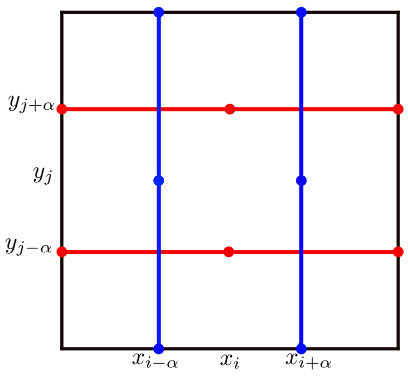
$$Q_{ij} = \frac{1}{\Delta x \Delta y} \int_{I_{ij}} Q(x, y)\, \mathrm{d}x\mathrm{d}y.$$

We then wish to perform a one-dimensional WENO reconstruction to reduce the cell average to a line average on a chosen line ℓ. If we start by a sweep in the x direction, we end up with the average on the vertical line $\ell = (x_{i+a}, y)$, for a fixed,

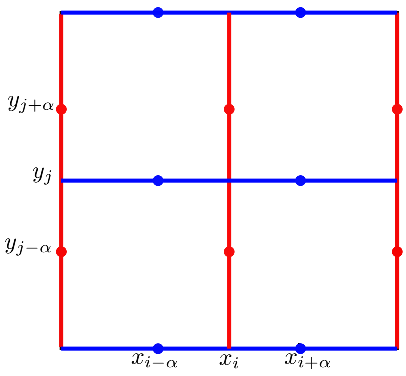
$$Q_{i+a,j} \approx \frac{1}{\Delta y} \int_{y_{j-1/2}}^{y_{j+1/2}} Q(x_{i+a}, y)\, \mathrm{d}y,$$

whereas if we start by a sweep in the y direction we obtain the averages on the horizontal line $\ell = (x, y_{j+a})$,

$$Q_{i,j+a} \approx \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} Q(x, y_{j+a})\, \mathrm{d}x,$$



(a) Interior lines



(b) Interface lines

Figure 3.1: Illustration of the two different approaches to do a WENO reconstruction of the same points on a cell in two dimensions.

From these averages we then perform a sweep in the direction parallel to the current line to obtain the values of the reconstructed function at the required integration points (3.13)-(3.14).

In each of the one-dimensional sweeps, we seek to find the reconstruction of Q_{ij},

$$\hat{Q}_{ij}(x, y) = \sum_{k=0}^{r-1} \omega_{ij}^k P_{ij}^k(x, y), \tag{3.15}$$

where Q_{ij} can be either a line average or a cell average. The coefficients ω_{ij}^k are responsible for giving more weight to the best suited stencils, and the polynomials, P_{ij}^k, are solely determined from the grid.

Each polynomial, P_{ij}^k, requires a stencil consisting of r grid cells in the direction of the current sweep, so that the complete stencil for $\hat{Q}_{ij}(x, y)$ covers $2r - 1$ cells. The polynomials are chosen based on a linear combination of the values of Q_{ij} in each of the r grid cells. Because of the dimension-by-dimension approach, we might as well simply consider a one-dimensional problem. We want the polynomials to be such that at any integration point, x_ξ, they are given as a linear combination of the function values, i.e.

$$P_i^k(x_\xi) = \sum_{j=0}^{r-1} c_i^{k,j} Q(x_{\xi-k+j}), \quad x_\xi \in [x_{i-1/2}, x_{i+1/2}] = I_i$$

Thus, all that is needed is to find the coefficients $c_i^{j,k}$. Notice that j and k are local indexes relative to i. The same stencil can give rise to different reconstruction polynomials depending on where the stencil is located compared to the integration point x_ξ. For I_i, and $r = 3$, i.e. a fifth-order WENO reconstruction, we have the total stencil

$$S_i = \{Q_{i-2}, Q_{i-1}, Q_i, Q_{i+1}, Q_{i+2}\},$$

which consists of three possible three-point stencils,

$$S_i^0 = \{Q_{i-2}, Q_{i-1}, Q_i\},\ S_i^1 = \{Q_{i-1}, Q_i, Q_{i+1}\},\ S_i^2 = \{Q_i, Q_{i+1}, Q_{i+2}\},$$

and clearly, in general, the reconstructed values of $\hat{Q}_i$ will be different in each of the three stencils.

The coefficients, $c_i^{k,j}$ are, on a uniform grid, given by the formula [25]:

$$c_i^{k,j} = \sum_{m=j+1}^{r} \frac{\sum\limits_{\substack{l=0 \\ l\neq m}}^{r} \prod\limits_{\substack{q=0 \\ q\neq m,l}}^{r} (r - q + \frac{1}{2} + \xi)}{\prod\limits_{\substack{l=0 \\ l\neq m}}^{r} (m - l)}. \tag{3.16}$$

For the reconstructions at $x_{i\pm1/2}, x_{i\pm\alpha}$ the coefficients are given explicitly in the literature. For a cell I_i the WENO reconstruction polynomials at the two interfaces read

$$\begin{aligned} \hat{Q}^R_{i-\frac{1}{2}} = \hat{Q}_i(x_{i-\frac{1}{2}}) = \frac{1}{6}\Omega_0(11Q_i - 7Q_{i+1} + 2Q_{i+2}) \\ + \frac{1}{6}\omega_1(2Q_{i-1} + 5Q_i - Q_{i+1}) + \frac{1}{6}\omega_2(-Q_{i-2} + 5Q_{i-1} + 2Q_i) \end{aligned} \tag{3.17}$$

and

$$\begin{aligned} \hat{Q}^L_{i+\frac{1}{2}} = \hat{Q}_i(x_{i+\frac{1}{2}}) = \frac{1}{6}\omega_0(2Q_i + 5Q_{i+1} - Q_{i+2}) \\ + \frac{1}{6}\omega_1(-Q_{i-1} + 5Q_i + 2Q_{i+1}) + \frac{1}{6}\omega_2(2Q_{i-2} - 7Q_{i-1} + 11Q_i), \end{aligned} \tag{3.18}$$

where R, L denotes right and left reconstructed value relative to the flux interface. Again, $\hat{Q}^L_{i+1/2}$ and $\hat{Q}^R_{i-1/2}$ can be either one-sided line averages of the interface or one-sided point values.

The line averages or point values at $x_{i\pm\alpha}$ are reconstructed using

$$\begin{aligned}\hat{Q}_{i-\alpha} = \hat{Q}_i(x_{i-\alpha}) = \omega_0 &\left[Q_i + (3Q_i - 4Q_{i+1} + Q_{i+2})\frac{\sqrt{3}}{12}\right] + \\ \omega_1 \left[Q_i - (-Q_{i-1} + Q_{i+1})\frac{\sqrt{3}}{12}\right] &+ \omega_2 \left[Q_i - (Q_{i-2} - 4Q_{i-1} + 3Q_i)\frac{\sqrt{3}}{12}\right]\end{aligned} \tag{3.19}$$

and

$$\begin{aligned}\hat{Q}_{i+\alpha} = \hat{Q}_i(x_{i+\alpha}) = \omega_0 &\left[Q_i - (3Q_i - 4Q_{i+1} + Q_{i+2})\frac{\sqrt{3}}{12}\right] + \\ \omega_1 \left[Q_i - (Q_{i-1} - Q_{i+1})\frac{\sqrt{3}}{12}\right] &+ \omega_2 \left[Q_i - (-Q_{i-2} + 4Q_{i-1} - 3Q_i)\frac{\sqrt{3}}{12}\right]\end{aligned} \tag{3.20}$$

Provided we have the non-linear weights, ω_k, these four reconstruction schemes is all we need to compute the eight integration points used in the flux computations.

If we, however, want to use the fourth order well-balanced source term (see section 5.1), we would also need the to reconstruct values at the cell centre, $x = x_i$. For these points values or line averages, the coefficients of the reconstruction polynomials, $c_i^{k,j}$ are not stated explicitly in the literature studied for this thesis. Therefore we truly need the formula (3.16). Inserting the central point x_i, we find

$$\begin{aligned}\hat{Q}_i = &\frac{1}{24}\omega_0(23Q_i + 2Q_{i+1} - Q_{i+2}) \\ &+ \frac{1}{24}\omega_1(-Q_{i-1} + 26Q_i - Q_{i+1}) + \frac{1}{24}\omega_2(-Q_{i-2} + 2Q_{i-1} + 23Q_i),\end{aligned} \tag{3.21}$$

which gives us all needed polynomials for fourth-order Gaussian and source term quadratures. E.g. to obtain the points needed for the F flux, using the approach from Figure 3.1a, we do one sweep in y direction, applying (3.19) and (3.20) to obtain the two line averages (marked in red). Next we do a sweep in the x

direction, applying (3.17), (3.18) and (3.21) to each of the (red) line averages. By instead doing the first sweep in the x direction and the second in the y direction, we obtain the integration points for the G flux. The reconstruction procedure shown in Figure 3.1b, follows analogously.

To determine the non-linear weights, ω_k, we need the optimal linear weights, d_k, based on the grid, and so-called smoothness indicators, β_k, based on cell averages.

The optimal weights are based purely on the grid. They are chosen such that when $\omega^k = d_k$ are inserted into (3.15) we ensure that the reconstruction is of order $2r-1$ for smooth data. Additionally the optimal weights must satisfy the condition

$$\sum_k d_k = 1.$$

The WENO reconstruction relies on good choices of the smoothness indicators. They are designed to produce polynomials of order $2r-1$ in smooth sections and avoid oscillations in non-smooth sections. They are given by

$$\beta_k = \sum_{m=1}^{r-1} \int_{\xi_{i-1/2}}^{\xi_{i+1/2}} \left(\frac{\mathrm{d}^m}{\mathrm{d}\xi^m} P_k(\xi) \right)^2 \Delta\xi^{2m-1}\, \mathrm{d}\xi, \quad k = 0, ..., r-1 \tag{3.22}$$

where ξ denotes the variable in a generic sweep direction, i.e. x or y in the 2D case. From this we have $\beta_k = \mathcal{O}(\Delta\xi^2)$ if the corresponding stencil contains smooth data, and $\beta_k = \mathcal{O}(1)$ in case of a discontinuity.

Having d_k and β_k we calculate the non-linear weights as follows:

$$\omega_k = \frac{\alpha_k}{\sum_l \alpha_l}, \quad k, l = 0, ..., r-1, \tag{3.23}$$

with

$$\alpha_k = \frac{d_k}{(\epsilon + \beta_k)^2}, \tag{3.24}$$

for some small value of ϵ, simply to avoid division by zero. The choice of ϵ might change depending on the problem. For computations using double precision, $\epsilon = 10^{-6}$ or $\epsilon = 10^{-8}$ are typical choices. For single precision, as we will use on the GPU, $\epsilon = 10^{-3}$ has been used with success. Whenever the choice of ϵ has been of any significance for the problems discussed herein, it will be commented.

This yields

$$\omega_k = \mathcal{O}(1)$$

for smooth data and

$$\omega_k = \mathcal{O}(\Delta\xi^4)$$

in case of discontinuities. That is, the weights will be close to the optimal weights for smooth data and disregard the stencil corresponding to discontinuous data.

For a fifth-order WENO scheme, $r = 3$, the numerical values of β are

$$\begin{aligned}
\beta_0 &= \frac{13}{12}(Q_i - 2Q_{i+1} + Q_{i+2})^2 + \frac{1}{4}(3Q_i - 4Q_{i+1} + Q_{i+2})^2 \\
\beta_1 &= \frac{13}{12}(Q_{i-1} - 2Q_i + Q_{i+1})^2 + \frac{1}{4}(Q_{i-1} - Q_{i+1})^2 \\
\beta_2 &= \frac{13}{12}(Q_{i-2} - 2Q_{i-1} + Q_i)^2 + \frac{1}{4}(Q_{i-2} - 4Q_{i-1} + 3Q_i)^2
\end{aligned} \tag{3.25}$$

.

The optimal linear weights, d, are stated in Table 3.1. Worth commenting are the weights for x_i, which strictly speaking are fourth order weights. They are to be used instead of the fifth-order weights because some of the latter are negative, which may lead to oscillations near discontinuities. Since the center integration points are only used in the source term quadrature, which anyway is of order four, this does not effect the accuracy of the scheme.

x_ξ	d_0	d_1	d_2
$x^L_{i+\frac{1}{2}}$	$\frac{3}{10}$	$\frac{6}{10}$	$\frac{1}{10}$
x_i	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{1}{4}$
$x^R_{i-\frac{1}{2}}$	$\frac{1}{10}$	$\frac{6}{10}$	$\frac{3}{10}$
$x_{i-\alpha}$	$\frac{210-\sqrt{3}}{1080}$	$\frac{11}{18}$	$\frac{210+\sqrt{3}}{1080}$
$x_{i+\alpha}$	$\frac{210+\sqrt{3}}{1080}$	$\frac{11}{18}$	$\frac{210-\sqrt{3}}{1080}$

Table 3.1: The optimal linear weights used in the WENO reconstruction.

With the optimal linear weights and smoothness indicators (3.25), we are able to form the non-linear weights (3.23), and thus also the complete WENO procedure (3.15). As we can see, the computational cost is massive compared to a simple second-order reconstruction, and in the GPU implementation it will be crucial to store as few variables as possible during the computation.

4. Graphics processor units in heterogeneous computing

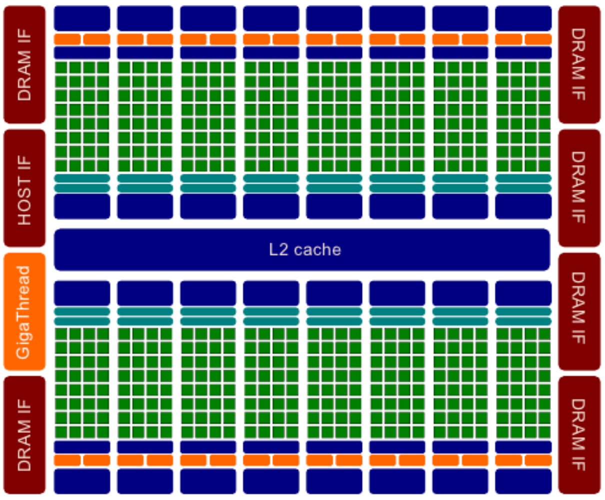
Heterogeneous computing has gained a lot of attention in the recent years. While the traditional computing architecture, the processor (CPU), has reached a point where the current cooling techniques have become a limiting factor of the attainable clock frequency [3], the need for performance improvement by means of parallelism has increased, both through multiple core CPUs and, more importantly, through heterogeneous computing architectures, designed for high data throughput rather than high performance on serial instructions. An overview of heterogeneous architectures such as CBEA, GPU and FPGA can be found in [3]. Our focus will be limited to the Graphics processor unit (GPU), see e.g. [3,5]. The GPU is controlled by the CPU, and was initially intended for handling computer graphics, by computing the color of each pixel on the screen. This is a highly parallel process, meaning that the pixels can be computed independently, which now has been extended to more general computations. Both AMD, Intel and NVIDIA offer programmable GPUs, but we will restrict ourselves to the GPUs by NVIDIA.

4.1. GPU architecture

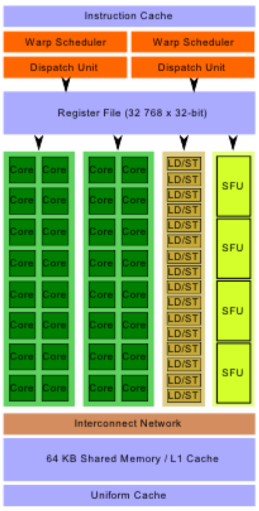
The GPU architecture is in constant development. While the Kepler[1] technology is the most recent contribution from NVIDIA, we will focus on their Fermi[2] technology from 2010. Both the GPU used for benchmarking in this thesis,

[1]`http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`

[2]`http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`



(a) GPU



(b) SM

Figure 4.1: A modern Fermi-type GPU and a detailed illustration of one multiprocessor, SM. Figures taken from [5].

and the one used by Brodtkorb [7], are Fermi GPUs. In Figure 4.1 there is an illustration of the architecture of a Fermi GPU, which consists of 16 streaming multiprocessors (SMs), which share an L2 cache and DRAM global memory, as well as being provided with a local memory hierarchy. The SMs are responsible for distributing the computations to the scalar processors (SPs), or CUDA cores, denoted as 'cores' in Figure 4.1b. This type of parallelism is called SIMD (Single Instruction Multiple Data). Each of the 16 SMs contain 32 CUDA cores. The cores are equipped with L1 cache and registers and also a particularly fast, but limited, memory, called *shared memory.*

The term 'core' in this case might be somewhat misleading, seeing as the performance of the SPs are nowhere near the levels we find on CPUs. An SP is best suited for simple instructions, such as in stencil computing, typically using single precision. Double precision is also supported, but this will halve the maximum throughput, so we opt to use single precision whenever possible. On a Fermi-type GPU, there are in total $16 \cdot 32 = 512$ CUDA cores, which clearly underscores the massively parallel potential in GPU computing. However, one or more of the SMs are typically disabled. On the GPU used for benchmarks in chapter 7, a NVIDIA Quadro 5000, see Figure 4.2, five of the multiprocessors are disabled, resulting in $11 \cdot 32 = 352$ CUDA cores.

Figure 4.2: The Quadro 5000 GPU from NVIDIA.

4.2. GPU computing using CUDA

Recently, GPU programming has become significantly more user-friendly, with the introduction of GPU programming environments based on familiar high-level programming languages, with the most known being the C99-based OpenCL[3] and the C-based NVIDIA product CUDA [20]. Numerous other platforms exist, such as DirectCompute, OpenACC, C++ AMP, PGI Accelerator. The code developed for this thesis, however, uses CUDA, and consequently this section will revolve around the aspects of GPU programming in a CUDA setting.

A CUDA program is initialized at the CPU. Necessary data is then copied from CPU memory to GPU memory, via the PCI-express bus, which is a potential bottleneck. Next, the GPU executes the designated functions in parallel, using the streaming multiprocessors, before transferring the results back to the CPU, so that the CPU and GPU make up a heterogeneous system. We refer to the CPU as the 'host' and the GPU as the 'device'

When the computations on the GPU are initiated, the multiprocessors schedules hardware threads to execute different parts of the computations in parallel. Threads are scheduled in groups of 32, referred to as a *warp*. On the Fermi-type GPUs, two warps are completed during two clock cycles and one SM can keep 48 active warps at a time. The threads are organized into blocks,

[3]http://www.khronos.org/opencl/

which are equipped with the mentioned shared memory. Data stored in shared memory is accessible to all threads in the block. The shared memory consists of 32 memory banks, each serving one warp every other clock cycle.

The grid, on which we want to perform computations, is then partitioned to match the hardware threads and blocks, as illustrated in Figure 4.3. The threads have designated temporary registers and communicate through shared memory, while the blocks are linked to the global memory, see Figure 4.4. The size of a block is limited by 512 threads, but in practice the computations themselves might set the limit by requiring too much memory or register space. For example, the shared memory capacity for one block can be configured to be either 16 KB or 48 KB.

The functions that executes GPU code in a CUDA program are called *kernels*. They take the number of blocks and threads per block as arguments, and then performs the grid computations. The CUDA codes are written in a language based on C, with some C++ features. We will not cover the details regarding the syntax here, but the intrinsic CUDA function, *syncthreads()*, is essential to point out, as it, when called, makes sure to synchronize all the threads within a block. The importance of syncing threads arises in a CUDA kernel exploiting the shared memory. The typical structure of such a kernel is to first perform computations based on the input data and write the results to shared memory, then synchronize the threads, before each thread reads the data stored in shared memory and use it in its further computations.

More on CUDA programming is found in [19, 20]. For more advanced kernel optimization, see e.g. [21, 22].

4.2.1. Stencil computations on the GPU

Stencil computation is an essential part in explicit methods for PDEs, and are particularly well-suited for parallel computing on cores with restricted computing capacity, like the SPs. It also encourages extensive use of shared memory. To illustrate this, consider for example the heat equation in two spatial dimensions,

$$u_t - au_{xx} - au_{yy} = 0,$$

which can be discretized as

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} - a\frac{u_{i+1,j}^{n} - 2u_{ij}^{n} + u_{i-1,j}^{n}}{\Delta x^2} - a\frac{u_{i,j+1}^{n} - 2u_{ij}^{n} + u_{i,j-1}^{n}}{\Delta y^2} = 0,$$

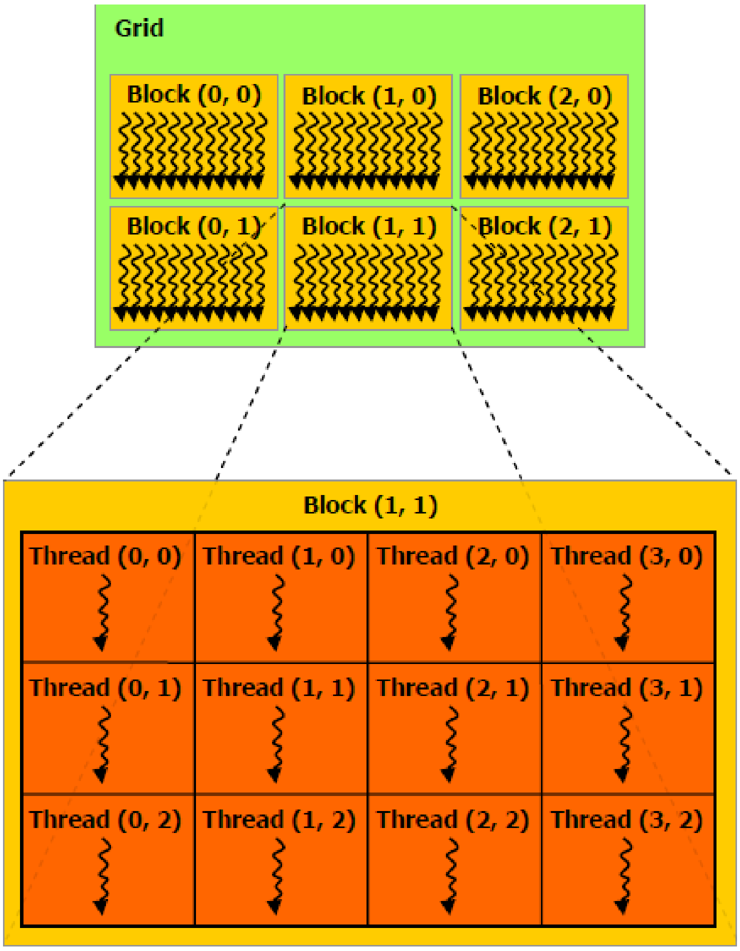


Figure 4.3: Illustration of how the computational grid is mapped into threads and blocks on the GPU. Picture taken from [20].

leading to the explicit finite-differences scheme,

$$u_{ij}^{n+1} = L_{ij}(u^n) = u_{ij}^n + a\frac{\Delta t}{\Delta x^2}(u_{i+1,j}^n - 2u_{ij}^n + u_{i-1,j}^n) + a\frac{\Delta t}{\Delta y^2}(u_{i,j+1}^n - 2u_{ij}^n + u_{i,j-1}^n),$$

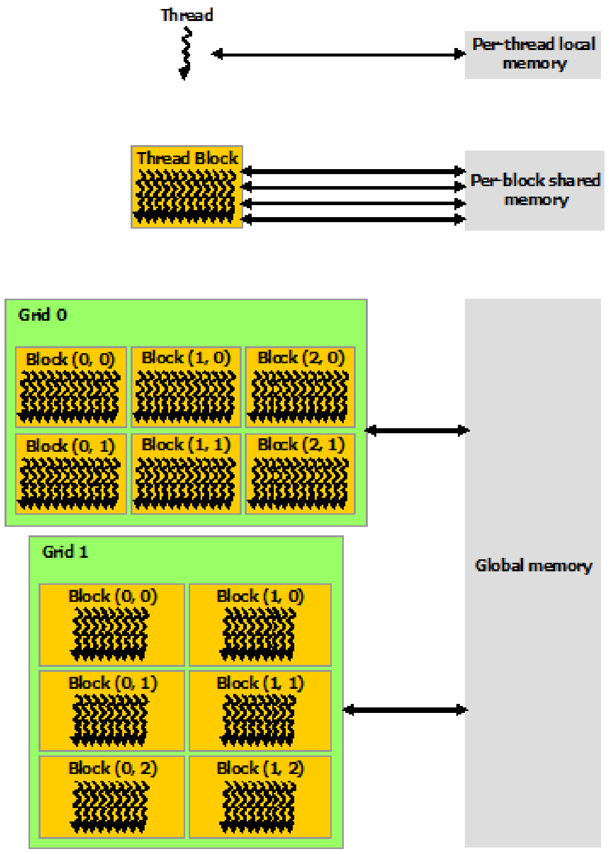


Figure 4.4: The GPU memory hierarchy. Picture taken from [20].

where the stencil needed for computing u_{ij}^{n+1} by the operator L_{ij}, simply is

$$S_{ij} = \{(x_{i-1}, y_j), (x_i, y_{j-1}), (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1})\}.$$

Given a grid with the solution, $\{u_{ij}^n\}$, at $t = t_n$, the new grid values at time $t = t_{n+1}$ are computed using a simple formula, $L_{ij}(u^n)$, and only a small part of the total domain, S_{ij}, which fits perfectly with the GPU hardware. High-resolution methods using explicit ODE solvers are, in essence, just this kind of stencil computation, although admittedly slightly more complex. For example, the second-order reconstruction, using linear polynomials, uses the very same stencil, S_{ij}.

For evolving the heat equation one time step on the GPU we need to partition the domain into blocks, as already seen in Figure 4.3. Say, for simplicity, that the physical domain is square and given by the data points

$$\Omega = (x, y) \in [x_0, x_N] \times [y_0, y_N].$$

These N^2 values are loaded into global GPU memory and the CUDA kernel responsible for the time step launches the desired number of blocks and threads. Each block then loads a portion of the total domain into shared memory. For simplicity, let the blocks also be square and of size $r \times r$, where $r \leq N$ is even. Each block has a unique block index, (b, w). The threads are locally indexed within each block, so we need to map the local thread index, (p, q), and its block index, (b, w), to the global index (i, j) when communicating with the global memory. This mapping is a matter of choice, but we have chosen to position the index (0,0) in the bottom left corner of the grid, such that the GPU indices match the mathematical indices in the grid.

With this set up, block B_{bw} of size $r \times r$, is given by

$$B_{bw} = [x_{br}, x_{br+r-1}] \times [y_{wr}, y_{wr+r-1}].$$

Each thread is commonly given responsibility for one element in the grid. The structure of the kernel would be as mentioned above, where first all the values of u^n in the domain of the current block, is read into shared memory. The reading is done in parallel, distributed over the threads. We sync threads, and then each thread is able to compute the corresponding function value $u_{ij}^{n+1} = L_{ij}(u^n)$.

This procedure works fine until we hit a boundary, either the boundary of the current block, B_{bw}, or the boundary of the physical domain Ω. The numerical scheme, as defined here, is oblivious to boundaries, and to keep that property, we introduce *ghost cells*, also called halo or apron. The ghost cells are added as an extra layer outside the boundary of each block, so that the stencil is valid also for the points at the boundary. For a block situated in the interior of the global domain, Ω, these ghost cells can simply be read from the already existing grid. For example, if

$$x_{br} > x_0, \quad x_{br+r-1} < x_N, \quad y_{wr} > y_0, \quad y_{wr+r-1} < b_N,$$

we simply read the cells along the lines

$$x_{br-1}, \quad x_{br+r}, \quad y_{wr-1}, \quad y_{wr+r}$$

from global memory, in order to make them visible also to the current block. Consequently, an element belonging to a block halo is in total read at least twice from global memory. First once as an interior point for the block it originally belongs to, and then once for each block halo it is included in. This also means that there will be more elements in shared memory than there will be threads per block.

To cope with the case when $x_{br+r-1} = x_N$ or $x_{br} = x_0$, we also need to create global ghost cells, which evidently also serve as boundary conditions for the PDE. Note also that the introduction of ghost cells, both globally and in each block, results in an offset between the thread indexing on the GPU and the physical domain. Point (x_0, y_0) in the grid will in fact, in this example, be given the index $(1, 1)$ on the GPU, since the allocated array needs the 0 index for ghost cells. Furthermore, it is customary to extend the physical domain to fit an integer number of blocks.

For further discussion of ghost cells and boundary conditions, exemplified using image convolution, see [23].

5. Numerical aspects of the shallow water equations

The previous chapters have for the most part revolved around general conservation laws. We will now devote our attention to the shallow water equations in two dimensions and present some of the important numerical issues that arise when solving them. First we present both a second-order and fourth-order well-balanced source term quadrature for the shallow water equations, then we introduce what is called a hydrostatic reconstruction, which is a slight variation of the WENO reconstruction, used to avoid non-physical discontinuities in the bottom topography. We also briefly discuss dry states, when the water height is zero.

5.1. Fourth-order well-balanced source terms

A naive treatment of the bed slope source term, $S_B(Q)$, would be to use some standard quadrature rule, say Simpson's rule or Gaussian quadrature, to approximate the source integrals of each cell, given by

$$S^{(2)} = \frac{1}{\Delta x \Delta y} \int_{\Omega_{ij}} g(w-B)B_x \, \mathrm{d}x\mathrm{d}y,$$

$$S^{(3)} = \frac{1}{\Delta x \Delta y} \int_{\Omega_{ij}} g(w-B)B_y \, \mathrm{d}x\mathrm{d}y.$$

This would however, in general, violate the analytical lake at rest property (3.12), which states that

$$F(Q)_x + G(Q)_y = S(Q).$$

To capture the solution well in, or close to, stationary state, we need to ensure that whenever $w = const$ and $u, v = 0$, the source term quadrature cancels out the numerical fluxes exactly. A properly well-balanced quadrature rule for the source term is derived in [12], based on Simpson's rule in the flux integral, but as previously stated, Gaussian quadrature gives higher order of convergence using less integration points, and is therefore a more desirable choice. We will show the details of the calculations for $S^{(2)}$ and then state the result for $S^{(3)}$, which is derived analogously.

We consider the shallow water equations using the variable change $w = h+B$ (2.9), and start by investigating the numerical flux terms, and in particular the flux on the eastern cell interface, indexed by $(i+1/2, j)$. When in stationary state, it follows that

$$w^+_{i+1/2,j} = w^-_{i+1/2,j}$$

which implies that

$$a^+_{i+1/2,j} = -a^-_{i+1/2,j}, \quad F^+_{i+1/2,j} = F^-_{i+1/2,j}$$

so on the eastern cell interface we have that the central-upwind flux (3.6), is given by

$$H^x_{i+\frac{1}{2},j} = \frac{a^+_{i+\frac{1}{2},j}(F^-_{i+\frac{1}{2},j} - F^+_{i+\frac{1}{2},j})}{2a^+_{i+\frac{1}{2},j}} + \frac{(a^+_{i+\frac{1}{2},j})^2}{2a^+_{i+\frac{1}{2},j}} \underbrace{(w^+_{i+\frac{1}{2},j} - w^-_{i+\frac{1}{2},j})}_{=0} \tag{5.1}$$

$$= \frac{F^+_{i+\frac{1}{2},j} + F^-_{i+\frac{1}{2},j}}{2} = F^+_{i+\frac{1}{2},j} = F^-_{i+\frac{1}{2},j}, \tag{5.2}$$

and analogously on the western interface for F and northern and southern for G. Again, we denote by $F^{\pm}_{i,j}$ the one-sided flux on a particular cell interface, whereas, by $F(Q_{i,j})$ we mean the flux function, as stated in the actual PDE, evaluated at the point (x_i, y_j).

Now, keep in mind that the fluxes $F^{\pm}_{i+1/2,j}$ are calculated using a quadrature rule along the cell interface, in this case a Gaussian quadrature (3.8). We have

$$\begin{aligned} F^-_{i+1/2,j} &= \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} F(\hat{Q}^-(x_{i+\frac{1}{2}}, y))\,\mathrm{d}y \\ &= \frac{1}{2}\Big(F(\hat{Q}^-(x_{i+\frac{1}{2}}, y_{j-\alpha})) + F(\hat{Q}^-(x_{i+\frac{1}{2}}, y_{j+\alpha})\Big), \end{aligned} \tag{5.3}$$

where the integration points are given by

$$\alpha = \frac{1}{2\sqrt{3}}.$$

Next, we use the fact that $u = v = 0$, which reduces the flux functions to

$$F(Q) = \begin{bmatrix} hu \\ \frac{(hu)^2}{w-B} + \frac{1}{2}g(w-B)^2 \\ \frac{(hu)(hv)}{w-B} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2}g(w-B)^2 \\ 0 \end{bmatrix}, \tag{5.4}$$

$$G(Q) = \begin{bmatrix} hv \\ \frac{(hu)(hv)}{w-B} \\ \frac{(hv)^2}{w-B} + \frac{1}{2}g(w-B)^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}g(w-B)^2 \end{bmatrix},$$

which is simply the hydrostatic pressure. Notice that the F and G fluxes contribute to only the second and third equation, respectively.

Based on this, inserting (5.4) and (5.3) into (5.2), we now have that

$$H^x_{i+\frac{1}{2},j} = \frac{1}{2}\left(\frac{g(w_{i+\frac{1}{2},j+\alpha} - B_{i+\frac{1}{2},j+\alpha})^2}{2} + \frac{g(w_{i+\frac{1}{2},j-\alpha} - B_{i+\frac{1}{2},j-\alpha})^2}{2}\right),$$

and analogously for the northern, western and southern interfaces. The total F flux contribution to the cell I_{ij} in stationary state, amounts to

$$\begin{aligned} &\frac{H^x_{i+\frac{1}{2},j} - H^x_{i-\frac{1}{2},j}}{\Delta x} \\ &= \frac{1}{2\Delta x}\left(\frac{g(w_{i+\frac{1}{2},j+\alpha} - B_{i+\frac{1}{2},j+\alpha})^2}{2} + \frac{g(w_{i+\frac{1}{2},j-\alpha} - B_{i+\frac{1}{2},j-\alpha})^2}{2}\right) \\ &- \frac{1}{2\Delta x}\left(\frac{g(w_{i-\frac{1}{2},j+\alpha} - B_{i-\frac{1}{2},j+\alpha})^2}{2} + \frac{g(w_{i-\frac{1}{2},j-\alpha} - B_{i-\frac{1}{2},j-\alpha})^2}{2}\right). \end{aligned}$$

Re-arranging the terms we obtain

$$
\begin{aligned}
&\frac{H^x_{i+\frac{1}{2},j} - H^x_{i-\frac{1}{2},j}}{\Delta x} \\
&= \frac{1}{2\Delta x}\left(\frac{g(w_{i+\frac{1}{2},j+\alpha} - B_{i+\frac{1}{2},j+\alpha})^2}{2} - \frac{g(w_{i-\frac{1}{2},j+\alpha} - B_{i-\frac{1}{2},j+\alpha})^2}{2}\right) \\
&+ \frac{1}{2\Delta x}\left(\frac{g(w_{i+\frac{1}{2},j-\alpha} - B_{i+\frac{1}{2},j-\alpha})^2}{2} - \frac{g(w_{i-\frac{1}{2},j-\alpha} - B_{i-\frac{1}{2},j-\alpha})^2}{2}\right),
\end{aligned}
$$

which, as in [12], can be re-written into

$$
\begin{aligned}
&\frac{H^x_{i+\frac{1}{2},j} - H^x_{i-\frac{1}{2},j}}{\Delta x} \\
&= \frac{g}{2}\left(\frac{B_{i+\frac{1}{2},j+\alpha} - B_{i-\frac{1}{2},j+\alpha}}{\Delta x} \cdot \frac{(w_{i+\frac{1}{2},j+\alpha} - B_{i+\frac{1}{2},j+\alpha}) + (w_{i-\frac{1}{2},j+\alpha} - B_{i-\frac{1}{2},j+\alpha})}{2}\right) \\
&+ \frac{g}{2}\left(\frac{B_{i+\frac{1}{2},j-\alpha} - B_{i-\frac{1}{2},j-\alpha}}{\Delta x} \cdot \frac{(w_{i+\frac{1}{2},j-\alpha} - B_{i+\frac{1}{2},j-\alpha}) + (w_{i-\frac{1}{2},j-\alpha} - B_{i-\frac{1}{2},j-\alpha})}{2}\right).
\end{aligned}
$$

So we see that we end up with two terms, one for each of the Gaussian integration point on the cell interface. To simplify the indexing a bit, we can for cell $I_{i,j}$ denote the four cell interfaces $W = i-1/2$, $E = i+1/2$, $S = j-1/2$, $N = j+1/2$, i.e. west, east, south and north, and the integration points by $\pm\alpha$. Furthermore, we write $h = w - B$.

Then the source term quadrature is found by (3.12). The F and G fluxes dictate

$$
\begin{aligned}
S^{(2)}_{i,j} &= -\frac{g}{2}\left(\frac{B^E_\alpha - B^W_\alpha}{\Delta x} \cdot \frac{h^E_\alpha + h^W_\alpha}{2}\right) - \frac{g}{2}\left(\frac{B^E_{-\alpha} - B^E_{-\alpha}}{\Delta x} \cdot \frac{h^E_{-\alpha} + h^W_{-\alpha}}{2}\right), \\
S^{(3)}_{i,j} &= -\frac{g}{2}\left(\frac{B^N_\alpha - B^S_\alpha}{\Delta y} \cdot \frac{h^N_\alpha + h^S_\alpha}{2}\right) - \frac{g}{2}\left(\frac{B^N_{-\alpha} - B^S_{-\alpha}}{\Delta y} \cdot \frac{h^N_{-\alpha} + h^S_{-\alpha}}{2}\right).
\end{aligned}
\tag{5.5}
$$

Such a quadrature is, however, only of second-order. We use four points, but they are all at the cell interfaces, and therefore do not coincide with the Gaussian quadrature points in two dimensions, $(x_{i+\alpha}, y_{j\pm\alpha})$, $(x_{i-\alpha}, y_{j\pm\alpha})$. In order to increase the order, we need to add interior cell points, $(x_i, y_{j\pm\alpha})$ and

$(x_{i\pm\alpha}, y_j)$, in an extrapolation procedure, as shown in [18]. We denote the reconstructed function values at the interior integration points $B^C_{\pm\alpha}$ and $h^C_{\pm\alpha}$, where the superscript refers to the coordinate in the current flux direction, and the subscript refers to the coordinate in the direction normal to the flux. A fourth-order well-balanced quadrature for the second component of the bed slope source term, reads

$$\begin{aligned}
S^{(2)}_{i,j} &= \frac{g}{2}\left(\frac{4S^2_\alpha - S^1_\alpha}{3} + \frac{4S^2_{-\alpha} - S^1_{-\alpha}}{3}\right),\\
S^2_{\pm\alpha} &= -\left[\frac{(B^E_{\pm\alpha} - B^C_{\pm\alpha})(h^E_{\pm\alpha} + h^C_{\pm\alpha}) + (B^C_{\pm\alpha} - B^W_{\pm\alpha})(h^C_{\pm\alpha} + h^W_{\pm\alpha})}{2\Delta x}\right], \qquad (5.6)\\
S^1_{\pm\alpha} &= -\left[\frac{(B^E_{\pm\alpha} - B^W_{\pm\alpha})(h^E_{\pm\alpha} + h^W_{\pm\alpha})}{2\Delta x}\right].
\end{aligned}$$

The third component, $S^{(3)}_{ij}$, is given by

$$\begin{aligned}
S^{(3)}_{i,j} &= \frac{g}{2}\left(\frac{4S^2_\alpha - S^1_\alpha}{3} + \frac{4S^2_{-\alpha} - S^1_{-\alpha}}{3}\right),\\
S^2_{\pm\alpha} &= \left[\frac{(B^S_{\pm\alpha} - B^C_{\pm\alpha})(h^S_{\pm\alpha} + h^C_{\pm\alpha}) + (B^C_{\pm\alpha} - B^N_{\pm\alpha})(h^C_{\pm\alpha} + h^N_{\pm\alpha})}{2\Delta y}\right], \qquad (5.7)\\
S^1_{\pm\alpha} &= \left[\frac{(B^S_{\pm\alpha} - B^N_{\pm\alpha})(h^S_{\pm\alpha} + h^N_{\pm\alpha})}{2\Delta y}\right].
\end{aligned}$$

Thus, the new quadrature is a linear combination of the second-order well-balanced quadrature (5.5) and the three-point quadrature S^2. They are added in such a way that in stationary-state, $w = \mathit{const}$, the centre integration points cancel out, and we have $S^2 = S^1$ and the fourth-order quadrature is equivalent to the second-order well-balanced quadrature. This way we obtain high-order accuracy, without violating the well-balanced property in stationary-state.

Notice how we need six points to obtain order four for a well-balanced scheme, compared to the naive approach of using four Gaussian integration points to obtain order four.

5.2. Hydrostatic reconstruction

Out of the variables h, w and B in (2.9), we need only reconstruct two of them, as the third follows from the two others. Which two to reconstruct, may depend on the problem and method. A natural choice for high-order schemes seems to be reconstructing h and w from cell averages and compute $\hat{B} = \hat{w} - \hat{h}$, as done in [1,18]. At the cell interfaces, $\hat{B}$ will then, in general, be discontinuous, therefore we will use what is called a hydrostatic reconstruction of h, which involves using the maximum one-sided value of B at the interface integration points, i.e.

$$\hat{B}_{ij}(x,y) = \max\left\{\hat{B}^+ \, , \, \hat{B}^-\right\}, \quad (x,y) \in \partial I_{ij}.$$

Consequently, exemplified on the eastern and northern cell interface, we have these adjusted one-sided values of h, denoted h^*:

$$\begin{aligned}
h^*_{ij}(x^-_{i+\frac{1}{2}}, y_{j\pm\alpha}) &= \max\left\{0 \, , \, w^-_{i+\frac{1}{2},\pm\alpha} - B_{i+\frac{1}{2},j\pm\alpha}\right\} \\
h^*_{i+1,j}(x^+_{i+\frac{1}{2}}, y_{j\pm\alpha}) &= \max\left\{0 \, , \, w^+_{i+\frac{1}{2},\pm\alpha} - B_{i+\frac{1}{2},j\pm\alpha}\right\} \\
h^*_{ij}(x_{i\pm\alpha}, y^-_{j+\frac{1}{2}}), &= \max\left\{0 \, , \, w^-_{i\pm\alpha,j+\frac{1}{2}} - B_{i\pm\alpha,j+\frac{1}{2}}\right\} \\
h^*_{i,j+1}(x_{i\pm\alpha}, y^+_{j+\frac{1}{2}}), &= \max\left\{0 \, , \, w^+_{i\pm\alpha,j+\frac{1}{2}} - B_{i\pm\alpha,j+\frac{1}{2}}\right\}.
\end{aligned} \tag{5.8}$$

In the flux computations we will use the adjusted conserved variables

$$Q^* = \begin{bmatrix} h^* \\ (hu) \\ (hv) \end{bmatrix},$$

which means that we are actually altering the fluxes. This is resolved by adding a hydrostatic correction. One hydrostatic correction term is added for each of the integration points in each flux, which in the case of Gaussian quadrature gives us two terms. The modified fluxes read, for the F fluxes, using a Gaussian quadrature for the one-sided fluxes,

$$
\begin{aligned}
H^x_{i+\frac{1}{2},j} = F^h_{i+\frac{1}{2},j} \quad &+ \frac{g}{4}\begin{bmatrix} 0 \\ h^2_{ij}(x_{i+\frac{1}{2}}, y_{j+\alpha}) - (h^*)^2_{ij}(x_{i+\frac{1}{2}}, y_{j+\alpha}) \\ 0 \end{bmatrix} \\
&+ \frac{g}{4}\begin{bmatrix} 0 \\ h^2_{ij}(x_{i+\frac{1}{2}}, y_{j-\alpha}) - (h^*)^2_{ij}(x_{i+\frac{1}{2}}, y_{j-\alpha}) \\ 0 \end{bmatrix}, \\
H^x_{i-\frac{1}{2},j} = F^h_{i-\frac{1}{2},j} \quad &+ \frac{g}{4}\begin{bmatrix} 0 \\ h^2_{ij}(x_{i-\frac{1}{2}}, y_{j+\alpha}) - (h^*)^2_{ij}(x_{i-\frac{1}{2}}, y_{j+\alpha}) \\ 0 \end{bmatrix} \\
&+ \frac{g}{4}\begin{bmatrix} 0 \\ h^2_{ij}(x_{i-\frac{1}{2}}, y_{j-\alpha}) - (h^*)^2_{ij}(x_{i-\frac{1}{2}}, y_{j-\alpha}) \\ 0 \end{bmatrix},
\end{aligned}
\tag{5.9}
$$

where F^h is some regular homogeneous two-sided numerical flux. In [18], the Lax-Friedrich flux (3.4) is used. We will use the central-upwind flux (3.6).

Compared to [18], the expressions in (5.9) seem slightly different, but whereas they first compute the two-sided flux at each integration point, using the Lax-Friedrich flux, and then apply the Gaussian quadrature, we do it the other way around, thus resulting in an extra factor $1/2$ in the added hydrostatic corrections, yielding $g/4$. We must also stress that these corrections are unique to each cell, thus seemingly violate the physically necessary condition that the eastern flux of cell $(i-1, j)$ equals the western flux of (i, j). Therefore it might be more instructive to think of the hydrostatic correction as some sort of source term, which is also indicated by the notation in [1]. When it comes to the actual implementation, adding the the corrections to the source term also seems as the easiest and 'cleanest' approach.

5.3. Dry states

Dry states in a system of shallow water flow refers to the areas where the water height is zero, or close to zero. Computationally the treatment of these areas is difficult. Since the computation of the CFL condition, which is necessary for stability, typically involves calculation of the maximum eigenvalues at integration points, which in turn are given by $u \pm \sqrt{gh}$ and $v \pm \sqrt{gh}$, the numerical system breaks down if h is negative at an integration point. For any reconstruction of more than order 1, this is bound to happen when the cell average $\bar{h}$ is small enough compared to both zero and the neighbouring averages. Negative water heights are of course also physically meaningless. Consequently, in order for a high-resolution scheme to support dry states, it has to ensure non-negativity both for h at every integration point at time $t = t_n$, and also for $\bar{Q}_1^{n+1}$, the resulting averages at time $t = t_{n+1}$.

One way of dealing with dry states, as proposed in [4,12], is to reconstruct the physical variables $Q = (h, u, v)$ instead of $Q = (w, hu, hv)$ when $h < K$, where K is a given small constant. As pointed out in [29], the conservation law governing $Q = (h, u, v)$ is only mathematically conservative, and will only serve as an accurate model in the presence of weak shock waves. Switching to this system also violates the well-balanced property, as small, purely numerical waves are introduced, even if the physical system is in stationary state.

Another approach, applicable to second-order reconstructions, is to alter the slopes used for the reconstruction when a negative water height is encountered, as done in [14]. This is a better solution with respect to GPU memory, since there is no need for storing both the physical and conserved variables, but it cannot be generalized to higher order reconstructions, since by altering the slopes it is only possible to ensure non-negativity at one integration point at one interface at a time, whereas in the high-order methods more integration points are needed at each interface. It also violates the well-balanced property, but Kurganov suggested an improved correction procedure [2] which preserves well-balancedness even near dry areas.

Dry states support has not been the focus in this thesis, but a plausible workaround is to swap to the above mentioned second-order positive-preserving scheme near dry zones. That is, if we can accept the reduced order of convergence .

6. Implementation on the GPU

The starting point for the implementations done for this thesis is a stripped down version of the shallow water simulator by Bordtkorb et al. [7], which is a GPU implementation of the second-order scheme of Kurganov and Petrova [14]. That scheme is well-balanced, supports dry states and discontinuous bottom topography and is also equipped with an early-exit optimization, to avoid unnecessary computations on dry cells or cells in stationary state. Furthermore, it supports a bed friction term with spatially varying Manning coefficient and reflective, fixed, outflow and inflow boundary conditions. Not all of these properties are easily transferable to schemes of higher order. The focus here will therefore mainly be to ensure that the scheme is well-balanced. Dry states are treated naively, by simply setting $h = 0$ whenever $h < 0$. A more proper treatment is difficult in high-order schemes.

Two schemes have been implemented, one using the fourth-order source term quadrature (5.6)-(5.7), the hydrostatic reconstruction of h from section 5.2, and one using the second-order well-balanced source term quadrature (5.5) and a bilinear interpolation of B (3.3). Using the notation from section 3.5, we denote these schemes ($\cdot$,5,4,4) and ($\cdot$,5,4,2), respectively, where the dots indicate that several Runge-Kutta methods are supported. The ($\cdot$,5,4,2) scheme is build directly into the framework of Brodtkorb's scheme, the only changes being in the flux-source kernel and the global ghost cells. The ($\cdot$,5,4,4) scheme involves slightly more fundamental changes, as a consequence of going from reconstructing B to reconstructing h. We will therefore focus mostly on the ($\cdot$,5,4,4) scheme, and comment on the ($\cdot$,5,4,2) scheme when appropriate. Both schemes are implemented using single precision, which is the preferred choice on GPUs due to the suboptimal support for double precision.

6.1. Outline of the scheme

The central part of the simulator in [7], and also the ones presented here, is the REA algorithm, shown in Algorithm 1. One time step of the simulator is illustrated in Figure 6.1. From cell averages of Q, and point values of B, they first reconstruct the values at the integration points, using linear reconstruction supplied with the minmod limiter. The reconstructed point values are used to calculate the flux and source integrals, and the maximum and minimum eigenvalues. The latter is used to decide the maximal allowed time step, according to the CFL condition (2.11). For the flux integral, the central-upwind flux (3.6) is used for the two-sided flux and the midpoint quadrature rule (3.7) for the one-sided fluxes. The bed slope source term is calculated by the second-order well-balanced source term using the midpoint rule, which is equivalent to inserting $\alpha = 0$ into (5.5).

The flux and source values are evolved in time, and boundary conditions are added using global ghost cells, as explained in section 4.2.1, to obtain the cell averages at the next time step. Each of these tasks is performed on the GPU by its own dedicated kernel.

The main changes from Brodtkorb's scheme are in the flux and source calculations, referred to as the *flux-source kernel*, and we will dig into that algorithm in more detail. The changes made in the Runge-Kutta kernel are more straight-forward, we simply add one sub step and apply (3.11) instead of (3.10). Also the initial data generation had to be adjusted. The input data is typically described by a function given at points. For a second-order scheme, evaluating the initial functions at the cell midpoints yields a second-order approximation of the cell averages. To ensure that no initial error would effect the simulation, the initial cell averages were instead computed using a standard fourth-order Gaussian quadrature. Boundary conditions have not been the main area of attention, and will only be discussed briefly later on.

Before we state the algorithm of the flux-source kernel, we will use the theory of the WENO reconstruction and well-balanced source quadratures to get an overview over the needed data. By that, we will determine the appropriate halo size. The change in the halo also needs to be implemented in the boundary conditions kernel.

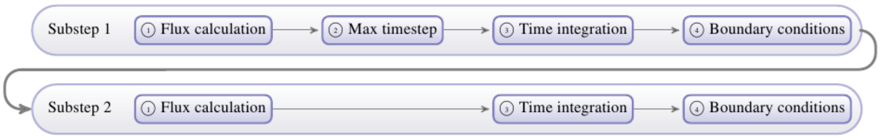
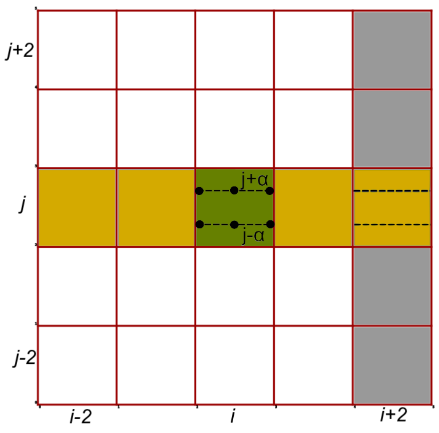


Figure 6.1: Shows the general outline of the scheme using a second-order Runge-Kutta method. Add one more line for third-order Runge-Kutta. The focus here will be the Flux calculation kernel ① and the Time integration kernel ③. Illustration taken from [7].

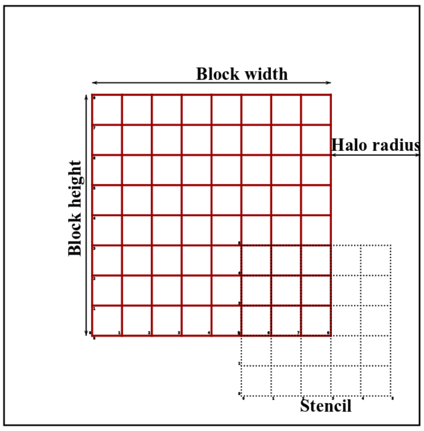
6.2. Reconstruction of the conserved variables

For each cell, $I_{i,j}$, we need to apply a WENO reconstruction procedure, as described in Section 3.6. Since the WENO computations are rather costly, we would like to avoid computing any values more than once for each block. Naturally, in all cells used as ghost cells, the same WENO reconstructions will be carried out also in adjacent blocks, in order to minimize the use of global memory. If we store the line averages in shared memory and reconstruct the point values where they are needed in the flux and source computation, we do not calculate any values more than once per block. Furthermore, the points needed in the H^x flux computation and the second source term, $S^{(2)}$, are all situated along different lines than the points used in H^y and $S^{(3)}$, see Figure 3.1. By realizing this, we need only compute and store half of the needed line averages at a time, which truly underscores the dimension-by-dimension approach. This saves shared memory a great deal.

The bottom topography, B, is known and does not vary in time and could ideally be computed only once and later re-used. To save global memory, however, Brodtkorb et al. chose to re-compute B at each time substep, a procedure we will adopt. In the works of Kurganov and Brodtkorb [7, 12, 14], amongst others, the strategy has been to reconstruct w and B, and then compute h for use in the flux and source integrals. This approach has been used in the (·,5,4,2) scheme, where the water elevation, w, is reconstructed by the fifth-order WENO reconstruction from cell averages, whereas B is reconstructed by the means of regular bilinear interpolation from the point values at the four cell corners (3.3). This procedure could probably be extended to higher order reconstructions, but is not easy to find in the literature, if mentioned at all. In [1] it is concluded that reconstructing w and h, actually is the preferable choice, even for linear reconstruction. The application to linear reconstruction will not be investigated here, but for higher order reconstructions this choice seems more suitable. We will therefore apply the hydrostatic reconstruction from Section 5.2. By this we are evolving both w and h in time, computing B at each integration point at each time step. In the current implementation, to run the hydrostatic reconstruction on homogeneous problems, $B = const$, we simply set $water\_height\_no = 0$ as a runtime parameter, which will cause h to be reconstructed using the values of w. This implementation is of course not optimal, as we do not need to reconstruct h if the problem is homogeneous, but it yields correct results. For further optimization for homogeneous problems, the reconstruction of h and the computation of the source term should be omitted, possibly by calling a different flux kernel, without these computations, based on the parameter $water\_height\_no$.



(a) Stencil



(b) Block and halo

Figure 6.2: Illustration of the WENO stencil and the needed halo.

To compute the fourth-order source term quadrature (5.6)-(5.7), we need interior cell points for both h and w. As previously described, we can choose to either reconstruct the line averages at the interfaces and the two lines through the cell midpoint, in total six line averages, or reconstruct the four lines located along $x = x_{i\pm\alpha}$ and $y = y_{j\pm\alpha}$, see Figure 3.1. The latter is the method of choice. The reasoning behind, is that we wish to store the line averages in shared memory and compute the point values whenever they are needed. Fewer line averages means less load on shared memory. This reconstruction process is illustrated in Figure 6.2, exemplified with the points needed for H^x and $S^{(2)}$, where we show the need for a 25-point, non-compact, stencil in order to determine a point value in a given cell, (i, j). To obtain one of the six points, we perform a sweep in the x direction (yellow), using the *line* averages in the stencil

$$S^x_{i,j} = \big\{(i-2, j), (i-1, j), (i, j), (i+1, j), (i+2, j)\big\}.$$

These line averages are in turn computed by a sweep in the y direction (shown in grey for cell $(i+2, j)$), using the *cell* averages in the stencils

$$S^y_{i+2,j} = \big\{(\cdot, j-2), (\cdot, j-1), (\cdot, j), (\cdot, j+1), (\cdot, j+2)\big\},$$

hence we need a 25-point stencil. Again, we obtain the points used for $S^{(3)}$ and H^y, by performing the sweeps in reversed order.

When the flux kernel is called, which is responsible for the reconstructions, we divide the domain into blocks. We will come back to the block size shortly, but for now, consider the 8×8 example in Figure 6.2b, which clearly illustrates how we need two ghost cells to compute all the point values in the block. However, in order to compute the fluxes on the block boundary, we also need point values and fluxes from adjacent cells, which forces us to increase the halo radius to 3, see Figure 6.3. Note here that the stencil in Figure 6.2a would be the same if we were to reconstruct interface averages instead, but the line average sweep (grey) and point value sweep (yellow) would have been in the opposite directions.

6.3. The flux-source kernel

The most central part of the simulator, is the kernel responsible for computing the net flux and source terms for each cell, as well as the maximum and minimum eigenvalue. Here, both the WENO reconstruction, the Gauss integral, the central-upwind flux computation and the source term integral are performed. Each thread is responsible for its designated cell, and will return the net change in the conserved variables in that cell, i.e. the right-hand side of (2.10). The algorithm is outlined in Algorithm 2. First we compute all the interface fluxes and source terms, before we sum them up to one value, to save global memory, as done in [7]. These values are stored in an array, $R = (R_1, R_2, R_3)$, which is sent to the timestep kernel.

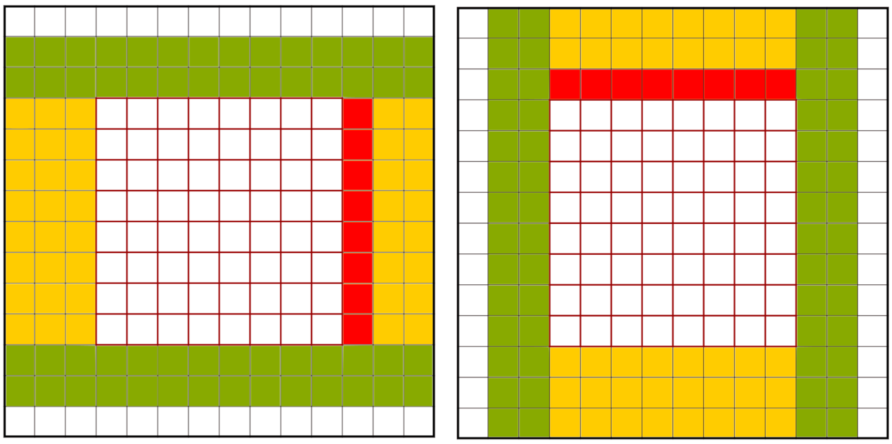

(a) y-x sweep for F flux (b) x-y sweep for G flux

Figure 6.3: Ghost cells needed (green, yellow, red) for flux computation, exemplified on a 8×8 block. In red cells, we calculate fluxes, the yellow ones are needed in the reconstruction of both *point values* and *line averages*, whereas the green ones are needed only in the reconstruction of *line averages*.

Ghost cells

The main issue in the flux-source kernel is the handling of ghost cells, which is illustrated in Figure 6.3. We will compute the western and southern fluxes in each cell, as done in [7], and thus, to get the fluxes on the northern and eastern block boundary, we compute the fluxes also in the innermost row and column of ghost cells (marked as red). This is done by one warp after finishing the flux computations in the actual computational domain, the output domain. Furthermore, in the computation at the southern and western block boundary, we need point values from the adjacent ghost cells, which explains the need for increasing the halo radius by one cell in each direction. In the current implementation, the one-sided 'minus' point values used for the western and southern boundary of the block, are precomputed, and stored in shared memory, before the actual flux computation starts.

In each block, the output domain is positioned at local indices $i = 3, ...bw + 2$, and $j = 3, ..., bh + 2$, if we define bw = 'block width' and bh = 'block height'. Thus we compute the south fluxes for cells $(i = 3, ..., bw + 2, j = bh + 3)$ and the west fluxes for cells $(i = bw + 3, j = 3, ..., bh + 2)$ in order to get the fluxes at the block boundaries.

From Figure 6.3, we see how the halo is different for the two fluxes, H^x and H^y. To be able to compute all the necessary point values in the second WENO sweep, line averages needs to be computed in the yellow and red area, in addition to the output domain. The reconstruction of these line averages, in turn needs cell averages also from the green area. In total, only two rows or two columns are unused in each of the flux computations.

Note that this is only valid for the reconstruction procedure using interior line averages, see Figure 3.1a. When using the line averages on the cell interfaces, Figure 3.1b, the halo looks slightly different.

Maximum time step

In the flux and source computations in each cell (i, j), the maximal allowed time step of that particular cell interface is computed by inserting the eigenvalues of the shallow water equations into (2.11). We have

$$r_x = C_{cfl}\Delta x / \max |u \pm \sqrt{gh}|, \quad u = u^{\pm}_{i-\frac{1}{2},j\pm\alpha}, h = h^{\pm}_{i-\frac{1}{2},j\pm\alpha}, \tag{6.1}$$

for the four integration points involved in the two-sided F flux on the western cell interface, and

$$r_y = C_{cfl}\Delta y / \max |v \pm \sqrt{gh}|, \quad v = v^{\pm}_{i\pm\alpha,j-\frac{1}{2}}, h = h^{\pm}_{i\pm\alpha,j-\frac{1}{2}}, \tag{6.2}$$

for the four integration points involved in the two-sided G flux on the southern cell interface, and the maximum time step for that particular cell, (i, j), is set to

$$\Delta t \leq \min \{r_x \, , \, r_y\}. \tag{6.3}$$

The CFL coefficient, C_{cfl}, is taken as a input parameter to the flux-source kernel, and is decided runtime based on the chosen Runge-Kutta method. This was not needed in the scheme of Brodtkorb et. al, since the $C_{cfl} = 0.25$ both for the first-order and second-order TVD Runge-Kutta method.

Each thread writes its maximal allowed time step to global memory. The max time step kernel then collects these values and finds the maximum time step for the entire grid, which is the one to be used by the time step kernel.

Algorithm 2 Implementation of the flux-source kernel in the $(\cdot,5,4,4)$ scheme

```
Current cell has index $(x, y) = (i, j)$
Shared memory arrays: $S, Q_s, Q_M, Q_P$
Local variables: $R_1, R_2, R_3$, for storing flux and source
Load $h, Q_1, Q_2, Q_3$
Reconstruct lines $(x, y_{j\pm\alpha})$ for cell $(i, j)$, one warp takes care of ghost cells, store in $Q_M(i, j), Q_P(i, j)$.
Reconstruct point values in block column 2.
calcFluxSourceWest($S, Q_M, Q_P, i, j$);
if $i = bw + 2$ then
  calcFluxSourceWest($S, Q_M, Q_P, bw + 3, j$);
end if
$R_1 = F^{(1)}_{i-1/2} - F^{(1)}_{i+1/2}$;  $R_2 = F^{(2)}_{i-1/2} - F^{(2)}_{i+1/2} + S^{(2)}$;  $R_3 = F^{(3)}_{i-1/2} - F^{(3)}_{i+1/2}$;
Reconstruct lines $(x_{i\pm\alpha}, y)$ for cell $(i, j)$, one warp takes care of ghost cells, store in $Q_M(i, j), Q_P(i, j)$.
Reconstruct point values in block row 2.
calcFluxSourceSouth($S, Q_M, Q_P, p, q$);
if $j = bh + 2$ then
  calcFluxSourceSouth($S, Q_M, Q_P, i, bh + 3$);
end if
$R_1 = R_1 + G^{(1)}_{j-1/2} - G^{(1)}_{j+1/2}$;  $R_2 = R_2 + G^{(2)}_{j-1/2} - G^{(2)}_{j+1/2}$;  $R_3 = R_3 + G^{(3)}_{j-1/2} - G^{(3)}_{j+1/2} + S^{(2)}$;
Calculate minimum eigenvalue for the current block
```

The flux and source integrals

In a second-order scheme, it could be instructive and clean to put the source and flux computations in separate functions. In high-order schemes, however, if we want to avoid multiple computations of smoothness indicators and point values, it is better to keep the two computations in the same functions, so that the reconstructed point values can be used for all its purposes right away.

When all needed line averages in the current flux direction are computed, we initiate the flux-source function, calcFluxSourceWest and calcFluxSource-South in Algorithm 2. The calcFluxSourceWest function is outlined in Algorithm 3. Each thread first computes all six integration points in its designated cell and the bed slope source term using (5.6)-(5.7). All the needed points are computed by the same thread, so this implementation is rather straight forward.

Algorithm 3 Implementation of the 'calcFluxSourceWest(S, Q_M, Q_P, i, j)' function in the ($\cdot$,5,4,4) scheme

```
Current cell has index $(x, y) = (i, j)$
Input: Line averages, $\pm\alpha$, empty source term array, $S$.
Reconstruct $Q^+_{i-1/2,j\pm\alpha}, \quad Q_{i,j\pm\alpha}, \quad Q^-_{i+1/2,j\pm\alpha}$
$S(i,j) = S^{(2)}_{i,j}$
Add hydrostatic correction from $h^+_{i-1/2,j\pm\alpha}, h^-_{i+1/2,j\pm\alpha}$ to $S(i,j)$
$Q_M(i-2,j) \leftarrow Q_{i+1/2,j-\alpha}, \quad Q_P(i-2,j) \leftarrow Q_{i+1/2,j+\alpha}$
syncthreads();
Compute $h^*_{i-1/2,j\pm\alpha}$ by using $Q_M(i-2,j), Q_P(i-2,j)$
Compute two-sided flux $F_{i-1/2,j}$ and eigenvalues.
$Q_M(i-3,j) \leftarrow h^*_{i-1/2,j-\alpha}, \quad Q_P(i-3,j) \leftarrow h^*_{i-1/2,j-\alpha}$
syncthreads();
$S(i,j) = S(i,j) +$ hydrostatic correction form $h^*$
$Q_P(i-3,j) = F_{i-1/2,j}$
Return max eigenvalue
```

At this point, also the contribution to the hydrostatic flux correction (5.9) coming from $h_{i\pm1/2,j}$, are ready to be computed. This correction is simply added to the source term. The contribution from h^* is added later on.

In the flux computation, however, we need reconstructions from adjacent cells to get the 'plus' and 'minus' points used in the central-upwind flux (3.6). The 'plus' points are already available, seeing as we have chosen to compute the west and south fluxes. Thus we have the choice of doing the reconstructions of the 'minus' points yet again, or we can use shared memory to read from adjacent threads. The latter approach leads to frequently use of the syncthreads() command, but no significant effects on runtime from this extra syncing, has been detected, which makes it the preferable strategy.

As soon as Q and h have been computed at the integration points, we are done with the arrays storing the line averages. This space can now be exploited by letting each thread write the points needed by adjacent cells, to the free shared memory arrays. These values are then used, first in the hydrostatic reconstruction, h^*, (5.8), then to the hydrostatic correction in the current cell (5.9), before h^* is written back to shared memory, to be used by the adjacent cell in the hydrostatic correction.

To exemplify for the F flux computation: A cell (i, j) sends its eastern reconstructed values of w and h to shared memory, to be read by cell $(i+1,j)$. Then cell (i,j) uses information from cell $(i-1,j)$ in the hydrostatic reconstruction on the western interface. The values of h^* are then sent back to shared memory to be used in the hydrostatic correction of the eastern interface of cell $(i-1,j)$. Again, all the hydrostatic corrections are simply added to the source term.

Finally, since we are computing the flux on the eastern and northern border of the output domain in a second call to the calcFluxSource function, we must make sure that all writing to shared memory during the calcFluxSource function is made with an offset of 2. The line averages in cells $i = [bw{+}1\,,\, bw{+}2]$ and $j = [bh+1\,,\, bh+2]$, are needed in the fluxes in cells $i = bw+3$ and $j = bh+3$, and cannot be overwritten during the first call to 'calcFluxWest' and 'calcFluxSouth'.

6.3.1. Implementation of second-order source term

The implementation of the (·,5,4,2) is slightly less intricate. The fifth-order WENO reconstruction was carried out for Q, and the flux integrals were computed using Gaussian quadrature, but the bilinear interpolation of B from the Kurganov-Petrova scheme was kept, alongside with the second-order bed slope source term (5.5). Although using a fifth-order WENO method, technically the reconstruction of $Q_1 = h + B$ is of only second-order since we use bilinear interpolation on B, but since the reconstruction of B is only relevant when the source term is non-constant, that is when the total order of the scheme is 2 anyway, we use the notation $(\cdot, 5, 4, 2)$ to indicate that we are in fact using a WENO reconstruction of Q.

For this scheme, the reconstruction procedure using interface averages was used, see Figure 3.1b. Since the source term integration was only second-order accurate, there was no need for the interior quadrature points, hence only two line averages were needed and thus the shared memory load was tolerable. Also, there was room in shared memory for storing the interpolated values of B at the two integration points on the current flux interface, i.e. the western interface of each cell while computing the F flux and the southern interface of each cell while computing the G flux.

In this variant, the flux computation is simpler, since the calcFluxSource function starts by reconstructing the four points needed in the central-upwind flux. However, to avoid recomputing the same values in the adjacent cell, shared memory was used for the source term. If we consider the two source term components (5.5), we are able to nicely split the source terms into one term for the reconstructed 'plus' values of h at the western boundary of the current cell, (i,j), and one term for the reconstructed 'minus' values on the western interface of the adjacent cell, i.e. $(i+1,j)$ for $S^{(2)}$ and $(i,j+1)$ for $S^{(3)}$. Exemplified for the F flux, we get

$$\begin{aligned} S^{(2)}_{i,j} &= \frac{g}{2}\left(\frac{B^E_\alpha - B^W_\alpha}{2\Delta x}\cdot h^W_\alpha + \frac{B^E_{-\alpha} - B^W_{-\alpha}}{2\Delta x}\cdot h^W_{-\alpha}\right) \\ &+ \frac{g}{2}\left(\frac{B^E_\alpha - B^W_\alpha}{2\Delta x}\cdot h^E_\alpha + \frac{B^E_{-\alpha} - B^W_{-\alpha}}{2\Delta x}\cdot h^E_{-\alpha}\right) \\ &= \hat{S}^{(2)}_{i,j}(F_{i-\frac{1}{2},j}) + \hat{S}^{(2)}_{i,j}(F_{i+\frac{1}{2},j}), \end{aligned} \tag{6.4}$$

meaning that the term $\hat{S}^{(2)}_{i,j}(F_{i-\frac{1}{2},j})$ can be computed using the B values known to the entire block, and the reconstructed values of h at the western cell interface, which is computed by thread (i,j). The term $\hat{S}^{(2)}_{i,j}(F_{i+\frac{1}{2},j})$ is computed by thread $(i+1,j)$. This way we avoid reconstructing $h^E_{\pm\alpha} = h^-_{i+1/2,j\pm\alpha}$ twice. For this to work, it is important that B is known to the entire block, which is possible using the bilinear interpolation.

On homogeneous problems, this scheme actually is of spatial order 4, and it is computationally lighter than the $(\cdot,5,4,4)$ scheme, both because of the lack of interior points and because this scheme does not perform WENO reconstruction on h in addition to Q.

6.3.2. Shared memory usage

Using the fourth-order source term quadrature, we need to store four cell averages, of h, Q_1, Q_2, Q_3. For each of these four variables we also need to store two lines averages at a time. First for the lines $(x, y_{j\pm\alpha})$ for computing the F flux, then for the lines $(x_{i\pm\alpha}, y)$ for the G flux. The complete list of shared memory arrays for the WENO method of order (3,5,4,4) is given in Table 6.1. One array could also be used for the source term since the hydrostatic correction of the fluxes is computed by more than one thread, which sums up to in total 13 arrays of size $(bw+6)\times(bh+6)$. The source term could also be computed using a variable stored in the temporary registers, which frees up some shared memory space. It has, however, no huge impact on performance.

The actual size of the shared memory arrays needs to take into account the ghost cells, which explains the added 6 cells in each direction. We refer to $bw \times bh$ as the block size and $(bw+6)\times(bh+6)$ as the shared memory size.

Shared memory arrays for method $(\cdot, 5, 4, 4)$		
Array	Size	Description
S	$(bw + 6) \times (bh + 6)$	Save values of S.
Q	$4 \times (bw + 6) \times (bh + 6)$	Store cell averages of Q and h.
$RU1$	$4 \times (bw + 6) \times (bh + 6)$	Store line averages on 'plus alpha' points
$RU2$	$4 \times (bw + 6) \times (bh + 6)$	Store line averages on 'minus alpha' points.

Table 6.1: List of the shared memory arrays used in the WENO method of order (·,5,4,4), which has a fourth-order source term.

The size of the source term array, S, need not be more than $bw \times bh$, but as just stated, this shared memory array can be skipped altogether.

The shared memory arrangement for the $(\cdot, 5, 4, 2)$ scheme is slightly different. We use one shared memory array for the source term, since two adjacent cells are cooperating on the source term. Again, this is most likely possible to omit, but this was not investigated further. Since we are reconstructing B, we need to store both the cell corner values of B, and also the two interpolated values at the current flux interface. These values will be overwritten when we are done with the F flux and ready to start on the G flux. The final shared memory usages is summarized in Table 6.2. Also here we need in total 13 arrays of size $(bw + 6) \times (bh + 6)$.

Shared memory arrays for method $(\cdot, 5, 4, 2)$**)**		
Array	Size	Description
B	$(bw + 6) \times (bh + 6)$	Store B at cell corners.
$RB1$	$(bw + 6) \times (bh + 6)$	Value of B at $-\alpha$ integration point
$RB2$	$(bw + 6) \times (bh + 6)$	Value of B at $+\alpha$ integration point
S	$(bw + 6) \times (bh + 6)$	Save values of S.
Q	$3 \times (bw + 6) \times (bh + 6)$	Store cell averages of Q.
$RU1$	$3 \times (bw + 6) \times (bh + 6)$	Store west/south interface averages, Q^R
$RU2$	$3 \times (bw + 6) \times (bh + 6)$	Store east/north interface averages of Q^L

Table 6.2: List of the shared memory arrays used in the WENO method of order (·,5,4,2), which only has a second-order source term quadrature.

6.3.3. Block size

The optimal block size depends on several parameters, and is not easy to find mathematically. Below, the optimal choices for the implemented schemes are discussed in light of the most crucial parameters. The final block sizes, to be used in experiments, were found by the means of trial and error.

Shared memory limit

Based on Table 6.1 and 6.2 it is straight forward to compute the total shared memory load. Given the block size we know the number of elements in shared memory. Each element is a number with single precision, which occupy 4 bytes. The shared memory capacity is 16 KB, which dictates an upper bound for the block size. With respect to this limit, the maximum number of elements per shared memory array in a block would be 315, or 341, depending on whether we use a shared memory array for S or not. This corresponds to a maximum block size of 15×9 or 14×11, respectively. 13 arrays and a 15×9 block size results in 15.996 KB shared memory usage, while 12 arrays and block size of 14×11 equals 15.938 KB. Using 12 arrays, blocks of size 16×9 and 15×10 also gives close to optimal shared memory usage.

Number of threads vs. warp size

Typically we want the number of threads in a block to be a multiple of the warp size, 32, since the streaming multiprocessors are launching threads one warp at a time. A block size of 16×8 gives us 128 threads which is exactly 4 warps, which is the largest possible block size without exceeding the shared memory limit.

Avoiding bank conflicts

To avoid bank conflicts when reading and writing to shared memory, the width of each shared memory array should be a multiple of 33, i.e. a block width of 27. This parameter was violated in the implementation in [7], and will be violated also here. In the shared memory configuration using 12 arrays, we would need a block size of 27×4 in order to avoid bank conflicts, which is an awkwardly sized rectangle, and did not yield good results either.

Square blocks

While the three above mentioned parameters are hardware specific, this one is more logical. Ideally we want blocks that are as square as possible, because we want the ratio of output cells to halo cells in both spatial directions to be such that the number of computations done in ghost cells is as small as possible compared to the number of computations made in the actual output cells. With the given halo, we need to perform computations in 6 ghost cells in each direction. The squarest possible block size, without exceeding the shared memory limit, would be 12×12, which also proved to perform well compared to most other block sizes, despite not being an integer multiple of 32.

6.3.3.1. Optimal block size

The feasible block sizes based on the parameters above, were all tested and the 14×11 blocks were in fact found to yield the best runtime, although not significantly better than the sizes 16×8, 16×9, 15×10 and 12×12. If using 13 shared memory arrays, the optimal sizes were 16×8, 15×9 and 12×11. Surprisingly, having block sizes of an integer multiple of the warp size, did not turn out to be as crucial as expected.

In experiments, we have settled for a block size of 14×11 for the $(\cdot,5,4,4)$ schemes, which are then using only 12 shared memory arrays, and a block size of 16×8 for the $(\cdot,5,4,2)$ schemes, which do rely on 13 shared memory arrays.

Developing code on the Quadro FX 380 GPU

On the GPU used for initial development, the Quadro FX 380 with only 16 CUDA cores and 256 MB GDDR3 memory, the compiler would not accept the maximal block sizes determined by the shared memory limit using 13 arrays, which made 16×8 or 12×11 more preferable sizes with that configuration. Although slightly smaller in terms of total size, the 12×11 blocks yielded runtime errors, resulting in 16×8 being the optimal block size on that GPU. with respect to shared memory usage, which for 13 arrays results in a shared memory usage of 15.64 KB. As it turns out, the Quadro FX 380 GPU gave runtime errors if the block size exceeded 16×8, regardless of how much shared memory was in use. It is apparent that this hardware is not capable of handling all the arithmetic involved in the WENO reconstructions. No hardware restrictions other than shared memory size, was discovered on the Quadro 5000, used for final testing.

6.3.4. Note on the intrinsic *powf()* function

Initially, the code was implemented using the intrinsic C++ function $powf()$, which obviously was called quite often in the WENO reconstruction to raise the expressions in the smoothness indicators to the power of 2, $powf(x, 2) = x^2$. As it turns out, this is an extremely slow process, perhaps not all that surprising seeing as squaring numbers, is one of the simplest variants of x^n. Substituting the $powf()$ function with the direct multiplication $x^2 = x \cdot x$ sped up the code as much as about 6-8 times. Using the *powf()* function, we were in fact able to run larger block sizes than 16×8 on the Quadro FX 380 GPU, but still at an incredibly slow speed.

6.4. The time step kernel

The timestep kernel starts by reading cell averages of Q and h at $t = t_n$, cell averages of Q and h at the last time substep, and the net change in each cell, R, from shared memory. For this kernel, there is no need for ghost cells, which makes the implementation much more straight forward than for the flux-source kernel. It simply consists of reading the data, compute the timestep and store in global GPU memory.

Since we are reconstructing h, and not B, we also need to evolve h in time, using the same net cell change as for w. For each substep in time, $i \to i + 1$, $t_n = t_0 \leq t_i \leq t_{n+1} = t_m$, $i = 0, ..., m$, the timestep kernel then compute

$$\begin{aligned}
h^n, R_1^i &\longmapsto h^{i+1} \\
w^n, R_1^i &\longmapsto w^{i+1} \\
(hu)^n, R_2^i &\longmapsto (hu)^{i+1} \\
(hv)^n, R_3^i &\longmapsto (hv)^{i+1},
\end{aligned}$$

using a, preferably TVD, Runge-Kutta method. In the implementation in [7], both the classic Euler's method and the second-order TVD Runge-Kutta (3.10) are supported. In the current high-order scheme, also the third-order Runge-Kutta (3.11) is available.

The time step kernel is also responsible for computing the bed friction source term, leading to a semi-implicit approximation of $S_F(Q)$. This part of the implementation was left unchanged from [7].

Also, in SINTEF's simulator the time step kernel was set to use the same block size as the flux-source kernel, a configuration we will leave unchanged.

6.5. Boundary conditions

So far, we have avoided the treatment of boundary conditions. The (2,2,2,2) scheme by Brodtkorb has support for wall, outflow and inflow conditions. The implementation of the former is the same also in a high-order setting, the cell averages in the ghost cells are set to the same value as its counterpart in the interior domain, except for the velocity component normal to the boundary interface, which changes sign.

Outflow and inflow conditions turns out to be more difficult if we want the flux kernel to remain oblivious of boundaries. In the second-order reconstruction, the approach is to simply set the two ghost cells equal to the interior cell at the boundary. That way, all the derivatives in the ghost cells closest to the boundary will be zero. In the high-order scheme, however, if x_3 and x_{N+2} are the cells at the domain boundaries on a one-dimensional domain, we need to specify values in the ghost cells, x_0, x_1, x_2 and x_{N+3}, x_{N+4}, x_{N+5}, such that there is no ingoing flow at the boundary, i.e. the derivatives in cells x_2 and x_{N+3} should be zero. But simply setting all the ghost cells equal to the cell at the boarder, is not sufficient, since the stencils for the cells x_{N+3} and x_2, depend on the cells x_{N+1} and x_1, which generally are not equal to the cells x_3, x_{N+2}. Consequently, four out of five points in the WENO stencils are equal, while the fifth differs from the rest. When applying this naive implementation, a wave front leaving the domain will create a small reflecting wave, which possibly could destroy the computations in the interior cells.

7. Numerical results

To highlight the properties of the high-order schemes on the shallow-water equations (2.9), we consider a variety of problems, from discontinuous and homogenous ones to smooth and non-homogeneous. We will measure the code performance, numerically verify the order of convergence, investigate the well-balanced properties and, perhaps most importantly, compare the accuracy of the high-order schemes with that of the second-order scheme, in light of the runtime differences.

7.1. Code performance

We start off by comparing the performance of the (2,2,2,2) and the higher order schemes, seeing as that sets a premiss for the rest of the discussion. The hardware used for the benchmarks was an Intel Core i7-3939K CPU with 3.20GHz and 64 GB RAM. The GPU was a NVIDIA Quadro 5000 with 2.5 GB GDDR5 frame buffer memory in a PCI-express 2.0 x16 slot.

The runtimes were measured in *simulated time per wall clock second.* The simulation time unit was taken to be one time step using the second-order Runge-Kutta method on a 100×100 grid, which we will denote

$$\Delta t_{n=100}.$$

Since the eigenvalues computed by the different methods are approximately the same, this time unit will also be approximately the same, regardless of the numerical scheme, which makes it suitable as a reference unit.

The reference unit is essentially a modification of the quantity *iterations per second*, which is an already in-build feature of the simulator created by Brodtkorb. We recall that the time step, Δt, is computed by the CFL condition (6.3). Provided that the eigenvalues obtained for the different schemes and grids, are about equal, the time step is scaled according to the scaling of Δx and Δy. Halving the cell size in either direction, results in a halving of the time step, which means that twice as many iterations need to be carried out to simulate the same time interval, $t \in [0, T]$.

To convince ourselves that a difference in the maximum eigenvalues would not have dramatic consequences for the runtime measurements, have in mind that the quantity 'simulated time per wall clock second', typically is in the range $10^1\ \Delta t_{n=100}$ - $10^3\ \Delta t_{n=100}$ in our experiments, and an error in the maximum eigenvalue caused by the flux computations, probably would be about $10^3 - 10^6$ times as small in absolute value. In the experiments performed for this thesis, no significant differences in the maximum eigenvalues have been discovered.

The third-order Runge-Kutta obviously spends more time on each iteration, seeing as it consists of one more sub step, but we are also able to increase the CFL condition to $C_{cfl} = 0.5$, which results in a computational gain, seeing as we are increasing the workload by 50 % and the time step length by 100 %. To ensure stability, we have used $C_{clf} = 0.45$ in the computations, as done in [28]. One time step performed by the third-order Runge-Kutta on a 100×100 grid, then amounts to

$$\Delta t^{RK3}_{n=100} = \frac{0.45}{0.25} \Delta t^{RK2}_{n=100}.$$

The runtimes using the third-order Runge-Kutta method are not included here, as they follow directly from this relation.

The runtimes of the implemented schemes scaled to the reference time unit, are compared to the second-order Kurganov-Petrova scheme in Table 7.1. We see how the high-order WENO methods are somewhere between 3 and 5 times slower than the second-order scheme on a given grid size. The difference between the (2,5,4,2) and the (2,5,4,4) schemes is mostly due to the difference in block size in the flux-source kernel. The 14×11 blocks makes the (2,5,4,4) scheme slightly faster. Running the (2,5,4,4) scheme on 16×8 blocks, as used by the (2,5,4,2) scheme, yields close to identical runtimes. The (2,2,2,2) scheme has the advantage of using 16×12 blocks, but it is still way faster when reducing the block size to 16×8 or 14×11.

From these results it is clear that in order to achieve a speed-up, we need the high-order schemes to, at least, match the accuracy of the second-order scheme, but by halving the number of grid cells in each direction. The high-order schemes on a $n \times n$ grid are about 1.3-1.7 times faster than the second-order scheme on a $(2n) \times (2n)$ grid. If we manage to further increase accuracy, so that the high-order schemes on a $n \times n$ grid match the second-order

Grid size	**Runtime $[\Delta t^{n=100}_{RK2}/s]$**		
	(2,2,2,2)	(2,5,4,2)	(2,5,4,4)
2200×2200	0.8	-	-
1100×1100	6.5	-	1.3
550×550	-	-	9.8
400×400	112.1	23.5	25.0
300×300	244.3	55.4	56.8
200×200	698.1	163.9	172.3
150×150	1336.6	362.1	381.1
100×100	2854.5	904.9	890.2
50×50	-	3872.6	3939.6

Table 7.1: Comparison of runtimes measured in simulated time steps of length $\Delta t_{n=100}$ per wall clock second, for the second-order Kurganov-Petrova method and the high-order WENO methods with second- and fourth-order source term quadrature.

scheme on a $(3n) \times (3n)$ grid, we get speed-ups by a factor around 3.0-3.5. In the extreme case, when the high-order scheme on a $n \times n$ grid is comparable to the second-order scheme on a $(4n) \times (4n)$ grid, we can get a speed-up by a factor 5 for the lower grids and up to a factor 12 for the largest grids.

Note that since we have used the time unit $\Delta t_{n=100}$, the results in Table 7.1 are problem independent. Thus, the discussion above is rather general.

The smallest grid sizes included in Table 7.1 are the ones suited for solving the test problems done for this thesis. The largest grids are included due to the conclusion in [7] that the best utilization of the GPU capacity is reached for larger grids, in the sense that it can process more cells per second. It is comforting to see that the possible speed-ups from using the high-order methods, actually increase slightly on the largest grids. During testing, the largest successfully executed domain size was 7236×7236, which equals over 52 million grid cells, about 60 % more than benchmarked for the second-order scheme in [7], which fits well considering the Quadro 5000 GPU has about 67 % more memory available. For these large grids, the second-order scheme runs at almost 100 megacells/s, about 5-6 times as fast as the high-order schemes. These benchmarks should perhaps also be done on a state-of-the-art GPU, but the runtime ratios were similar both on the old Quadro FX 380 and the Quadro 5000, and would be expected to be so also on faster hardware.

7.2. The idealised circular dam break problem

First we consider the idealised circular dam break problem on the domain $\Omega = [-40, 40] \times [-40, 40]$, with the initial data

$$
\begin{aligned}
Q_1(x,y,0) &= \begin{cases} 2.5, & x^2+y^2 < 2.5 \\ 0.5 & \text{else} \end{cases} \\
Q_2(x,y,0) &= 0, \\
Q_3(x,y,0) &= 0,
\end{aligned}
\tag{7.1}
$$

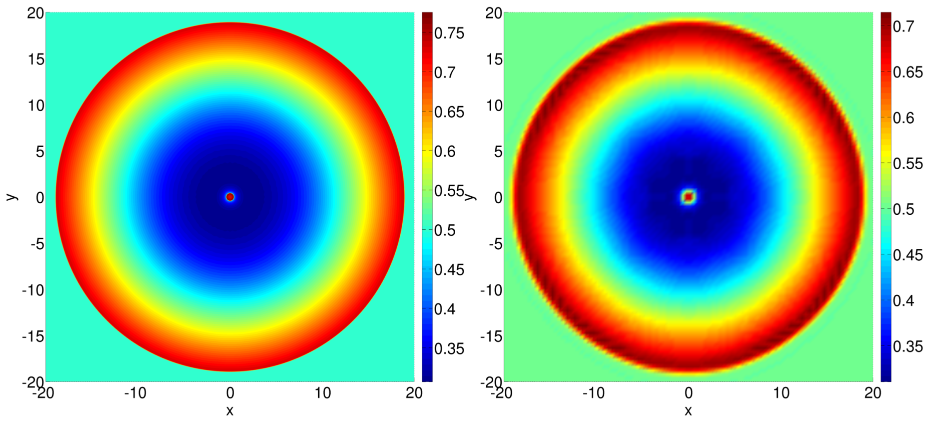
and with flat bottom topography, $B(x,y) = 0$. The problem is taken from [29], where an extensive numerical analysis is found, and variants of this problem appear frequently in the literature. The initial data (7.1) creates a shock wave moving away from the origin, while a rarefaction wave moves towards the origin. Around $t = 4.0$ the water elevation at the origin has 'bored' almost all the way down to the bottom, $h = Q_1 = 0$. The created 'hole' is rapidly filled which causes the surrounding water to pile up at the origin, creating a second outgoing wave.

With these initial data, wall boundary conditions and for sufficiently low times, t, we avoid the potential problem of dry states, $h = 0$. As no analytical solution exists, we compare the results with a reference solution, obtained by the scheme of order (2,2,2,2) on a 1600×1600 grid.

The full solution after $t = 4.7s$ is shown in Figure 7.1. At this point the water front is approaching the boundary and a second wave is coming from the origin. The high peaks are somewhat smeared out in the coarse solution and note also that the Q_1 range is smaller, meaning that the high peaks are underestimated.

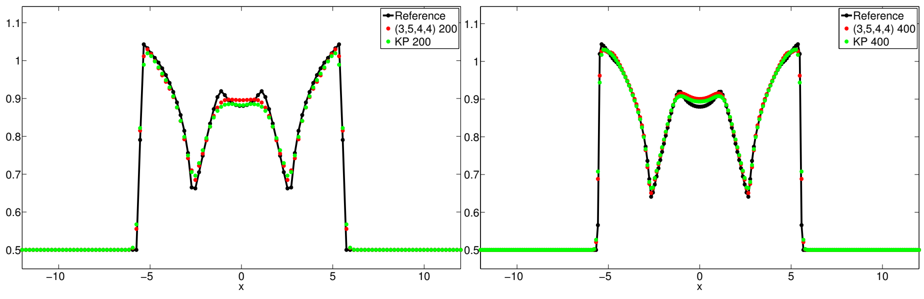
The dam break problem is a homogeneous problem, and thus it is more than sufficient to use the $(\cdot, 5, 4, 2)$ schemes, with a second-order source term quadrature and bilinear interpolation of B. To illustrate the properties of the (2,5,4,2) scheme compared to the (2,2,2,2) scheme, we will investigate the solution at two times, $t = 0.7s$ and $t = 4.7s$. The times might seem somewhat arbitrarily chosen, but they coincide with solution plots in [29], which serves as an extra reference.

As seen in Figure 7.2, at $t = 0.7$ we avoid spurious oscillations, the discontinuities and the general height and shape of the middle peak are approximated well, but we lose the fine structure of the 'mountain top' around the origin, and even on a 400×400 grid we struggle to capture this correctly. The noticeable



(a) Reference (b) (2,5,4,2) 100

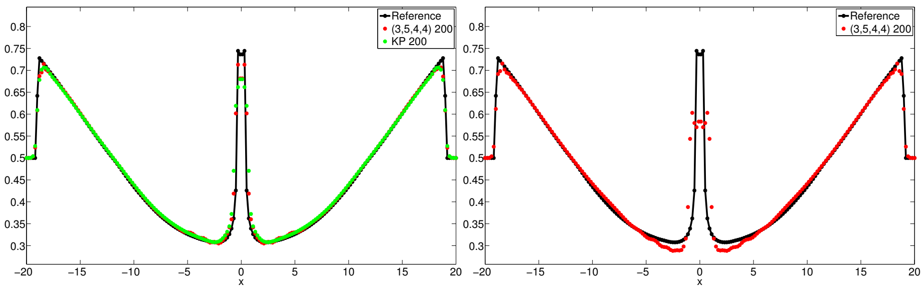
Figure 7.1: The complete solution of Q_1 after $t = 4.7s$ for the fine grid reference solution and the (2,5,4,2) scheme on a 100×100 grid.



(a) 200×200 (b) 400×400

Figure 7.2: Cross-section at $y = 0$, at time $t = 0.7s$, of the (2,5,4,2) and the (2,2,2,2) on two grid sizes, compared to the reference solution.

differences between the 200×200 and 400×400 grids are restricted to this fine structure around the origin. On the 200×200 grid, we see that the (2,2,2,2) scheme and (2,5,4,2) scheme are under-estimating and averaging it, respectively. On the 400×400 grid, the (2,5,4,2) scheme is able to capture the two small peaks, while failing on the valley in between. Here, the (2,2,2,2) scheme is the one averaging. The plots are also indicating that the (2,5,4,2) scheme, perhaps surprisingly, is slightly better at approximating the discontinuities of the first derivative around $x = \pm 3$ and $x = \pm 5$.



(a) (2,5,4,2) vs (2,2,2,2) (b) (3,5,4,4)

Figure 7.3: Cross-section around $y = 0$, at time $t = 4.7s$, The (2,5,4,2) and the (2,2,2,2) schemes against the reference solution on a 200×200 grid. The (2,5,4,4) scheme fails.

Slightly different results are found for $t = 4.7s$. Now some disturbing oscillations in the neighbourhood of the middle peak are present in the (2,5,4,2) scheme, although with rather small amplitudes. On the other hand, the thin, high peak around the origin is approximated slightly better by the high-order scheme. It takes a 400×400 grid to capture the height accurately.

So far, all the solutions obtained by the (·,5,4,4) schemes, using the hydrostatic reconstruction, have been close to identical to the ones by the (2,5,4,2) scheme. At $t = 4.7$, however, we are experiencing some extremely troubling results. Close to the origin the method is failing miserably to capture both the high peak itself and the area around it. These problems seem to occur as h approaches zero, which indeed is the case around the origin for t around 4.0, i.e. right before the creation of the high, thin peak seen in Figure 7.3. That a high-order scheme, with no particular method for handling dry states, fails as $h \to 0$ is not at all remarkable. In that regard, it is more correct to say that the (2,5,4,2) scheme is surprisingly well-behaved. Unfortunately, no proper explanation of this difference between the WENO schemes has been found.

The comparison of the (2,5,4,2) and the Kurganov-Petrova scheme of order (2,2,2,2) on the dam break problem, clearly shows how we are not getting more accurate results by using a high-order scheme. As expected, higher level of smoothness in the reconstruction procedure, does not help on a highly non-smooth problem, but the dam break problem shows how WENO-based high-resolution methods are, for the most part, handling discontinuities well.

7.3. Accuracy and verification of order

To measure the errors and order of converge of the methods, it is common to fix the time step, Δt, to a value low enough to ensure stability for all grid sizes with respect to the CFL condition, and also to ensure that the spatial error is dominating the problem. This is a good way of verifying the spatial order of the method after a relatively short simulation time. A different approach would be to use the maximal time steps dictated by the CFL condition, meaning that the time step length would be smaller on finer grids. This approach is used in [28], and is arguably better for testing the performance in a more realistic setting, since the methods are run as they would be in practical experiments. Here, both approaches are used, and we will clearly state which one in each case. While the to approaches yield close to identical results after short simulation time, the former actually failed to yield more than order 2 for longer simulations times. Naturally, the total error is smaller when fixing Δt to a small value, but the convergence rates are the same. For the second-order Runge-Kutta method we choose the CFL coefficient, $C_{cfl} = 0.25$ in all examples, whereas, for the third-order Runge-Kutta, $C_{cfl} = 0.45$ is used.

For most of the results presented here, the (3,5,4,4) method on a 1600×1600 was taken to be the reference solution, but we would like to remark that using reference solutions produced by any other method, including the (2,2,2,2) method, does not lead to significantly different results. The error is measured against the reference solution in the induced L^1 matrix norm,

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} |A_{ij}|, \quad A \in \mathbb{R}^{m\times n},$$

which is the norm of choice in most articles on the topic of conservation laws, quite logically because in conservation laws, the integrals of the functions involved, are of particular interest. To compute the order of convergence, we first measure the error of a solution obtained on a grid with cell size Δx_0. We then compute the error on grids with cell sizes $\Delta x_i = \Delta x_0/2^i$, $i = 1, \ldots.$ Since we are dealing with finite-volume methods, we need to compare the errors on the same volume in the physical domain, which means that we are comparing the errors, measured against the reference solution, on a volume corresponding to one cell in the coarsest domain, Δx_0, thus all the fine grid solutions must be locally averaged to fit the coarsest grid. The order of convergence obtained at grid i is computed by

$$r_i = \log_2 \left(\frac{E(\Delta x_{i-1})}{E(\Delta x_i)} \right),$$

where $E(\Delta x_i)$ is the L_1 error of the solution on a grid of size $\Delta x = \Delta x_i$. Alternatively we could compute the order without the use of a reference solution, by the formula

$$r_i = \log_2 \left(\frac{|E(\Delta x_{i-1}) - E(\Delta x_{i-2})|}{|E(\Delta x_i) - E(\Delta x_{i-1})|} \right).$$

The results using this approach were satisfying, but will not be included here.

Most of the examples used to verify order of convergence in the literature, involves periodic boundary conditions. We will, on the other hand, use exponential functions in the initial data which tend to zero at the boundaries, which means that these tests are valid for various types of boundary conditions, including periodic, outflow and wall conditions. The most important thing is to avoid creation of shock waves at the boundaries, which happens when the water waves hit a wall. Ensuring that all components of Q are zero close to the boundaries for low times, wall boundaries will not be a problem.

Circular dam break problem

To properly test the convergence order of a method, test problems must be chosen with special care. For example, when applied to the dam break problem (7.1), in which the initial data are discontinuous for Q_1 and non-smooth (only C^0) for Q_2 and Q_3, a high-order method does not produce convergence rates of more than 1, in the L^1 norm, which is also verified for a high-order method on a discontinuous problem in [28]. Table 7.2 shows convergence results for Q_1 at two different times, using

$$\Delta t = 0.0001,$$

and clearly confirms convergence of order 1.

Grid size	$t = 0.1s$		$t = 4.7s$	
	Error	Order	Error	Order
50×50	$2.94 \cdot 10^{-2}$	-	$2.59 \cdot 10^{-2}$	-
100×100	$1.39 \cdot 10^{-2}$	1.12	$1.17 \cdot 10^{-2}$	0.57
200×200	$7.95 \cdot 10^{-3}$	0.80	$9.35 \cdot 10^{-3}$	0.89
400×400	$2.42 \cdot 10^{-3}$	1.71	$1.53 \cdot 10^{-3}$	2.61

Table 7.2: Verification of the order of the implemented (2,5,4,2) scheme on the dam break problem (7.1). Here only $Q_1 = w = h - B$, is considered, but similar results are found for Q_2 and Q_3. The errors were measured in the L^1 norm.

Homogeneous problem with smooth data in one component

Next, we consider a smooth version of the dam break, where we assume the initial water pile to be shaped as a Gaussian function,

$$\begin{aligned} Q_1(x, y, 0) &= 1 + e^{\frac{-x^2-y^2}{3}} \\ Q_2(x, y, 0) &= Q_3(x, y, 0) = 0. \end{aligned} \tag{7.2}$$

This problem still models the dam break setting, since the initial moments, Q_2, Q_3 are zero. We now have continuous initial data in all components, but only $Q_1(x, y, 0)$ is properly smooth, and as seen in Table 7.3, we are only able to achieve second-order convergence. Again, we have used $\Delta t = 0.0001$.

$t = 0.1$, (3,5,4,4)th order scheme

Grid size	Q_1	
	Error	Order
50×50	$4.23 \cdot 10^{-3}$	-
100×100	$1.02 \cdot 10^{-3}$	2.05
200×200	$2.85 \cdot 10^{-4}$	1.84
400×400	$7.10 \cdot 10^{-5}$	2.01

Table 7.3: Verification of the order of the implemented (3,5,4,4) scheme on the smooth dam break problem (7.2), for component Q_1. The errors were measured in the L^1 norm.

Homogeneous problem with smooth data in all components

In order to achieve maximal possible order of convergence, the schemes were tested on a problem with smooth initial data in all components,

$$\begin{aligned} Q_1(x,y,0) &= 1 + e^{\frac{-x^2-y^2}{4}} \\ Q_2(x,y,0) &= Q_3(x,y,0) = e^{\frac{-x^2-y^2}{4}} \end{aligned} \tag{7.3}$$

We still use $B(x,y) = 0$, but since $Q_1 = h + B$, Q_1 is smooth even for constant bottom topography. With this setup we expect to utilize the maximum order of the spatial components. Hence, as long as the error of the reconstruction and numerical flux dominate the total error, we will essentially have a fourth-order method. The formal order however, is restricted by the lowest order component, which in this case is the order of the Runge-Kutta method. Over time the total error will be dominated by the least accurate component of the method.

First we set the time steps to

$$\Delta t = 0.0001,$$

which is low enough for simulations to fulfill the CFL condition for all grid sizes, and tested the methods after $t = 0.1$. The results for the (2,5,4,·) schemes are found in Table 7.4, where the errors and convergence analysis for the second-order Kurganov-Petrova scheme is included, for comparison.

We notice right away that for small times, the order is close to, or even above, the theoretical spatial order of 4. The convergence rates using the third-order Runge-Kutta are not shown, seeing as they do not differ noticeably from the results obtained by the (2,5,4,·) method.

To benchmark the methods after $t = 1.0$, we let the time steps be controlled by the CFL condition (6.1), (6.2), (6.3). The results for the two high-order methods using hydrostatic reconstruction, are found in Table 7.5. At $t = 1.0$, the total order drops to around 2.5-3.0, and thus we conclude that, at this time, the error caused by the Runge-Kutta method, is more prominent in the total error. By using the third-order Runge-Kutta method we are able to increase the order slightly. Particularly noteworthy is the drastic drop in convergence rate from 200^2 to 400^2 cells for the water elevation component, Q_1. It seems as a further decrease of error below $\sim 10^{-4}$ is difficult, in the L^1 norm. It is, however, not surprising that the L^1 error increases for larger grids, and we see in Table 7.6 that in the max, or L^∞, norm, defined by

Convergence rates at $t = 0.1$, problem (7.3)

Grid size	Q_1		Q_2		Q_3	
	Error	Order	Error	Order	Error	Order
			(2,5,4,2) scheme			
25×25	$1.28 \cdot 10^{-1}$	-	$5.32 \cdot 10^{-1}$	-	$5.88 \cdot 10^{-1}$	-
50×50	$1.28 \cdot 10^{-2}$	3.32	$7.16 \cdot 10^{-2}$	2.89	$9.63 \cdot 10^{-2}$	2.61
100×100	$9.92 \cdot 10^{-4}$	3.69	$3.19 \cdot 10^{-3}$	4.49	$5.55 \cdot 10^{-3}$	4.12
200×200	$3.04 \cdot 10^{-5}$	5.03	$7.56 \cdot 10^{-5}$	5.40	$1.29 \cdot 10^{-4}$	5.43
400×400	$1.56 \cdot 10^{-6}$	4.29	$2.72 \cdot 10^{-6}$	4.80	$3.68 \cdot 10^{-6}$	5.13
			(2,2,2,2) scheme			
25×25	$2.48 \cdot 10^{-1}$	-	$4.44 \cdot 10^{-1}$	-	$5.54 \cdot 10^{-1}$	-
50×50	$8.27 \cdot 10^{-2}$	1.59	$6.43 \cdot 10^{-2}$	2.79	$3.31 \cdot 10^{-2}$	3.13
100×100	$1.57 \cdot 10^{-2}$	2.40	$2.83 \cdot 10^{-2}$	1.18	$2.43 \cdot 10^{-2}$	1.37
200×200	$3.11 \cdot 10^{-3}$	2.33	$5.50 \cdot 10^{-3}$	2.36	$6.75 \cdot 10^{-3}$	1.85
400×400	$6.07 \cdot 10^{-4}$	2.36	$1.42 \cdot 10^{-3}$	1.95	$1.89 \cdot 10^{-3}$	1.84

Table 7.4: Comparison of the implemented (2,5,4,2) scheme and the (2,2,2,2) scheme against the reference solution on a 1600×1600 grid, on the smooth test problem (7.3) after $t = 0.1$. The errors were measured in the L^1 norm.

$$\|A\|_{\max} = \max_{ij} |A_{ij}|, \quad A \in \mathbb{R}^{m \times n},$$

we obtain an order of convergence closer to expectations.

Based on these results, it is clear that, for low times, on this constructed smooth problem, the high-order schemes clearly outperforms the (2,2,2,2) scheme with respect to the total L^1 error. Using the (2,5,4,4) scheme on a 200×200 grid is sufficient to ensure higher level of accuracy in all components compared to the (2,2,2,2) scheme on a 400×400 grid, and we can even argue that a 100×100 grid might be enough for most practical purposes. For larger times, the convergence rate drops for the high-order schemes, but as long as we maintain around order 3, we can still produce equally accurate results on half the grid size as with the (2,2,2,2) scheme.

Convergence rates at $t = 1.0$, problem (7.3)

Grid size	Q_1		Q_2		Q_3	
	Error	Order	Error	Order	Error	Order
			(2,5,4,4) scheme			
25×25	$1.90 \cdot 10^{-1}$	-	$7.09 \cdot 10^{-1}$	-	$5.34 \cdot 10^{-1}$	-
50×50	$2.91 \cdot 10^{-2}$	2.70	$1.16 \cdot 10^{-1}$	2.61	$7.89 \cdot 10^{-2}$	2.76
100×100	$3.93 \cdot 10^{-3}$	2.89	$1.53 \cdot 10^{-2}$	2.92	$1.26 \cdot 10^{-2}$	2.65
200×200	$8.14 \cdot 10^{-4}$	2.27	$2.92 \cdot 10^{-3}$	2.39	$2.61 \cdot 10^{-3}$	2.27
400×400	$2.09 \cdot 10^{-4}$	1.96	$7.28 \cdot 10^{-4}$	2.01	$6.42 \cdot 10^{-4}$	2.02
			(3,5,4,4) scheme			
25×25	$2.03 \cdot 10^{-1}$	-	$7.20 \cdot 10^{-1}$	-	$5.48 \cdot 10^{-1}$	-
50×50	$2.28 \cdot 10^{-2}$	2.83	$1.13 \cdot 10^{-1}$	2.68	$8.03 \cdot 10^{-2}$	2.77
100×100	$2.54 \cdot 10^{-3}$	3.49	$9.33 \cdot 10^{-3}$	3.59	$7.69 \cdot 10^{-3}$	3.38
200×200	$3.25 \cdot 10^{-4}$	2.97	$5.84 \cdot 10^{-4}$	4.00	$5.70 \cdot 10^{-4}$	3.76
400×400	$1.95 \cdot 10^{-4}$	0.74	$1.18 \cdot 10^{-4}$	2.31	$1.14 \cdot 10^{-4}$	2.32

Table 7.5: Comparison of the implemented (·,5,4,4) schemes against the reference solution on a 1600×1600 grid, on the smooth test problem (7.3) after $t = 1.0$. The errors were measured in the L^1 norm.

$t = 1.0$, (3,5,4,4)th order scheme

Grid size	Q_1	
	Error	Order
25×25	$4.29 \cdot 10^{-2}$	-
50×50	$8.39 \cdot 10^{-3}$	2.35
100×100	$8.54 \cdot 10^{-4}$	3.30
200×200	$7.02 \cdot 10^{-5}$	3.60
400×400	$1.60 \cdot 10^{-5}$	2.13

Table 7.6: Verification of the order of the implemented (3,5,4,4) scheme on the smooth test problem (7.3) after $t = 1.0$. Errors are measured in the the max norm.

Non-homogeneous problem with smooth data in all components

In all the test problems so far, the bottom topography has been constant, which results in a homogeneous system, $S \equiv 0$. Thus, the source term quadrature is irrelevant to the accuracy of the method, and the results obtained by the (·,5,4,4) schemes are all identical to the ones obtained by the (·,5,4,2) schemes. To test the fourth-order source term quadrature, we need to consider a problem involving a smoothly varying bottom topography, such as

$$\begin{aligned} Q_1(x,y,0) &= 1 + 2\,e^{\frac{-x^2-y^2}{10}} \\ Q_2(x,y,0) &= Q_3(x,y,0) = e^{\frac{-x^2-y^2}{4}} \\ B(x,y) &= e^{\frac{-x^2-y^2}{15}}. \end{aligned} \tag{7.4}$$

Again we use Gaussian functions, to avoid problems close to the boundaries. We let Δt be controlled by the CFL condition, either $C_{cfl} = 0.25$ or $C_{cfl} = 0.45$, depending on the Runge-Kutta method. Problems involving varying bottom topography are arguably more relevant in a real-world setting, and we will therefore provide a slightly better comparison of the second-order and high-order schemes than we did in the homogeneous case.

Results after $t = 0.1$ for problem (7.4), using the (3,5,4,4) method, are found in Table 7.7, where a convergence rate close to 4 is verified. Also the results using the (2,2,2,2) scheme are included, for comparison. Once again, we experience a drop of convergence rate for Q_1 on the 400×400 grid, and partly also on the 200×200 grid, but, as before, the expected order of convergence is obtained in the L^∞ norm. The results using the second-order Runge-Kutta for $t = 0.1$ are omitted, as they were almost identical. For the coarsest test grid, 25^2 cells, only 1 time step was needed to reach $t = 0.1$, whereas on the reference grid, 1600^2 cells, we needed 37 time steps to complete the simulation.

The order verification computations rely on the number of grid cells being doubled. In addition, to properly measure the errors, we need all grid sizes, including the reference solution, to be an integer multiple of the coarsest grid size. For evaluating the accuracy of the methods, we would also like some intermediate grid sizes. We get that by using a reference solution computed by the (3,5,4,4) scheme on a 1500×1500 grid, and measuring the error on the grids 150×150 and 300×300, see Table 7.8. This change of reference solution has close to no effect on the errors on the grid sizes used in the convergence analysis, and we conclude that it makes sense to compare errors based on these

Convergence rates at $t = 0.1$, problem (7.4)

Grid size	Q_1		Q_2		Q_3	
	Error	Order	Error	Order	Error	Order
			(3,5,4,4) scheme			
25×25	$3.48 \cdot 10^{-2}$	-	$8.59 \cdot 10^{-2}$	-	$1.55 \cdot 10^{-1}$	-
50×50	$3.70 \cdot 10^{-3}$	3.23	$6.69 \cdot 10^{-3}$	3.68	$1.36 \cdot 10^{-2}$	3.52
100×100	$1.33 \cdot 10^{-4}$	4.79	$4.78 \cdot 10^{-4}$	3.81	$7.91 \cdot 10^{-4}$	4.10
200×200	$1.86 \cdot 10^{-5}$	2.84	$2.99 \cdot 10^{-5}$	4.00	$4.80 \cdot 10^{-5}$	4.04
400×400	$1.33 \cdot 10^{-5}$	0.49	$3.23 \cdot 10^{-6}$	3.21	$5.75 \cdot 10^{-6}$	3.06
			(2,2,2,2) scheme			
25×25	$6.06 \cdot 10^{-2}$	-	$1.36 \cdot 10^{-1}$	-	$1.71 \cdot 10^{-1}$	-
50×50	$1.63 \cdot 10^{-2}$	1.90	$2.94 \cdot 10^{-2}$	2.26	$4.16 \cdot 10^{-2}$	2.04
100×100	$3.57 \cdot 10^{-3}$	2.19	$1.38 \cdot 10^{-2}$	1.04	$1.42 \cdot 10^{-2}$	1.55
200×200	$9.01 \cdot 10^{-4}$	2.45	$3.18 \cdot 10^{-3}$	2.12	$3.55 \cdot 10^{-3}$	2.00
400×400	$1.66 \cdot 10^{-4}$	2.35	$5.82 \cdot 10^{-4}$	2.45	$6.76 \cdot 10^{-4}$	2.39

Table 7.7: Comparison of the implemented (3,5,4,4) scheme and the (2,2,2,2) scheme against the reference solution on a 1600×1600 grid, on the smooth test problem (7.4) after $t = 0.1$. The errors were measured in the L^1 norm.

two reference grids.

By comparing the (3,5,4,4) scheme to the (2,2,2,2) scheme on grids 50×50, 100×100, 150×150, 200×200, or on the grids 100×100, 200×200, 300×300, 400×400, we can see the effect of increasing the number of grid cells from n^2 to $(2n)^2$, $(3n)^2$ and $(4n)^2$.

For $t = 0.1$, on problem (7.4), running the (3,5,4,4) scheme on a 100×100 grid gives more accuracy in all three components, than running the (2,2,2,2) scheme on a 300×300 grid, and the accuracy is not far behind what we would get by running the (2,2,2,2) scheme on a 400×400 grid. Taken runtimes from Table 7.1 into account, this shows that given some tolerance in the L^1 norm, we can, by using the (2,5,4,4) scheme instead of the (2,2,2,2) scheme, achieve a speed-up somewhere between 3 and 5, for these small grids. For problems of larger scale, where a 1600×1600 grid would be no way near enough to produce a proper reference solution, the speed-up would be slightly higher.

In this discussion, it is probably not fair to consider runtimes using the

Errors at $t = 0.1$, problem (7.4)

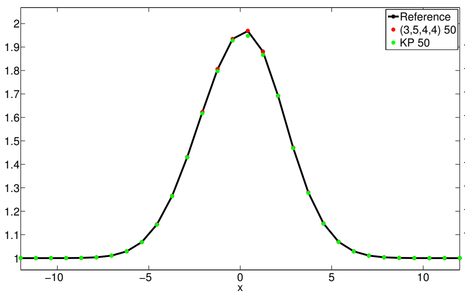
Grid size	Q_1	Q_2	Q_3
	(3,5,4,4) scheme		
50×50	$3.70 \cdot 10^{-3}$	$6.67 \cdot 10^{-3}$	$9.59 \cdot 10^{-3}$
100×100	$1.33 \cdot 10^{-4}$	$4.78 \cdot 10^{-4}$	$5.86 \cdot 10^{-4}$
150×150	$3.02 \cdot 10^{-5}$	$9.18 \cdot 10^{-5}$	$1.01 \cdot 10^{-4}$
300×300	$1.41 \cdot 10^{-5}$	$6.90 \cdot 10^{-6}$	$8.29 \cdot 10^{-6}$
	(2,2,2,2) scheme		
50×50	$1.63 \cdot 10^{-2}$	$2.84 \cdot 10^{-2}$	$4.16 \cdot 10^{-2}$
100×100	$3.57 \cdot 10^{-3}$	$1.38 \cdot 10^{-2}$	$1.42 \cdot 10^{-2}$
150×150	$1.59 \cdot 10^{-3}$	$5.99 \cdot 10^{-3}$	$6.48 \cdot 10^{-3}$
300×300	$3.55 \cdot 10^{-4}$	$1.21 \cdot 10^{-3}$	$1.40 \cdot 10^{-3}$

Table 7.8: Comparison of the implemented (3,5,4,4) scheme and the (2,2,2,2) scheme against the reference solution on a 1500×1500 grid, on the smooth test problem (7.4) after $t = 0.1$. The errors were measured in the L^1 norm.

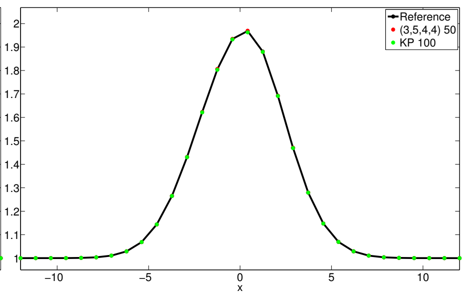
third-order Runge-Kutta, since it is not effecting accuracy, meaning that the gain would be only with respect to runtime, and could very well also be implemented in the second-order scheme.

To illustrate the differences in accuracy between the (3,5,4,4) scheme and the (2,2,2,2) scheme, we consider a cross section at $t \in [0.00, 0.04]$ at $t = 0.01$. We have compared the two methods in the first two variables in Figure 7.4 and 7.5. In most of the domain, the (2,2,2,2) scheme is accurate enough, except for around the function maximum, which is underestimated by the low-order scheme. Note that errors below 10^{-3} are close to invisible with the current axis scaling. The solution of Q_3 is identical to the one of Q_2 in shape and size, just aligned along the y axis instead of the x axis. All results are transferred to the coarsest grid, so that we are comparing the same volume.

In problem (7.4), the conserved variables, Q, evolve smoothly in time up until around $t = 2.0$, where the first dry states appear. As $t \to 2.0$, the behaviour is not as nice as for the Kurganov-Petrova scheme, which supports dry states, so to avoid that problematic region, we evaluate the methods at $t = 1.0$. The convergence rates and errors of the ($\cdot$,5,4,4) schemes compared to the (2,2,2,2) scheme are found in Table 7.9, Again, Δt is determined according to the CFL condition.

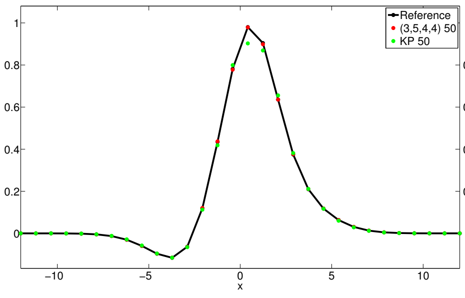


(a) WENO 50 vs KP 50, Q_1

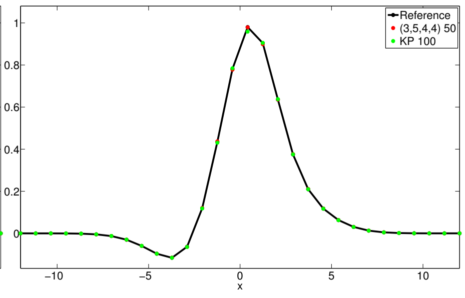


(b) WENO 50 vs KP 100, Q_1

Figure 7.4: Comparison of the (3,5,4,4) scheme on a 50×50 grid and the (2,2,2,2) scheme on 50×50 and 100×100 grids on problem (7.4) after $t = 0.1$, for the cross section $y \in [0.00, 0.04]$.



(a) WENO 50 vs KP 50, Q_2



(b) WENO 50 vs KP 100, Q_2

Figure 7.5: Comparison of the (3,5,4,4) scheme on a 50×50 grid and the (2,2,2,2) scheme on 50×50 and 100×100 grids on problem (7.4) after $t = 0.1$, for the cross section $y \in [0.00, 0.04]$.

We see that we obtain a convergence rate of about 3 for the high-order schemes. The error is slightly smaller for the finest grids, using a third-order Runge-Kutta method, although nothing spectacular, due to the (2,5,4,4) scheme performing better than the expected second-order accuracy. Clearly the high-order schemes produce better results than the Kurganov-Petrova scheme, despite only using Runge-Kutta methods of order 2 and 3.

Convergence rates at $t = 1.0$, problem (7.4)

Grid size	Q_1		Q_2		Q_3	
	Error	Order	Error	Order	Error	Order
(3,5,4,4) scheme						
25×25	$2.13 \cdot 10^{-1}$	-	$7.94 \cdot 10^{-1}$	-	$4.95 \cdot 10^{-1}$	-
50×50	$3.53 \cdot 10^{-2}$	2.59	$1.47 \cdot 10^{-1}$	2.44	$9.28 \cdot 10^{-2}$	2.41
100×100	$3.88 \cdot 10^{-3}$	3.19	$1.35 \cdot 10^{-2}$	3.44	$1.25 \cdot 10^{-2}$	2.90
200×200	$4.53 \cdot 10^{-4}$	3.10	$9.67 \cdot 10^{-4}$	3.81	$8.24 \cdot 10^{-4}$	3.92
400×400	$2.29 \cdot 10^{-4}$	0.98	$1.11 \cdot 10^{-4}$	3.12	$1.23 \cdot 10^{-4}$	2.75
(2,5,4,4) scheme						
25×25	$1.97 \cdot 10^{-1}$	-	$7.53 \cdot 10^{-1}$	-	$4.59 \cdot 10^{-1}$	-
50×50	$2.98 \cdot 10^{-2}$	2.73	$1.24 \cdot 10^{-1}$	2.60	$8.58 \cdot 10^{-2}$	2.42
100×100	$3.71 \cdot 10^{-3}$	3.01	$1.35 \cdot 10^{-2}$	3.21	$1.23 \cdot 10^{-2}$	2.80
200×200	$8.20 \cdot 10^{-4}$	2.18	$2.38 \cdot 10^{-3}$	2.50	$2.29 \cdot 10^{-3}$	2.43
400×400	$2.10 \cdot 10^{-4}$	1.97	$5.94 \cdot 10^{-4}$	2.00	$5.78 \cdot 10^{-4}$	1.99
(2,2,2,2) scheme						
25×25	$3.39 \cdot 10^{-1}$	-	$1.22 \cdot 10^{-0}$	-	$9.83 \cdot 10^{-1}$	-
50×50	$1.33 \cdot 10^{-1}$	1.36	$0.48 \cdot 10^{-1}$	1.34	$3.29 \cdot 10^{-1}$	1.58
100×100	$3.32 \cdot 10^{-2}$	1.99	$1.23 \cdot 10^{-1}$	1.96	$1.02 \cdot 10^{-1}$	1.69
200×200	$6.07 \cdot 10^{-3}$	2.45	$2.19 \cdot 10^{-2}$	2.49	$1.95 \cdot 10^{-2}$	2.38
400×400	$1.18 \cdot 10^{-3}$	2.35	$3.80 \cdot 10^{-3}$	2.53	$3.59 \cdot 10^{-3}$	2.44

Table 7.9: Comparison of the implemented (·,5,4,4) schemes and the (2,2,2,2) scheme against the reference solution on a 1600×1600 grid, on the smooth test problem (7.4) after $t = 1.0$. The errors were measured in the L^1 norm.

Once again, the convergence rates for the finest grid drops significantly for Q_1 in the L^1 norm, but are maintained in the L^∞ norm.

Figure 7.6 and 7.7 show the solution at $t = 1.0$ for Q_1 and Q_2. We also plot the errors, which simply is the difference $Q_{num} - Q_{ref}$ in each cell, i.e. positive error means that the method is over-estimating the solution, whereas negative error means that the method is under-estimating. Clearly the largest errors are situated at the rightmost wave front. In figure 7.8 and 7.9, we consider a cross section through these areas, where we compare the (3,5,4,4) scheme to

Errors at $t = 1.0$, problem (7.4)

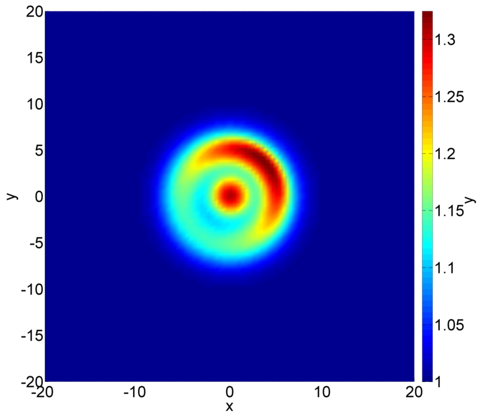
Grid size	Q_1	Q_2	Q_3
	(3,5,4,4) scheme		
50×50	$3.53 \cdot 10^{-2}$	$1.47 \cdot 10^{-1}$	$3.53 \cdot 10^{-2}$
100×100	$4.01 \cdot 10^{-3}$	$1.52 \cdot 10^{-2}$	$4.01 \cdot 10^{-3}$
150×150	$9.52 \cdot 10^{-4}$	$3.03 \cdot 10^{-3}$	$9.52 \cdot 10^{-4}$
300×300	$2.50 \cdot 10^{-4}$	$2.33 \cdot 10^{-4}$	$2.50 \cdot 10^{-4}$
	(2,2,2,2) scheme		
50×50	$1.33 \cdot 10^{-1}$	$4.84 \cdot 10^{-1}$	$3.29 \cdot 10^{-1}$
100×100	$3.32 \cdot 10^{-2}$	$1.23 \cdot 10^{-1}$	$1.02 \cdot 10^{-1}$
150×150	$1.24 \cdot 10^{-2}$	$4.64 \cdot 10^{-2}$	$3.96 \cdot 10^{-2}$
300×300	$2.31 \cdot 10^{-3}$	$7.51 \cdot 10^{-3}$	$7.26 \cdot 10^{-3}$

Table 7.10: Comparison of the implemented (3,5,4,4) scheme and the (2,2,2,2) scheme against the reference solution on a 1500×1500 grid, on the smooth test problem (7.4) after $t = 1.0$. The errors were measured in the L^1 norm.

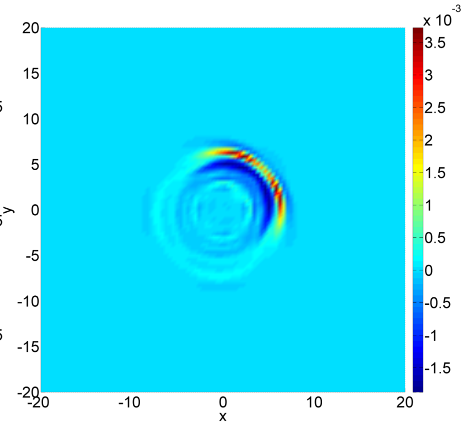
the (2,2,2,2) scheme and the fine grid reference solution.

Just as seen at $t = 0.1$, in most of the domain, the (2,2,2,2) scheme is accurate enough, except for around the function maximum, which is underestimated by the second-order scheme. To obtain about the same level of accuracy with the (2,2,2,2) scheme we need to increase the grid points in each direction by a factor somewhere between 2 and 3, as seen in Table 7.9 and 7.10. Hence, the (2,2,2,2) scheme does slightly better compared to the (3,5,4,4) scheme at $t = 1.0$, than it did at $t = 0.1$. Considering the runtimes in Table 7.1, we are able to speed up the computations by a factor somewhere between 1.3 and 3.5, by using the (2,5,4,4) scheme.

Note that the results using the (2,5,4,4) scheme, with $C_{clf} = 0.25$, are about the same as the ones by the (3,5,4,4) scheme. Even using the (3,5,4,4) scheme with $C_{cfl} = 0.25$ does not effect the solution noticeably.

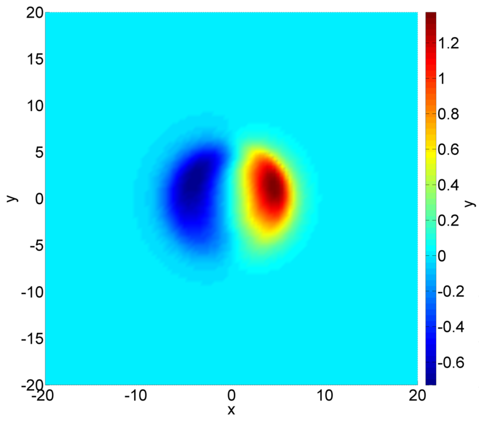


(a) WENO 100 solution, Q_1

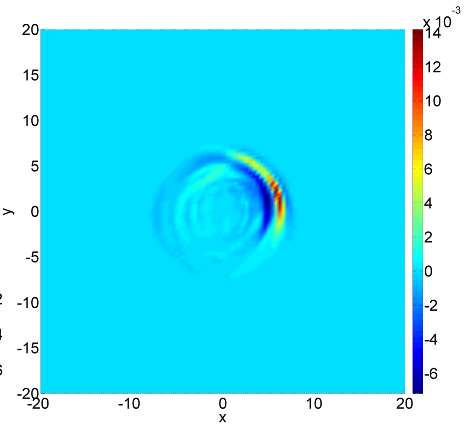


(b) WENO 100 error, Q_1

Figure 7.6: The solution for Q_1 on the problem (7.4) after $t = 1.0$, obtained by the (3,5,4,4) scheme on a 100 × 100 grid, and the error plot measured against the fine grid reference solution.

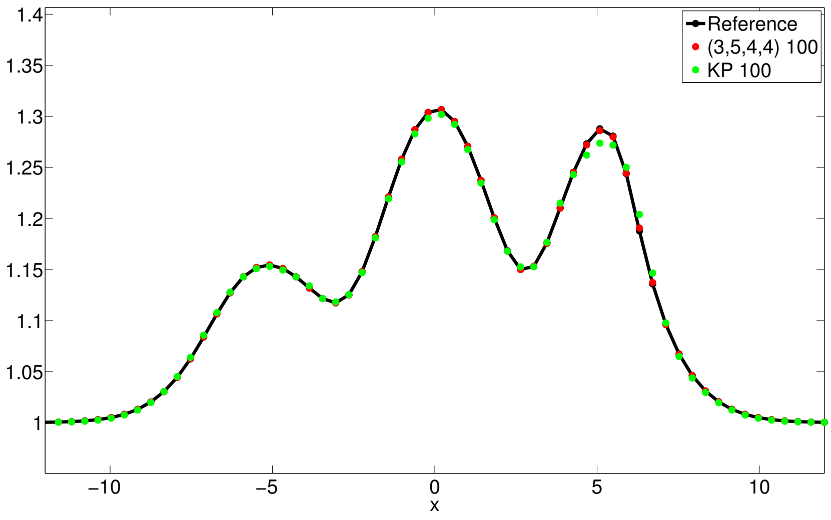


(a) WENO 100 solution, Q_2

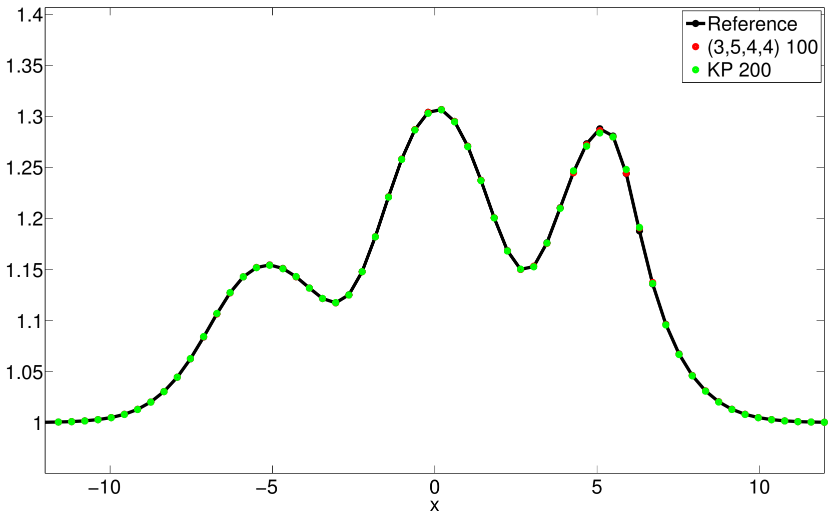


(b) WENO 100 error, Q_2

Figure 7.7: The solution for Q_2 on the problem (7.4) after $t = 1.0$, obtained by the (3,5,4,4) scheme on a 100 × 100 grid, and the error plot measured against the fine grid reference solution.

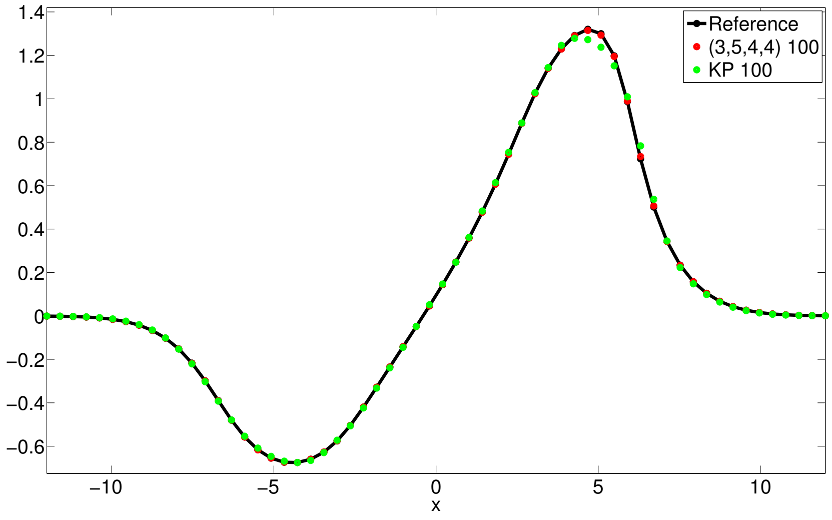


(a) WENO 100 vs KP 100, Q_1

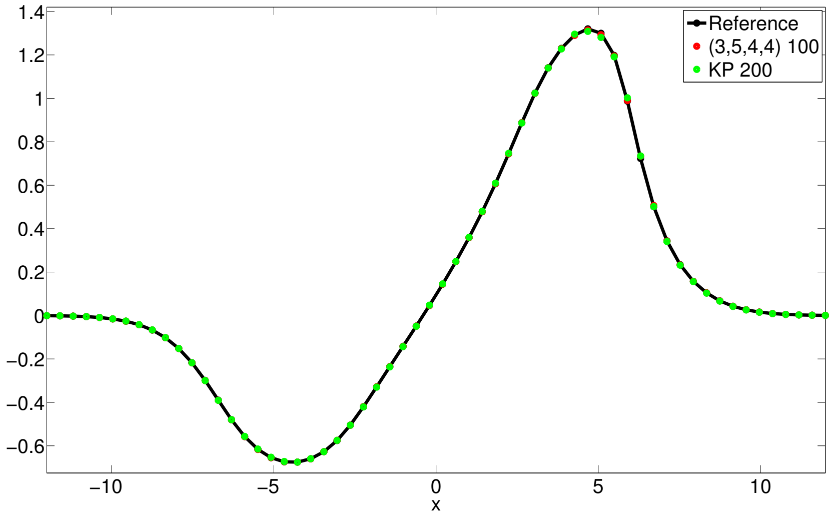


(b) WENO 100 vs KP 200, Q_1

Figure 7.8: Comparison of the (3,5,4,4) scheme on a 100×100 grid and the (2,2,2,2) scheme on 100×100 and 200×200 grids on problem (7.4) after $t = 1.0$, for the cross section $y \in [0.00, 0.04]$.



(a) WENO 100 vs KP 100, Q_2



(b) WENO 100 vs KP 200, Q_2

Figure 7.9: Comparison of the (3,5,4,4) scheme on a 100×100 grid and the (2,2,2,2) scheme on 100×100 and 200×200 grids on problem (7.4) after $t = 1.0$, for the cross section $y \in [1.96, 2.00]$.

7.4. Test of well-balanced property

In this section we aim to test the well-balanced property of the schemes. Test problems were taken from [18], but are commonplace in the literature, and also found in e.g. [12,16,30].

Stationary state

First we set up a lake at rest, with non-flat bottom topography, to test whether the scheme indeed is well-balanced. The problem is given by

$$\begin{aligned} B(x,y) &= 0.8e^{-50((x-0.5)^2+(y-0.5)^2)}, \\ Q(x,y,0) &= (1,\,0,\,0). \end{aligned} \tag{7.5}$$

on the domain $\Omega = [0,1]\times[0,1]$. Various boundary conditions could be used without effecting the problem in this case, since B goes to zero at the boundary. Here, wall conditions are used and we expect the water to remain at rest, i.e. $Q(x,y,t) = (1,0,0)$, for all t.

Stationary test $t = 0.1$, (2,5,4,4)th order scheme

Grid size	Norm	Q_1	Q_2	Q_3
50×50	L^1	$3.60\cdot 10^{-7}$	$1.12\cdot 10^{-5}$	$1.15\cdot 10^{-5}$
100×100	L^1	$4.80\cdot 10^{-7}$	$3.23\cdot 10^{-5}$	$2.41\cdot 10^{-5}$
200×200	L^1	$3.60\cdot 10^{-7}$	$7.77\cdot 10^{-5}$	$6.39\cdot 10^{-5}$
50×50	L^∞	$2.40\cdot 10^{-7}$	$8.50\cdot 10^{-7}$	$1.18\cdot 10^{-6}$
100×100	L^∞	$2.40\cdot 10^{-7}$	$1.23\cdot 10^{-6}$	$1.27\cdot 10^{-6}$
200×200	L^∞	$2.40\cdot 10^{-7}$	$1.61\cdot 10^{-6}$	$1.70\cdot 10^{-6}$

Table 7.11: Stationary state test. Here we use the L^∞ matrix norm defined by $\|A\|_{\max} = \max_{i,j}\{|A_{i,j}|\}$.

The L^1 norm and max cell error of the three components, using the (2,5,4,4) scheme on three different grid sizes at $t = 0.1$, are found in Table 7.11. For the water elevation, Q_1, we see that we get an error close to machine precision (keep in mind we are using single precision), and the small difference in L^1 error and the maximal cell error indicates that only a few cells are affected. For the two other components, however, the max errors are slightly bigger, and a vast number of cells have non-zero moments, shown by the relatively high

L^1 error. Although L^1 errors in the magnitude of 10^{-7} are reported in all components for single precision in [16, 30], we conclude that L^1 errors around 10^{-5} and L^∞ errors close to $1.0 \cdot 10^{-6}$ is sufficient to regard the scheme with the hydrostatic reconstruction as well-balanced. The (2,2,2,2) scheme is even more perfectly well-balanced in the Q_2 and Q_3 components, while the (2,5,4,2) scheme, without the hydrostatic reconstruction, achieve about the same results as the (2,5,4,4) scheme. The results also show that grid size does not appear to be of significance in the max norm.

As t increases, the errors accumulate, but not dramatically. At $t = 10.0$, for example, the errors in Table 7.11 are roughly ten times bigger, in both norms.

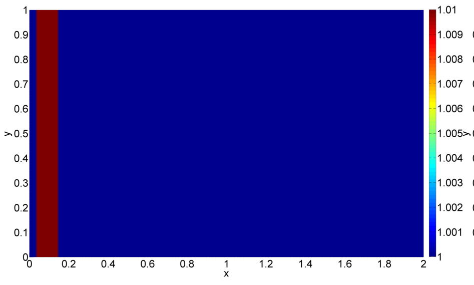
For the (3,5,4,4) scheme, using the third-order Runge-Kutta method, however, the stationary state test does not give satisfactory results. While the errors in Q_2 and Q_3 are only slightly bigger for the (3,5,4,4) scheme compared to the (2,5,4,4) scheme, the L^∞ errors in the Q_1 component are both bigger and, more disturbingly, the cell averages are systematically bigger than 1, causing huge L^1 errors. At $t = 10.0$ we have $Q_1 = 1.01$, which is 10^4 times as much as for the (2,5,4,4) scheme.

Stationary state with small perturbation

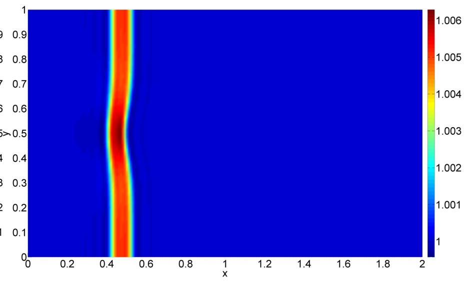
The next problem involves a slightly different bottom topography and a small perturbation of the lake-at-rest. It is given as

$$\begin{aligned} B(x, y) &= 0.8e^{-5(x-0.9)^2-50(y-0.5)^2}, \\ Q_1(x, y, 0) &= \begin{cases} 1.01, & x \in [0.05,\ 0.15] \\ 1.00, & \text{else}, \end{cases} \\ Q_2(x, y, 0) &= 0, \\ Q_3(x, y, 0) &= 0, \end{aligned} \tag{7.6}$$

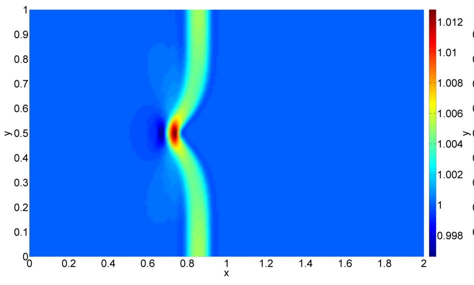
on the domain $\Omega = [0, 2] \times [0, 1]$, equipped with outflow boundaries. This initial perturbation causes waves to propagate to the left and right. The left-going wave quickly leaves the domain, and the right-going wave is deformed due to the bottom topography. The water falls back to rest when the wave fronts have passed. Snapshots of the right-going wave up to time $t = 0.6$ are shown in Figure 7.10. The results match the results obtained in [12,16,18,30], but due to the lack of a proper implementation of outflow boundary conditions, the domain was extended to $\Omega' = [-1, 2] \times [0.1]$, so that the left-going wave would not hit the left boundary right away. This would have caused a small, non-physical, reflection traveling to the right, chasing the initial right-going wave. Problem (7.6) is commonly solved on a 200×100 grid, thus, to get an equivalent setting, we solve it here on a 300×100 grid on the domain Ω'.



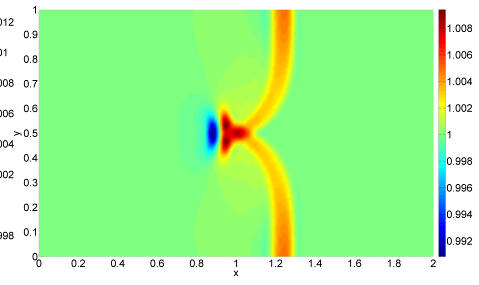
(a) $t = 0.00$



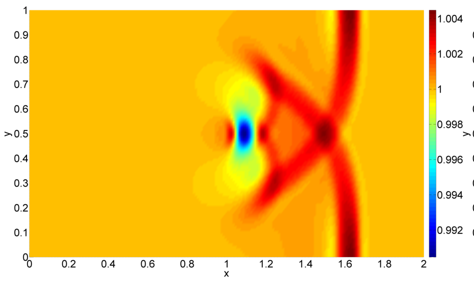
(b) $t = 0.12$



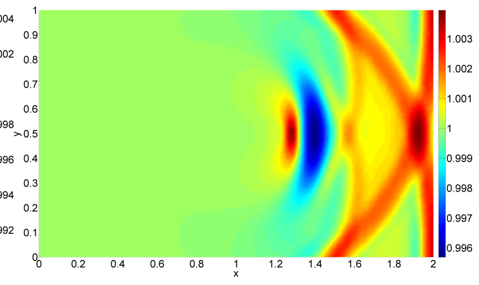
(c) $t = 0.24$



(d) $t = 0.36$



(e) $t = 0.48$



(f) $t = 0.60$

Figure 7.10: Snapshots of the Q_1 solution of the problem (7.6), using a 200×100 grid on Ω.

The small perturbation problem has non-smooth initial data, and even a small discontinuity in $Q_1(x, y, 0)$, but the bed slope source term, caused by the smooth bottom topography, is dominating the solution, and we are in fact clearly obtaining better results using the (2,5,4,4) scheme instead of the (2,2,2,2) scheme, as seen in Figures 7.11 and 7.12. Again we are observing that the peaks are approximated a lot better by the high-order scheme. The relative error of the KP scheme is huge on a 400×200 grid, and we need at least $800 \cdot 400$ cells to come close to the solution by the (2,5,4,4) scheme. There are, however, some spurious oscillations around $x = 1.2$, 1.5, 1.6, which are not appreciated. Here, we have not taken into account that a 400×200 grid for the (2,5,4,4) scheme, in fact is a 600×200 grid on the extended domain, Ω'. This naturally favours the KP scheme for time being. Implementing outflow boundary conditions would make speed-ups possible even on this problem, but probably not more than by 30-50 % due to the spurious oscillations.

Please note that the solutions obtained by the (3,5,4,4) scheme, match the (2,5,4,4) solutions around the wave fronts, while some disturbance is introduced in the part of the domain where $Q_1 = 1$, just as seen in the stationary state test.

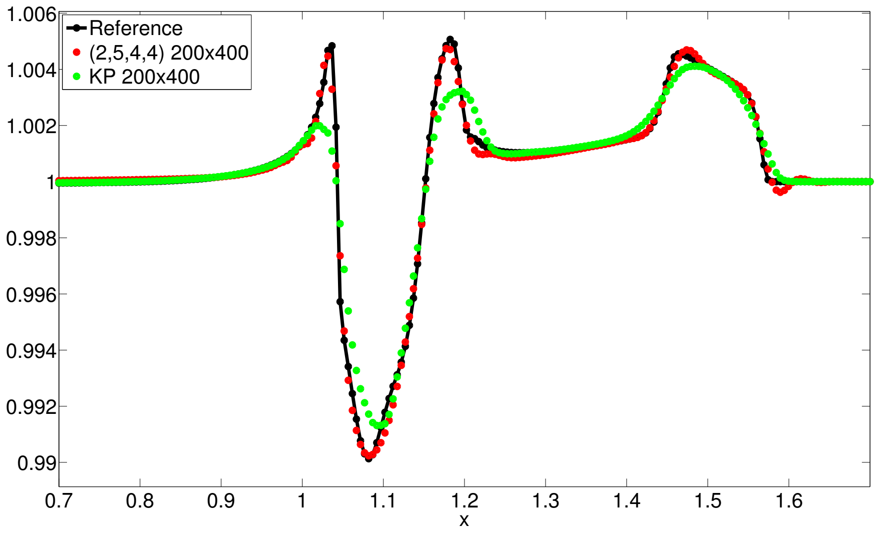


Figure 7.11: Comparison of the (2,5,4,4) and (2,2,2,2) scheme on the cross section at $y = [0.500, 0.505]$ of U_1 in problem (7.6). Here $t = 0.48$.

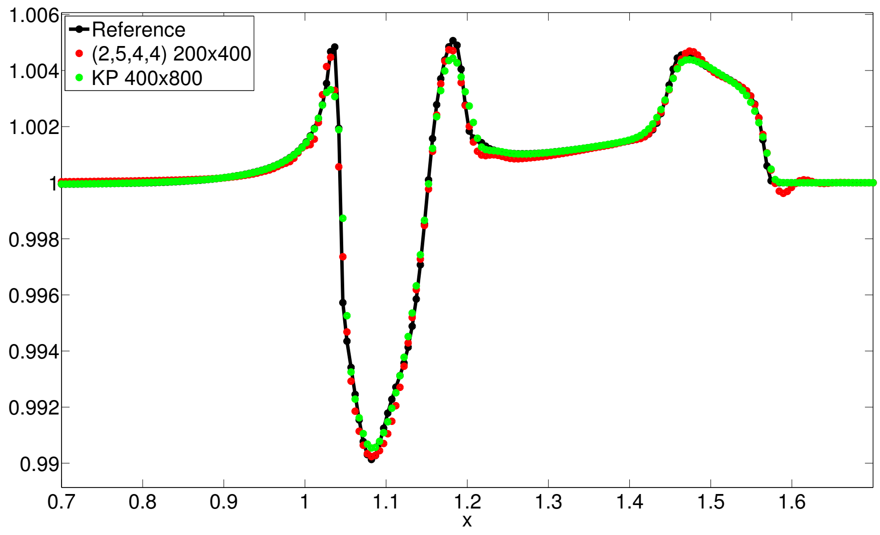


Figure 7.12: Comparison of the (2,5,4,4) and (2,2,2,2) scheme on the cross section at $y = [0.500, 0.505]$ of U_1 in problem (7.6). Here $t = 0.48$.

7.5. Unwanted oscillations

Consider the problem

$$\begin{aligned} Q_1(x,y,0) &= 1 + 2\,e^{\frac{-x^2-y^2}{10}} \\ Q_2(x,y,0) &= Q_3(x,y,0) = e^{\frac{-x^2-y^2}{4}} \\ B(x,y) &= 0.5\,e^{\frac{-x^2-y^2}{15}}, \end{aligned} \tag{7.7}$$

with wall boundary conditions, which is more or less the same as problem (7.4), but with the bottom topography multiplied by 0.5. This is done to avoid the presence of dry states. This problem exemplifies how hyperbolic problems might have discontinuous solutions, despite having smooth initial data.

Let us now evolve the solution up to $t = 4.0$, at which point the first water wavefront is close to hitting the walls. The solution of Q_1 is shown in Figure 7.13. Now, we have a smooth region to the left, in which the high-order

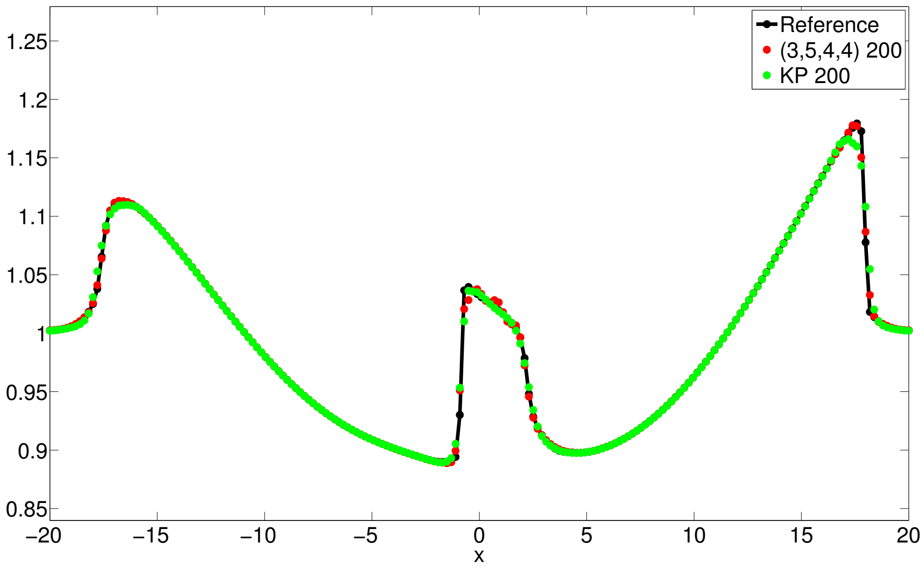
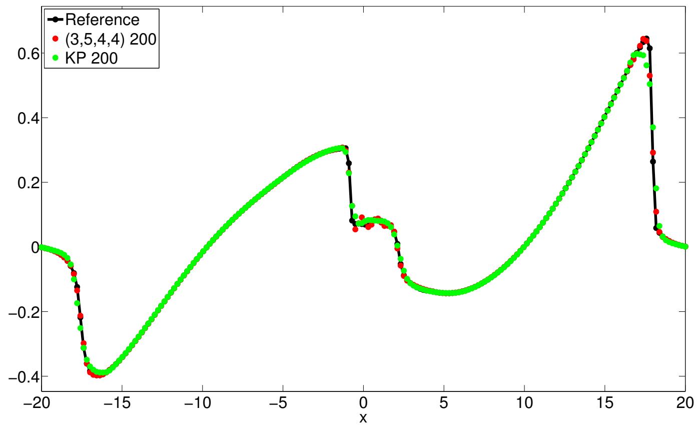
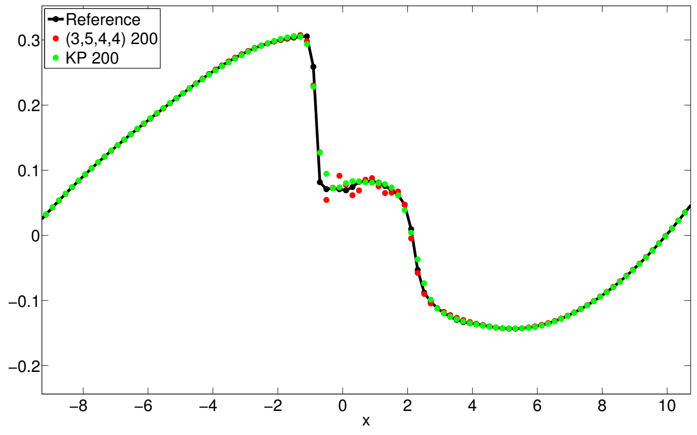


Figure 7.13: Comparison of the (3,5,4,4) and the KP scheme on a 200×200 grid (7.7) after $t = 4.0$, for Q_1 at the cross section $y \in [1.96, 2.00]$.

scheme is still approximating the local extrema slightly better, and two discontinuities, at approximately $x = 0$ and $x = 17$. Around the former discontinuity, the high-order schemes develop spurious oscillations. At this point, the choice of parameter ϵ, used to avoid division by zero in the non-linear WENO weights (3.24), turned out to have an impact on the solution. For example, the discontinuity at $x = 17$ caused oscillations in the solution for $\epsilon = 10^{-5}$, but for $\epsilon = 10^{-3}$, which was used in the shown plots, the rightmost discontinuity was actually approximated rather well compared to the (2,2,2,2) scheme. We were, however, not able to find a remedy for the oscillations around the origin. These oscillations are seen in all the high-order schemes, regardless of source term quadrature, Runge-Kutta method and reconstruction approach. Since the solution contains discontinuities, it comes as no surprise that the high-order schemes are not better at this point, but we would have wanted the high-order scheme to remain essentially non-oscillating. At earlier times, say $t = 2.0$, the high-order schemes performed as expected. The same phenomena occur in the other two variables, as seen for Q_2 in Figure 7.14. Again, for $\epsilon = 10^{-3}$, the smooth area and the rightmost discontinuity are captured better by the high-order scheme, while the oscillations around the origin, remain more or less unaffected by the choice of ϵ.



(a) Full domain



(b) Zoomed in

Figure 7.14: Comparison of the (3,5,4,4) and the KP scheme on a 200×200 grid (7.7) after $t = 4.0$, for Q_2 at the cross section $y \in [1.96, 2.00]$.

8. Concluding remarks

The main purpose of this work, was to develop high-order high-resolution methods for the shallow water equations on the GPU, and thereby investigate the possible gains in runtime compared to the already existing second-order scheme [7]. Two schemes were implemented, one using only a second-order quadrature for the bed slope source term and bilinear interpolation of the bottom topography, the other using a fourth-order source term quadrature, as well as a fifth-order WENO reconstruction of the water height. Both schemes use a fifth-order WENO reconstruction for the water elevation, fourth-order Gaussian quadratures for the one-sided flux integrals and the central-upwind flux for the interface fluxes. The schemes have been denoted $(\cdot, 5, 4, 2)$ and $(\cdot, 5, 4, 4)$ according to the order of the ODE solver, WENO reconstruction, one-sided flux integral and source term, respectively, with the 'dot', representing the ODE solver, indicating that several Runge-Kutta methods are supported.

The (·,5,4,2) scheme was implemented first, simply because it builds directly on the second-order scheme. It is, however, mainly suited for homogeneous problems, due to the low-order source term. For implementing the (·,5,4,4) scheme, more substantial changes of the second-order scheme were needed. This scheme can now be extended to WENO reconstructions and quadratures of even higher order without too much complications. Because the (·,5,4,2) should be regarded more as a step towards the (·,5,4,4) scheme, it lacks some features present in the (·,5,4,4) scheme, such as the removal of the shared memory array responsible for storing the source term. Due to this, the (·,5,4,4) can use slightly larger block sizes in the flux-source kernel.

The results from chapter 7 show that the (·,5,4,4) schemes has been successfully implemented and is of approximately order four in space, that it indeed is well-balanced and essentially non-oscillating around discontinuities, although occasionally some slightly disturbing oscillations were found, and the third-order Runge-Kutta method seems to be slightly increasing the water elevation in stationary state. The third-order Runge-Kutta method is, however, not causing any trouble in other test cases. It should also be mentioned that the (·,5,4,2) scheme, in these tests, is surprisingly good near dry states.

In our tests, no significant differences between the Runge-Kutta methods are seen with respect to accuracy, mostly because the second-order method performed surprisingly well. The third-order method is, however, slightly faster, since it allows for twice as long time steps. This result was slightly disappointing, but we have also seen that in these problems, complications such as dry states or discontinuities occur relatively early, possibly before the second-order Runge-Kutta has ruined the accuracy.

8.1. Hardware impact

Before commenting on the results, let us first consider the implementation and its effects on the GPU hardware. In the second-order scheme, the block sizes in the CUDA kernels were bounded entirely by the shared memory size and most of the GPU resources were spent on the flux-source kernel. The increased need for ghost cells in the WENO reconstruction obviously puts an even stricter memory bound to the high-order schemes, as well as severely increasing the amount of arithmetic in the flux-source kernel, making the maximum performance of the (·,5,4,4) scheme, in terms of computed cells per second, about 5-6 times slower. The performance ratio is, fortunately, about the same both for large and small grids.

On the GPU used for the final testing, the Quadro 5000 with 352 CUDA cores, no significant difference in runtime between the (·,5,4,2) and the (·,5,4,4) schemes was present, apart from that caused by the (·,5,4,4) scheme supporting larger blocks in the flux-source kernel.

8.2. Accuracy and speed-ups

As far as accuracy and runtime is concerned, we have seen that on smooth problems far away from dry states and after a low number of time steps, given a tolerance in the L^1 norm, the high-order schemes on n^2 grid cells will produce the same result as the second-order Kurganov-Petrova scheme on a grid consisting of somewhere between $(3n)^2$ and $(4n)^2$ cells. In light of the runtimes in Table 7.1, for low times, we are able to get a speed-up by a factor somewhere between 3 and 8, depending on the level of tolerance. These results are obtained both on homogeneous and non-homogeneous problems for the (·,5,4,4) scheme and on homogeneous problems for the (·,5,4,2) scheme.

After a larger number of time steps, we are not able to maintain the same speed-up. On smooth problems, the (2,2,2,2) scheme would typically need an increase in grid cells by a factor between 2 and 3, which in our experiments at best gives a speed-up by a factor 3.5, and a factor 1.3 in the worst case.

Based on the results, it is safe to say that for higher simulation times, the high-order scheme will at least not be inferior to the second-order scheme, provided the problem is smooth enough. By smooth enough, it is not necessarily essential that the initial data is smooth, as long as the dominant part of the problem is smooth, as seen in the small perturbation of stationary state (7.6), where the initial data is non-smooth and slightly discontinuous, but the bed slope term caused by the smooth bottom topography dominates the solution, and the fourth-order source term quadrature clearly produces results superior to the second-order scheme, at least on coarse grids.

We conclude that the crucial part for having success with high-order methods for the shallow water equations in two dimensions, is to maintain the high order of convergence for sufficiently long time. On the hardware used for testing, we need an order of convergence such that if the second-order scheme is run on a grid with $\Delta x = \Delta x_0$ and $\Delta y = \Delta y_0$, we are able to achieve the same accuracy with the high-order scheme on a grid with approximately $\Delta x > 2\Delta x_0$, $\Delta y > 2\Delta y_0$. The increased accuracy from using the high-order schemes, is mostly restricted to the areas around function extrema, as seen in the plots in Chapter 7. Hence, it would not make sense using a high-order scheme on problems in which we are not interested in a highly accurate approximations of the extreme values.

8.3. Further research

While the results using high-order schemes on smooth problems are promising, there are some issues that need to be addressed in order to make the high-order schemes a valid choice, possibly even on real-world problems.

In particular, boundary conditions is an important research field. As briefly mentioned, the naive implementation of outflow boundary conditions leads to a reflecting wave, although of small amplitude, potentially destroying the solution in the interior of the domain. By using exponential functions in our tests, we have avoided the potential problems at the boundaries. This, however, reduces our results to nothing more than a proof of concept. A good starting-point for further research would probably be [24], where both inflow and outflow boundary conditions are treated in a high-order setting, and their high-order scheme is applied to the problem of eddy formation in shelf slope jets along the Ormen Lange section of the Norwegian shelf. The runtime tests here indicates that the possible speed-ups on smooth problems would in fact be slightly bigger on large domains.

Also the ODE solver should be investigated more, as we experience a drop of convergence with time, convergence of order 3 leads only to small speed-ups. The development of fourth-order TVD Runge-Kutta methods is not as easy as for low-order methods. An alternative could be the SSP Runge-Kutta method [27], although the classical fourth-order Runge-Kutta has been used with success in [24] and [18]. As remarked by Titarev and Toro [28], the fourth-order SSP Runge-Kutta does not yield significantly better results than the third-order TVD Runge-Kutta, but requires more memory storage and the CFL coefficient should ideally not be increased enough to make the extra computations pay off with respect to runtime. If the aim is to get runtime speed-ups compared to the second-order scheme, a fourth-order Runge-Kutta method will have to be implemented with caution.

The fourth-order source term requires 8 more WENO reconstructions per cell, and 4 bilinear interpolations less of course, but it is still just as fast as the $(\cdot, 5, 4, 2)$ scheme, making it tempting to try a higher order quadrature both for the source term and the one-sided fluxes. However, this would put an even harder restriction to the block sizes in the flux-source kernel, seeing as we would need to store three line averages in each spatial direction per cell for using a three-point Gaussian quadrature, which with the current implementation would restrict the block size in the flux-source kernel to about 12×8. Increasing the order of the WENO reconstruction, would be even more costly, as we in a seventh-order WENO reconstruction would need 8 ghost cells in each direction, reducing the maxium block size to 12×9, with a two-point flux quadrature, and 8×8 in combination with a three-point quadrature. The ratio of ghost cells to interior cells in the y direction would go from $6/11$ to $8/8$, which is an increase of more than 80 %. The reason why the fourth-order source term is not effecting the runtime is because we can still get away with storing only two line averages in each direction per cell, meaning that adding a few more computations does not destroy the runtime, but running smaller blocks might do. Whether the accuracy of a scheme of even higher order would be sufficiently good to compensate for the smaller block sizes and more ghost cells, was not investigated, but it is perhaps not all that likely.

Bibliography

[1] Emmanuel Audusse, François Bouchut, Marie-Odile Bristeau, Rupert Klein, and Benoît Perthame. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. *SIAM J. Sci. Comput.*, 25(6):2050–2065, June 2004.

[2] Andreas Bollermann, Alexander Kurganov, and Sebastian Noelle. A well-balanced reconstruction for wetting/drying fronts. *Communications in Mathematical Sciences*, 2010.

[3] André R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 2010.

[4] André R. Brodtkorb, Trond R. Hagen, Knut-Andreas Lie, and Jostein R. Natvig. Simulation and visualization of the saint-venant system using gpus. *Comput. Visual. Sci. (Special issue on Hot Topics in Computational Engineering)*, 13, 2010.

[5] André R. Brodtkorb, Trond R. Hagen, and Martin L. Sætra. Gpu programming strategies and trends in gpu computing. *Journal of Parallel and Distributed Computing*, 2012.

[6] André R. Brodtkorb and Martin L. Sætra. Explicit shallow water simulations on gpus: Guidelines and best practices. 2012.

[7] André R. Brodtkorb, Martin L. Sætra, and Mustafa Altinakar. Efficient shallow water simulations on gpus: Implementation, visualization, verification, and validation. 2012.

[8] Sigal Gottlieb and Chi wang Shu. Total variation diminishing runge-kutta schemes. *Math. Comp*, 67:73–85, 1998.

[9] T. R. Hagen, M. O. Henriksen, J. M Hjelmervik, and K.-A. Lie. How to solve systems of conservation laws numerically using the graphics processor as a high-performance computational engine. In *Geometrical Modeling, Numerical Simulation, and Optimization: Industrial Mathematics at SINTEF.*

[10] Ami Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49(3):357 – 393, 1983.

[11] Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, 111. *J. Comput. Phys.*, 71(2):231–303, August 1987.

[12] Alexander Kurganov and Doron Levy. Central-upwind schemes for the saint-venant system. 2002.

[13] Alexander Kurganov, Sebastian Noelle, and Guergana Petrova. Semi-discrete central-upwind schemes for hyperbolic conservation laws and hamilton–jacobi equations. *SIAM J. Sci. Comput.*, 23(3):707–740, March 2001.

[14] Alexander Kurganov and Guergana Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant system. 2007.

[15] R. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-dependent Problems.* Society for Industrial and Applied Mathematics, 2007.

[16] Randall J. LeVeque. Balancing source terms and flux gradients in high-resolution godunov methods: The quasi-steady wave-propagation algorithm. *Journal of Computational Physics*, 146(1):346 – 365, 1998.

[17] R.J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[18] Sebastian Noelle, Normann Pankratz, Gabriella Puppo, and Jostein R. Natvig. Well-balanced finite volume schemes of arbitrary order of accuracy for shallow water flows. *Journal of Computational Physics*, 213(2):474 – 499, 2006.

[19] NVIDIA Corporation. *CUDA C Best Practices Guide*, 5.0 edition, 2012.

[20] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, 2012.

[21] NVIDIA Corporation, Paulius Micikevicius. *Analysis-Driven Optimization*, 2010.

[22] NVIDIA Corporation, Paulius Micikevicius. *Fundamental Optimizations*, 2010.

[23] NVIDIA Corporation, Victor Podlozhnyuk. *Image Convolution with CUDA*, 2007.

[24] Normann Pankratz, Jostein R. Natvig, Bjrn Gjevik, and Sebastian Noelle. High-order well-balanced finite-volume schemes for barotropic flows: Development and numerical comparisons. *Ocean Modelling*, 18(1):53 – 79, 2007.

[25] C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced numerical approximations of nonlinear hyperbolic equations.*

[26] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439 – 471, 1988.

[27] Raymond J. Spiteri and Steven J. Ruuth. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM Journal on Numerical Analysis*, 40(2):pp. 469–491, 2003.

[28] V.A. Titarev and E.F. Toro. Finite-volume weno schemes for three-dimensional conservation laws. 2003.

[29] E.F. Toro. *Shock-capturing methods for free-surface shallow flows.* John Wiley, 2001.

[30] Yulong Xing and Chi-Wang Shu. High order finite difference weno schemes with the exact conservation property for the shallow water equations. *J. Comput. Phys.*, 208(1):206–227, September 2005.