Editorial Manager(tm) for Journal of Scientific Computing
Manuscript Draft

Manuscript Number: JOMP497

Title: Two-dimensional compact third-order polynomial reconstructions. Solving nonconservative hyperbolic systems using GPUs.

Article Type: SI: Systems with Source Terms

Keywords: Finite volume methods; shallow water; nonconservative hyperbolic systems; high-order schemes; well-balanced; GPUs.

Corresponding Author: Dr. Jose M. Gallardo,

Corresponding Author's Institution: University of Malaga

First Author: Jose M. Gallardo

Order of Authors: Jose M. Gallardo; Sergio Ortega; Marc de la Asuncion; Jose M. Mantas

Abstract: We present a new kind of high-order reconstruction operator of polynomial type, which is used in combination with a high-order finite volume scheme for solving nonconservative hyperbolic systems. The implementation of the scheme is carried out on Graphics Processing Units (GPUs), thus achieving a substantial improvement of the speedup with respect to normal CPUs. As an application, the two-dimensional shallow water equations with geometrical source term due to the bottom slope is considered.

# Two-dimensional compact third-order polynomial reconstructions. Solving nonconservative hyperbolic systems using GPUs ⋆

José M. Gallardo, Sergio Ortega, Marc de la Asunción [a], José Miguel Mantas [b]

[a] *Department of Mathematical Analysis, University of Málaga, Spain*

[b] *Software Engineering Department, University of Granada, Spain*

**Abstract**

We present a new kind of high-order reconstruction operator of polynomial type, which is used in combination with the scheme presented in [2] for solving nonconservative hyperbolic systems. The implementation of the scheme is carried out on Graphics Processing Units (GPUs), thus achieving a substantial improvement of the speedup with respect to normal CPUs. As an application, the two-dimensional shallow water equations with geometrical source term due to the bottom slope is considered.

*Key words:* Finite volume methods, shallow water, nonconservative hyperbolic systems, high-order schemes, well-balanced, GPUs.
AMS (MOS): 65N06, 76B15, 76M20, 76N99.

⋆ This research has been partially supported by the Spanish Government Research projects MTM06-08075, P06-RNM-01594 and MTM09-11923. The numerical computations have been performed at the Laboratory of Numerical Methods of the University of Málaga.

*Email addresses:* {gallardo,sergio}@anamat.cie.uma.es, marc@correo.ugr.es (José M. Gallardo, Sergio Ortega, Marc de la Asunción), jmmantas@ugr.es (José Miguel Mantas).

## 1 Introduction

The motivation of this paper is the implementation, in a efficient way, of a class of high-order well-balanced numerical schemes for two-dimensional non-conservative hyperbolic systems ([2]):

$$W_t + \mathcal{A}_1(W)W_x + \mathcal{A}_2(W)W_y = 0 \tag{1}$$

with $\mathbf{x} = (x, y) \in D \subset \mathbb{R}^2$ and $t \in (0, T)$, where $W(\mathbf{x}, t)$ takes values on a convex domain $\Omega$ of $\mathbb{R}^N$ and $\mathcal{A}_i$, $i = 1, 2$, are smooth and locally bounded matrix-valued functions from $\Omega$ to $\mathcal{M}_N(\mathbb{R})$.

A fundamental point in the construction of such kind of high-order schemes is the use of appropriate reconstructions operators ([3,7,9,11,19]). These operators provide highly accurate approximations to the point values of the reconstructed variables, using the cell values on some given stencil. We present here a new kind of fully two-dimensional reconstruction operator, which is based on nonlinear weighted combinations of appropriate polynomials. This operator enjoys, among others, the desiderable property of having a compact stencil, which results in a highly stable numerical scheme.

The shallow water equations that govern the flow of one layer or two superposed layers of immiscible homogeneous fluids can be written in the form (1). Systems with similar characteristics also appear in other fluid models, as is the case of two-phase flows. In this work we focus on the one-layer shallow water implementation. The numerical solution of this model is useful for several applications related to geophysical flows: simulation of rivers, channels, dam-break problems, etc. These simulations require a great demand of computing power due to the dimensions of the domain, both in space and time. As a consequence, extremely efficient high-performance solvers are required to solve and analyze these problems in reasonable execution times.

Currently, a cost-effective emerging architecture exists which is specially indicated to considerably accelerate computational intensive tasks, like the one considered in this paper. Modern Graphics Processing Units (GPUs) are not only used to render 3D graphics, but can also be a cost-effective way to speedup the numerical solution of several mathematical models in Science and Engineering (see [17,18] for a revision on the topic). Modern GPUs offer over 100-200 processing units optimized for performing massively floating-point operations in parallel [14]. As a consequence, for several algorithmic structures, these architectures are able to obtain a substantially higher performance than a powerful CPU.

In [4], an explicit central-upwind scheme is implemented on a NVIDIA GeForce 7800 GTX card to simulate the one-layer shallow water system, achieving a

speedup from 15 to 30 with respect to an implementation on an Intel Xeon processor. In [8,10], a first-order path-conservative ([16]) Roe type solver has been implemented on several NVIDIA GeForce cards to simulate the one-layer shallow water system. For medium-size problems, a speedup of two orders of magnitude faster than a SSE-optimized CPU version of the solver is achieved.

In the present work, we follow the strategy described in [10] for designing an efficient implementation of the numerical scheme presented in [2] using CUDA.

The structure of the paper is as follows. In Section 2, the high-order finite volume scheme developed in [2] is reviewed. The new third-order reconstruction operator is presented in Section 3. Next, Section 4 is dedicated to explain the details of the implementation of the numerical scheme in GPUs, for the particular case of one-layer shallow water equations with bottom topography. Moreover, several numerical experiments are presented. Finally, some conclusions are drawn in Section 5.

## 2 High-order finite volume schemes

### *2.1 Roe methods*

Consider the two-dimensional nonconservative hyperbolic system (1), that it is assumed to be strictly hyperbolic, i.e., for each $W \in \Omega$ and $\boldsymbol{\eta} = (\eta_1, \eta_2) \in \mathbb{R}^2$, the matrix

$$\mathcal{A}(W, \boldsymbol{\eta}) = \mathcal{A}_1(W)\eta_1 + \mathcal{A}_2(W)\eta_2$$

has $N$ real and distinct eigenvalues

$$\lambda_1(W, \boldsymbol{\eta}) < \cdots < \lambda_N(W, \boldsymbol{\eta}).$$

Thus, $\mathcal{A}(W, \boldsymbol{\eta})$ is diagonalizable; it can be decomposed as

$$\mathcal{A}(W, \boldsymbol{\eta}) = \mathcal{K}(W, \boldsymbol{\eta}) \cdot \Lambda(W, \boldsymbol{\eta}) \cdot \mathcal{K}(W, \boldsymbol{\eta})^{-1}$$

where $\Lambda(W, \boldsymbol{\eta})$ is the diagonal matrix whose coefficients are the eigenvalues of $\mathcal{A}(W, \boldsymbol{\eta})$ and $\mathcal{K}(W, \boldsymbol{\eta})$ is a matrix whose $j$-th column is an eigenvector $R_j(W, \boldsymbol{\eta})$ associated to the eigenvalue $\lambda_j(W, \boldsymbol{\eta})$.

To discretize (1) the computational domain $D$ is decomposed into subsets with a simple geometry, called cells or finite volumes: $V_i \subset \mathbb{R}^2$. It is assumed that the cells are closed convex polygons whose intersections are either empty, a complete edge or a vertex. Denote by $\mathcal{T}$ the mesh, i.e., the set of cells, and by $NV$ the number of cells.

Given a finite volume $V_i$, $|V_i|$ will represent its area; $N_i \in \mathbb{R}^2$ its center; $\mathcal{N}_i$ the set of indexes $j$ such that $V_j$ is a neighbor of $V_i$; $E_{ij}$ the common edge of two neighboring cells $V_i$ and $V_j$, and $|E_{ij}|$ its length; $d_{ij}$ the distance from $N_i$ to $E_{ij}$; $\boldsymbol{\eta}_{ij} = (\eta_{ij,1}, \eta_{ij,2})$ the normal unit vector at the edge $E_{ij}$ pointing towards the cell $V_j$ (see figure 1); $\Delta$ the maximum of the diameters of the cells; $W_i^n$ the constant approximation to the average of the solution in the cell $V_i$ at time $t^n$ provided by the numerical scheme:

$$W_i^n \cong \frac{1}{|V_i|} \int_{V_i} W(\mathbf{x}, t^n)\, d\mathbf{x}.$$

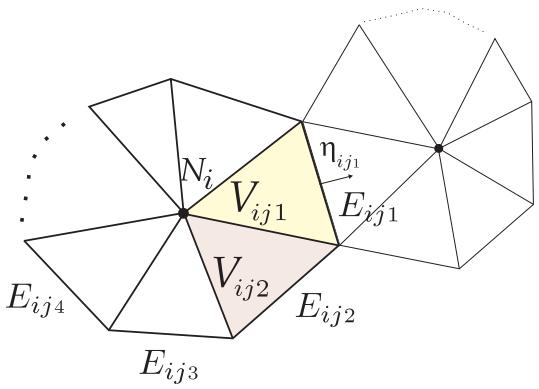


Fig. 1. Finite volume discretization.

A *family of paths* in $\Omega \subset \mathbb{R}^N$ is defined as a locally Lipschitz map

$$\Psi\colon [0,1] \times \Omega \times \Omega \times S^1 \to \Omega$$

such that for any $W_L, W_R \in \Omega$, $\boldsymbol{\eta} \in S^1$ (the unit sphere in $\mathbb{R}^2$) and $s \in [0,1]$, the following properties hold ([2]):

(1) $\Psi(0; W_L, W_R, \boldsymbol{\eta}) = W_L$ and $\Psi(1; W_L, W_R, \boldsymbol{\eta}) = W_R$.
(2) $\Psi(s; W_L, W_R, \boldsymbol{\eta}) = \Psi(1-s; W_R, W_L, -\boldsymbol{\eta})$.
(3) $\Psi(s, W, W, \boldsymbol{\eta}) = W$.

Given a family of paths $\Psi$, a *Roe linearization* of system (1) is a function

$$\mathcal{A}_\Psi\colon \Omega \times \Omega \times S^1 \to \mathcal{M}_N(\mathbb{R})$$

satisfying the following properties for each $W_L, W_R \in \Omega$ and $\boldsymbol{\eta} \in S^1$:

(1) $\mathcal{A}_\Psi(W_L, W_R, \boldsymbol{\eta})$ has $N$ distinct real eigenvalues

$$\lambda_1(W_L, W_R, \boldsymbol{\eta}) < \lambda_2(W_L, W_R, \boldsymbol{\eta}) < \cdots < \lambda_N(W_L, W_R, \boldsymbol{\eta}).$$

(2) $\mathcal{A}_\Psi(W, W, \boldsymbol{\eta}) = \mathcal{A}(W, \boldsymbol{\eta})$.
(3) $\mathcal{A}_\Psi(W_L, W_R, \boldsymbol{\eta}) \cdot (W_R - W_L) =$

$$\int_0^1 \mathcal{A}(\Psi(s; W_L, W_R, \boldsymbol{\eta}), \boldsymbol{\eta}) \frac{\partial \Psi}{\partial s}(s; W_L, W_R, \boldsymbol{\eta})\, ds. \tag{2}$$

Denote by $\Lambda_\Psi(W_L, W_R, \boldsymbol{\eta})$ the diagonal matrix whose coefficients are the eigenvalues $\lambda_j(W_L, W_R, \boldsymbol{\eta})$ and let $\mathcal{K}_\Psi(W_L, W_R, \boldsymbol{\eta})$ be the associated eigenvectors matrix. Define the positive and negative parts of $\mathcal{A}_\Psi(W_L, W_R, \boldsymbol{\eta})$ as

$$\mathcal{A}^{\pm}_\Psi(W_L, W_R, \boldsymbol{\eta}) = \mathcal{K}_\Psi(W_L, W_R, \boldsymbol{\eta}) \cdot \Lambda^{\pm}_\Psi(W_L, W_R, \boldsymbol{\eta}) \cdot \mathcal{K}_\Psi(W_L, W_R, \boldsymbol{\eta})^{-1},$$

where $\Lambda^{+}_\Psi(W_L, W_R, \boldsymbol{\eta})$ (respectively, $\Lambda^{-}_\Psi(W_L, W_R, \boldsymbol{\eta})$) is the diagonal matrix whose coefficients are the positive (respectively, negative) parts of the eigenvalues $\lambda_j(W_L, W_R, \boldsymbol{\eta})$.

In the particular case in which $\mathcal{A}_k(W)$, $k = 1, 2$, is the Jacobian matrix of a smooth flux function $F_k(W)$, property (2) does not depend on the family of paths and reduces to the usual Roe property:

$$\mathcal{A}_\Psi(W_L, W_R, \boldsymbol{\eta}) \cdot (W_R - W_L) = F_{\boldsymbol{\eta}}(W_R) - F_{\boldsymbol{\eta}}(W_L) \tag{3}$$

for any $\boldsymbol{\eta} \in S^1$, where

$$F_{\boldsymbol{\eta}}(W) = \eta_1 F_1(W) + \eta_2 F_2(W) \tag{4}$$

represents the flux along the $\boldsymbol{\eta}$ direction.

The general expression of a Roe scheme in upwind form for solving (1) is given by ([2]):

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \mathcal{N}_i} |E_{ij}| \mathcal{A}^-_{ij} \cdot (W_j^n - W_i^n) \tag{5}$$

where

$$\mathcal{A}^-_{ij} = \mathcal{A}^-_\Psi(W_i^n, W_j^n, \boldsymbol{\eta}_{ij}).$$

Additionally, a CFL condition must be imposed to ensure stability:

$$\Delta t \cdot \max \left\{ \frac{|\lambda_{ij,k}|}{d_{ij}};\ i = 1, \ldots, NV,\ j \in \mathcal{N}_i,\ \ k = 1, \ldots, N \right\} = \delta, \tag{6}$$

with $0 < \delta \leq 1$.

As in the case of systems of conservation laws, when sonic rarefaction waves appear it is necessary to modify the numerical scheme in order to obtain entropy-satisfying solutions. For instance, the Harten-Hyman entropy fix technique ([5]) can be easily adapted here.

Some general results concerning the consistency and well-balanced properties of Roe schemes have been studied in [2].

### 2.2 High-order extension

This section is devoted to the development of a high-order extension of scheme (5).

To begin with, consider a *reconstruction operator*, i.e., an operator that associates to a given family $\{W_i\}_{i=1}^{NV}$ of cell values two families of functions defined at the edges:

$$\gamma \in E_{ij} \mapsto W_{ij}^{\pm}(\gamma)$$

in such a way that, whenever

$$W_i = \frac{1}{|V_i|} \int_{V_i} W(\mathbf{x})\, d\mathbf{x} \tag{7}$$

for some smooth function $W$, then

$$W_{ij}^{\pm}(\gamma) = W(\gamma) + \mathcal{O}(\Delta^p), \quad \gamma \in E_{ij}.$$

It will be assumed that the reconstructions are calculated in the following way: given the family $\{W_i\}_{i=1}^{NV}$ of cell values, an approximation function is constructed at every cell $V_i$, based on the values $W_j$ at some stencil of neighboring cells to $V_i$:

$$P_i(\mathbf{x}) = P_i\left(\mathbf{x};\ \{W_j\}_{j\in\mathcal{B}_i}\right)$$

for some set of indexes $\mathcal{B}_i$. These approximation functions are usually constructed by means of interpolation or approximation methods. The reconstructions at $\gamma \in E_{ij}$ are defined as

$$W_{ij}^{-}(\gamma) = \lim_{\mathbf{x}\to\gamma} P_i(\mathbf{x}), \quad W_{ij}^{+}(\gamma) = \lim_{\mathbf{x}\to\gamma} P_j(\mathbf{x}). \tag{8}$$

Clearly, for any $\gamma \in E_{ij}$ the following equalities are satisfied:

$$W_{ij}^{-}(\gamma) = W_{ji}^{+}(\gamma), \quad W_{ij}^{+}(\gamma) = W_{ji}^{-}(\gamma).$$

The reconstruction operator is assumed to satisfy the following properties:

(H1) It is conservative, i.e., the following equality holds for any cell $V_i$:

$$W_i = \frac{1}{|V_i|} \int_{V_i} P_i(\mathbf{x}) d\mathbf{x}. \tag{9}$$

(H2) It is of order $p$, in the sense that

$$W(\gamma) - W_{ij}^{\pm}(\gamma) = \Delta^p g_{ij}(\gamma) + \mathcal{O}(\Delta^{p+1}), \quad \gamma \in E_{ij}$$

being $g_{ij}$ a regular function.

(H3) It is of order $q$ in the interior of the cells, i.e., if the operator is applied to a sequence $\{W_i\}$ satisfying (7) for some smooth function $W(\mathbf{x})$, then

$$P_i(\mathbf{x}) = W(\mathbf{x}) + \mathcal{O}(\Delta^q),\ \mathbf{x} \in \mathrm{int}(V_i). \tag{10}$$

(H4) The gradient of $P_i$ provides an approximation of order $m$ to the gradient of $W$:

$$\nabla P_i(\mathbf{x}) = \nabla W(\mathbf{x}) + \mathcal{O}(\Delta^m),\ \mathbf{x} \in \mathrm{int}(V_i). \tag{11}$$

The semidiscrete expression of the high-order extension of scheme (5), based on a given reconstruction operator, is the following ([2]):

$$\begin{aligned} W_i'(t) = -\frac{1}{|V_i|}\Bigg[ & \sum_{j\in\mathcal{N}_i} \int_{E_{ij}} \mathcal{A}_{ij}^-(\gamma,t)\Big(W_{ij}^+(\gamma,t) - W_{ij}^-(\gamma,t)\Big)d\gamma \\ & + \int_{V_i} \left(\mathcal{A}_1(P_i^t(\mathbf{x}))\frac{\partial P_i^t}{\partial x}(\mathbf{x}\big) + \mathcal{A}_2(P_i^t(\mathbf{x}))\frac{\partial P_i^t}{\partial y}(\mathbf{x})\right)d\mathbf{x}\Bigg] \end{aligned} \tag{12}$$

where $P_i^t$ is the approximation function at time $t$:

$$P_i^t(\mathbf{x}) = P_i\left(\mathbf{x};\ \{W_j(t)\}_{j\in\mathcal{B}_i}\right),$$

$W_{ij}^{\pm}(\gamma,t)$ are given by

$$W_{ij}^-(\gamma,t) = \lim_{\mathbf{x}\to\gamma} P_i^t(\mathbf{x}), \quad W_{ij}^+(\gamma,t) = \lim_{\mathbf{x}\to\gamma} P_j^t(\mathbf{x}), \tag{13}$$

and

$$\mathcal{A}_{ij}(\gamma,t) = \mathcal{A}_\Psi\left(W_{ij}^-(\gamma,t), W_{ij}^+(\gamma,t), \boldsymbol{\eta}_{ij}\right).$$

The following result can be proved (see [2]):

**Theorem 2.1** Assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are of class $\mathcal{C}^2$ with bounded derivatives and $\mathcal{A}_\Psi$ is bounded. Suppose also that the reconstruction operator satisfies hypotheses (H1)–(H4). Then (12) is an approximation of order at least $\alpha = \min(p, q, m)$.

**Remark 1** The conclusion of theorem 2.1 is rather pessimistic: the observed order in experiments is usually $\alpha = \min(p, q, m+1)$; see [2] for more details.

In practice, the integral terms in (12) must be approximated numerically. A one-dimensional quadrature formula of order $\bar{r}$ is applied to calculate the line integrals:

$$\int_a^b f(s)ds = (b-a)\left(\sum_{l=1}^{n(\bar{r})} \omega_l f(x_l)\right) + \mathcal{O}(\Delta^{\bar{r}}), \tag{14}$$

while a two-dimensional quadrature formula of order $\bar{s}$ is used to compute the volume integrals:

$$\int_{V_i} f(\mathbf{x})\, d\mathbf{x} = |V_i| \sum_{l=1}^{n(\bar{s})} \alpha_l f(\mathbf{x}_l^i) + \mathcal{O}(|V_i|^{\bar{s}}). \tag{15}$$

To preserve the order of the numerical scheme, it is necessary to have $\bar{r} \geq \alpha$ and $\bar{s} \geq \alpha$.

Finally, the numerical scheme is written as follows:

$$\begin{aligned} W_i'(t) = -\frac{1}{|V_i|}\Bigg[ & \sum_{j \in \mathcal{N}_i} |E_{ij}| \sum_{l=1}^{n(\bar{r})} w_l \mathcal{A}_{ij,l}^{-}(t)\Big(W_{ij,l}^{+}(t) - W_{ij,l}^{-}(t)\Big) \\ & + |V_i| \sum_{l=1}^{n(\bar{s})} \alpha_l \left( \mathcal{A}_1(P_i^t(\mathbf{x}_l^i)) \frac{\partial P_i^t}{\partial x}(\mathbf{x}_l^i) + \mathcal{A}_2(P_i^t(\mathbf{x}_l^i)) \frac{\partial P_i^t}{\partial y}(\mathbf{x}_l^i) \right) \Bigg] \end{aligned} \tag{16}$$

where

$$W_{ij,l}^{\pm}(t) = W_{ij}^{\pm}(a_{ij} + s_l(b_{ij} - a_{ij}), t)$$

and

$$\mathcal{A}_{ij,l}(t) = \mathcal{A}_{\Psi}(W_{ij,l}^{-}(t), W_{ij,l}^{+}(t), \boldsymbol{\eta}_{ij}).$$

**Remark 2** A technique that avoids the explicit computation of $\nabla P_i(\mathbf{x})$ has been introduced in [13] in the one-dimensional case. The use of this technique, that is based on the trapezoidal rule and Romberg extrapolation, makes the expected order of accuracy to be $\min(p, q)$. The extension to two-dimensional problems is straightforward for structured meshes, while for unstructured meshes a Romberg extrapolation formula for triangles can be used (see [22]).

For time stepping, high-order TVD Runge-Kutta methods like those described in [21] are applied. In particular, in this work we use a third-order reconstruction operator in space and a third-order TVD Runge-Kutta method to advance in time.

The well-balancedness properties of schemes (12) and (16) have been analyzed in [2].

## 3 Third-order reconstruction operator

The definition of reconstruction operators in two-dimensional domains is, in general, a difficult task. For structured rectangular meshes, the problem can be reduced to standard one-dimensional reconstructions in the coordinate directions, such as ENO ([20]), WENO ([7], [9]), PHM ([11]), etc. However,

when unstructured meshes are considered the construction of such operators becomes more difficult, and it may require a higher computational cost (see [3,6] for example).

In this section we present a new kind of reconstruction operator of polynomial type, that is third-order accurate on each computational cell. We will focus on the simpler case of uniform rectangular meshes; however, the method can be extended to general nonuniform quadrilateral meshes following similar guidelines. On the other hand, it is worth noting that uniform rectangular meshes are specially well suited for the implementation of the method on GPUs, that is the ultimate goal of the present work (see Section 4.1).

Assume that the computational domain is composed by rectangular uniform cells of size $\Delta x \times \Delta y$. Given a cell $V$, consider the nine-point stencil of neighboring cells $\{V_i;\, i = 0, \ldots, 8\}$ with $V_4 = V$, and the middle points $\mathbf{p}_N$, $\mathbf{p}_S$, $\mathbf{p}_E$ and $\mathbf{p}_W$ of each edge of $V_4$: see figure 2.

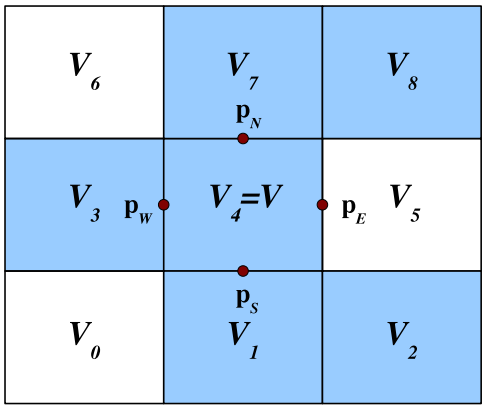


Fig. 2. Stencil used to approximate the derivatives of $F(\mathbf{x})$ at $\mathbf{p}_E$.

Let $F(\mathbf{x})$ be a sufficiently smooth function to be reconstructed on $V$, and denote $\mathbf{x} = (x, y)$. For each $\mathbf{p}_R = (x_R, y_R)$, $R \in \{N, S, E, W\}$, consider a second-order Taylor expansion of $F(\mathbf{x})$ around $\mathbf{p}_R$:

$$\begin{aligned} F(\mathbf{x}) =& F(\mathbf{p}_R) + F_x(\mathbf{p}_R)(x - x_R) + F_y(\mathbf{p}_R)(y - y_R) \\ &+ \frac{1}{2} F_{xx}(\mathbf{p}_R)(x - x_R)^2 + F_{xy}(\mathbf{p}_R)(x - x_R)(y - y_R) \\ &+ \frac{1}{2} F_{yy}(\mathbf{p}_R)(y - y_R)^2 + \mathcal{O}(\Delta^3), \end{aligned}$$

where $\Delta$ is the diameter of $V$. To approximate the derivatives of $F(\mathbf{x})$ at $\mathbf{p}_R$ we will apply the procedure presented in [19], that is briefly explained below.

Integrating the above expression in $V_i$ and dividing it by the cell area $|V_i|$, we

obtain an expression for the average $\overline{F}_i$:

$$\begin{aligned}\overline{F}_i =& F(\mathbf{p}_R) + \alpha_i F_x(\mathbf{p}_R) + \beta_i F_y(\mathbf{p}_R) + \gamma_i \frac{1}{2} F_{xx}(\mathbf{p}_R) \\ &+ \delta_i F_{xy}(\mathbf{p}_R) + \varepsilon_i \frac{1}{2} F_{yy}(\mathbf{p}_R) + \mathcal{O}(\Delta^3).\end{aligned}$$

For unstructured meshes, the coefficients $\alpha_i$, $\beta_i$, $\gamma_i$, $\delta_i$ and $\varepsilon_i$ only depend on the geometry of the mesh, so they can be computed and stored in a preprocessing step.

Next, the derivatives of $F(\mathbf{x})$ at each $\mathbf{p}_R$ can be approximated by appropriate linear combinations of the averages $\overline{F}_i$. To fix ideas, we will focus on the point $\mathbf{p}_E$. To approximate $F_{xx}(\mathbf{p}_E)$, an equality of the form

$$\sum_j \mu_j \overline{F}_j = F_{xx}(\mathbf{p}_E) + \sum_j \mu_j \mathcal{O}(\Delta^3)$$

is considered, where $j$ is a local index to be specified later. This equality is achieved if the condition

$$\mathbf{A}\boldsymbol{\mu} = \mathbf{b} \tag{17}$$

holds, where the $j$-th column of $\mathbf{A} \in \mathcal{M}_6(\mathbb{R})$ is $(1, \alpha_j, \beta_j, \gamma_j, \delta_j, \varepsilon_j)^T$ and $\mathbf{b} = (0, 0, 0, 2, 0, 0)^T$.

To fulfill (17) a six-point stencil is thus required. As pointed out in [19], the more natural stencil $\{V_j; j = 1, 2, 4, 5, 7, 8\}$ formed by the six nearest neighbouring cells to $\mathbf{p}_E$ is not adequate, as for uniform meshes the system (17) is under-determined. This problem can be fixed by moving one of the cells; in this case, if the stencil $\{V_j; j = 1, 2, 3, 4, 7, 8\}$ is chosen (see figure 2) then (17) has an unique solution, that leads to

$$\tilde{F}_{xx}(\mathbf{p}_E) := \frac{-\overline{F}_1 + \overline{F}_2 + 2\overline{F}_3 - 2\overline{F}_4 - \overline{F}_7 + \overline{F}_8}{2\Delta x^2} \approx F_{xx}(\mathbf{p}_E) + \mathcal{O}(\Delta). \tag{18}$$

In a similar way, the following approximations are obtained:

$$\begin{aligned}\tilde{F}_{xy}(\mathbf{p}_E) &:= \frac{\overline{F}_1 - \overline{F}_2 - \overline{F}_7 + \overline{F}_8}{2\Delta x \Delta y} \approx F_{xy}(\mathbf{p}_E) + \mathcal{O}(\Delta), \\ \tilde{F}_{yy}(\mathbf{p}_E) &:= \frac{\overline{F}_1 - 2\overline{F}_4 + \overline{F}_7}{\Delta y^2} \approx F_{yy}(\mathbf{p}_E) + \mathcal{O}(\Delta).\end{aligned} \tag{19}$$

The same procedure can be applied to obtain approximations to the first-order derivatives:

$$\begin{aligned}\tilde{F}_x(\mathbf{p}_E) &:= \frac{-\overline{F}_1 + \overline{F}_2 - \overline{F}_7 + \overline{F}_8}{2\Delta x} \approx F_x(\mathbf{p}_E) + \mathcal{O}(\Delta^2), \\ \tilde{F}_y(\mathbf{p}_E) &:= \frac{-\overline{F}_1 - \overline{F}_2 + \overline{F}_7 + \overline{F}_8}{4\Delta y} \approx F_y(\mathbf{p}_E) + \mathcal{O}(\Delta^2),\end{aligned} \tag{20}$$

and also to the point value:

$$\begin{aligned}\tilde{F}(\mathbf{p}_E) :=& \frac{1}{24}(-5\overline{F}_1 + 4\overline{F}_2 - 4\overline{F}_3 + 30\overline{F}_4 - 5\overline{F}_7 + 4\overline{F}_8) \\ \approx& F(\mathbf{p}_E) + \mathcal{O}(\Delta^3).\end{aligned} \tag{21}$$

The above technique can be adapted in a straightforward way to obtain approximations at each point $\mathbf{p}_R$, $R \in \{N, S, W\}$.

Next, define the four second-order polynomials

$$\begin{aligned}P_R(\mathbf{x}) =& \tilde{F}(\mathbf{p}_R) + \tilde{F}_x(\mathbf{p}_R)(x - x_R) + \tilde{F}_y(\mathbf{p}_R)(y - y_R) \\ &+ \frac{1}{2}\tilde{F}_{xx}(\mathbf{p}_R)(x - x_R)^2 + \tilde{F}_{xy}(\mathbf{p}_R)(x - x_R)(y - y_R) \\ &+ \frac{1}{2}\tilde{F}_{yy}(\mathbf{p}_R)(y - y_R)^2, \quad R \in \{N, S, E, W\},\end{aligned} \tag{22}$$

that provide third-order approximations to $F(\mathbf{x})$ on the whole cell $V$. It can be easily checked that these reconstructions are *conservative*, i.e., they conserve the average value $\overline{F}_4$.

In the presence of shocks, the approximations to the second-order derivatives in (22) involve stencils having cells at both sides of the discontinuity: this produces small but undesiderable numerical disturbances. To avoid this behaviour and obtain a sharp resolution of shocks, first-order truncations of (22) are instead considered. To maintain conservation and get a second-order reconstruction on $V$, the coefficients of the first-order polynomials must be recalculated; for example, the following approximations are considered for the point $\mathbf{p}_E$:

$$\begin{aligned}\tilde{F}_x(\mathbf{p}_E) &:= \frac{\overline{F}_2 - 2\overline{F}_4 + \overline{F}_8}{2\Delta x} \approx F_x(\mathbf{p}_E) + \mathcal{O}(\Delta), \\ \tilde{F}_y(\mathbf{p}_E) &:= \frac{-\overline{F}_2 + \overline{F}_8}{2\Delta y} \approx F_y(\mathbf{p}_E) + \mathcal{O}(\Delta), \\ \tilde{F}(\mathbf{p}_E) &:= \frac{\overline{F}_2 + 2\overline{F}_4 + \overline{F}_8}{4} \approx F(\mathbf{p}_E) + \mathcal{O}(\Delta^2).\end{aligned} \tag{23}$$

The following step consist in combining the polynomials $P_R(\mathbf{x})$ in a WENO-like manner (see [20] for details). First, we build the smoothness indicators

$$\begin{aligned}\beta_R &= \iint_V \Delta x \Delta y \left(\tilde{F}_x(\mathbf{p}_R)^2 + \tilde{F}_y(\mathbf{p}_R)^2\right) d\mathbf{x} \\ &= \Delta x^2 \Delta y \tilde{F}_x(\mathbf{p}_R)^2 + \Delta x \Delta y^2 \tilde{F}_y(\mathbf{p}_R)^2,\end{aligned} \tag{24}$$

that are used to detect possible discontinuities in the stencil. Notice that second-order derivatives are not involved in (24), as the stencils used to cal-

culate them always leave cells at both sides of a discontinuity; this would lead to terms of the same order for each $\beta_R$.

The nonlinear weights $\omega_R$ are then defined as

$$\omega_R = \frac{\alpha_R}{\alpha_N + \alpha_S + \alpha_E + \alpha_W}$$

where

$$\alpha_R = \frac{\lambda_R}{(\varepsilon + \beta_R)^r}.$$

All the linear weights $\lambda_R$ take the value $1/4$, as they are not chosen to improve accuracy as in the one-dimensional WENO method. Typically, the values $\varepsilon = 10^{-6}$ and $r = 4$ give good results in the applications.

Finally, the reconstruction operator $P(\mathbf{x})$ on $V$ is defined as

$$P(\mathbf{x}) = \omega_N P_N(\mathbf{x}) + \omega_S P_S(\mathbf{x}) + \omega_E P_E(\mathbf{x}) + \omega_W P_W(\mathbf{x}), \quad \mathbf{x} \in V. \tag{25}$$

Let us resume the full procedure of the reconstruction:

(1) For each $R \in \{N, S, E, W\}$, construct the approximations $\tilde{F}_x(\mathbf{p}_R)$ and $\tilde{F}_y(\mathbf{p}_R)$ as in (20).
(2) Build the smoothness indicators $\beta_R$ using formula (24).
(3) If every $\beta_R$ is under the tolerance $\text{tol} = \Delta^2$, calculate $\tilde{F}_{xx}(\mathbf{p}_R)$, $\tilde{F}_{xy}(\mathbf{p}_R)$, $\tilde{F}_{yy}(\mathbf{p}_R)$ and $\tilde{F}(\mathbf{p}_R)$ as in (18), (19) and (21). Then use (22) to define the second-order polynomials $P_R(\mathbf{x})$.
(4) If $\beta_R > \text{tol}$ for some $R$, compute $\tilde{F}(\mathbf{p}_R)$, $\tilde{F}_x(\mathbf{p}_R)$ and $\tilde{F}_y(\mathbf{p}_R)$ as in (23), for each index $R$. Then build the corresponding first-order polynomials $P_R(\mathbf{x})$ by truncation of the second-order terms in (22).
(5) Finally, define the reconstruction operator $P(\mathbf{x})$ using formula (25).

For smooth functions, the reconstruction operator $P(\mathbf{x})$ verifies hypotheses (H1)–(H4) in section 2.2 with $p = q = 3$ and $m = 2$. Taking into account Remark 2, the expected order of the high-order method (16) is then equal to $\min(p, q) = 3$.

## 4 Implementation and numerical tests

To test the properties of the high-order scheme (16) and the benefits of its implementation using CUDA, we will focus on the one-layer shallow water

equations with topography:

$$\begin{cases} \dfrac{\partial h}{\partial t} + \dfrac{\partial q_x}{\partial x} + \dfrac{\partial q_y}{\partial y} = 0, \\ \dfrac{\partial q_x}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_x^2}{h} + \dfrac{g}{2}h^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_x q_y}{h}\right) = gh\dfrac{\partial H}{\partial x}, \\ \dfrac{\partial q_y}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_x q_y}{h}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_y^2}{h} + \dfrac{g}{2}h^2\right) = gh\dfrac{\partial H}{\partial y}. \end{cases} \tag{26}$$

Here $h(\mathbf{x}, t)$ denotes the thickness of the water layer, $q_\alpha(\mathbf{x}, t)$, $\alpha = x, y$, are the mass-flows in the coordinate directions, $H(\mathbf{x})$ represents the depth function (bathymetry) and $g$ is the gravity constant.

Let us denote $U = [h,\, q_x,\, q_y]^T$ and

$$F_1(U) = \left[q_x, \frac{q_x^2}{h} + \frac{1}{2}gh^2, \frac{q_x q_y}{h}\right]^T, \quad F_2(U) = \left[q_y, \frac{q_x q_y}{h}, \frac{q_y^2}{h} + \frac{1}{2}gh^2\right]^T,$$
$$S_1(U) = [0, gh, 0]^T, \quad S_2(U) = [0, 0, gh]^T.$$

Let $J_i(U) = \dfrac{\partial F_i}{\partial U}(U)$ be the Jacobian of the flux $F_i$, for $i = 1, 2$. Given an unit vector $\boldsymbol{\eta} = (\eta_x, \eta_y) \in \mathbb{R}^2$, we define the matrix $A(U, \boldsymbol{\eta}) = J_1(U)\eta_x + J_2(U)\eta_y$, and the vectors $\boldsymbol{F_\eta}(U) = F_1(U)\eta_x + F_2(U)\eta_y$ and $\boldsymbol{S_\eta}(U) = \eta_x S_1(U) + \eta_y S_2(U)$.

In the particular case of the one-layer shallow water system, the numerical scheme (16) reads as follows (see [2] for more details):

$$U_i'(t) = -\frac{1}{|V_i|}\left[\sum_{j\in\mathcal{N}_i} |E_{ij}| \sum_{l=1}^{n(\bar{r})} w_l \mathcal{D}^-(U_{ij,l}^-(\gamma,t), U_{ij,l}^+(\gamma,t), H_{ij,l}^-(\gamma), H_{ij,l}^+(\gamma), \boldsymbol{\eta}_{ij}) \right.$$
$$\left. - |V_i| \sum_{l=1}^{n(\bar{s})} \alpha_l \left( S_1(P_i^t(\mathbf{x}_l^i)) \frac{\partial P_i^H}{\partial x}(\mathbf{x}_l^i) + S_2(P_i^t(\mathbf{x}_l^i)) \frac{\partial P_i^H}{\partial y}(\mathbf{x}_l^i) \right) \right]. \tag{27}$$

Next, the notation used in (27) is detailed. $P_i^t$ is the reconstruction function corresponding to the cell value $U_i(t)$, while $P_i^H$ is the reconstruction function associated to the cell averages of the given bathymetry. $U_{ij}^\pm(\gamma, t)$ and $H_{ij}^\pm(\gamma)$ are given, respectively, by

$$U_{ij}^-(\gamma,t) = \lim_{\mathbf{x}\to\gamma} P_i^t(\mathbf{x}), \quad U_{ij}^+(\gamma,t) = \lim_{\mathbf{x}\to\gamma} P_j^t(\mathbf{x})$$

and

$$H_{ij}^-(\gamma) = \lim_{\mathbf{x}\to\gamma} P_i^H(\mathbf{x}), \quad H_{ij}^+(\gamma) = \lim_{\mathbf{x}\to\gamma} P_j^H(\mathbf{x}).$$

$U_{ij,l}^\pm$ (respectively $H_{ij,l}^\pm$) corresponds to $U_{ij}^\pm(\cdot)$ (respectively $H_{ij}^\pm(\cdot)$) evaluated

at the quadrature points of the edge $E_{ij}$. Moreover,

$$\mathcal{D}^{\pm}(U_L, U_R, H_L, H_R, \boldsymbol{\eta}) = \boldsymbol{F}_{\boldsymbol{\eta}}(U_L) + P^{\pm}_{LR}\left(A_{LR}(U_R - U_L) - \boldsymbol{S}_{LR}(H_R - H_L)\right). \quad (28)$$

In the particular case of system (26),

$$A_{LR} = \begin{bmatrix} 0 & \eta_x & \eta_y \\ (-\bar{u}_x^2 + \bar{c}^2)\eta_x - \bar{u}_x\bar{u}_y\eta_y & 2\bar{u}_x\eta_x + \bar{u}_y\eta_y & \bar{u}_x\eta_y \\ -\bar{u}_x\bar{u}_y\eta_x + (-\bar{u}_y^2 + \bar{c}^2)\eta_y & \bar{u}_y\eta_x & \bar{u}_x\eta_x + 2\bar{u}_y\eta_y \end{bmatrix}$$

and

$$\boldsymbol{S}_{LR} = \left[0,\, \bar{c}^2\eta_x,\, \bar{c}^2\eta_y\right]^T$$

with

$$\bar{h} = \frac{h_L + h_R}{2}, \quad \bar{c} = \sqrt{g\bar{h}}, \quad \bar{u}_\alpha = \frac{\sqrt{h_L}u_{L,\alpha} + \sqrt{h_R}u_{R,\alpha}}{\sqrt{h_L} + \sqrt{h_R}}, \quad \alpha = x, y, \quad (29)$$

and the usual definitions of the velocity $u = q/h$ and the elevation $\eta = h - H$. Finally,

$$P^{\pm}_{LR} = \frac{1}{2}\mathcal{K}_{LR}(I \pm \mathrm{sgn}(\Lambda_{LR}))\mathcal{K}^{-1}_{LR}, \quad (30)$$

where $\Lambda_{LR}$ is the diagonal matrix whose coefficients are the eigenvalues of $A_{LR}$, $\mathcal{K}_{LR}$ is a matrix whose columns are associated eigenvectors, and $\mathrm{sgn}(\mathcal{L}_{LR})$ is the diagonal matrix whose coefficients are the signs of the eigenvalues of the matrix $A_{LR}$.

### *4.1 CUDA Implementation*

In this section we describe the potential data parallelism of the numerical scheme and its implementation in CUDA.

#### *4.1.1 Parallelism sources*

Initially, the finite volume mesh must be constructed from the input data with the appropriate setting of initial and boundary conditions. Then the time stepping is performed by applying a third-order Runge-Kutta TVD method, consisting on three steps. At each step, the spatial discretization (27) must be performed as follows:

(1) **Reconstruction and volume integral computation:** First a reconstruction procedure at each cell and for each variable must be performed

to define the functions $P_i(\mathbf{x})$ given by (25). Next, the numerical approximation of the volume integral is computed using a third-order Gaussian quadrature formula

$$\Sigma_i = -|V_i| \sum_{l=1}^{n(\bar{s})} \alpha_l \left( S_1(P_i(\mathbf{x}_l^i)) \frac{\partial P_i^H}{\partial x}(\mathbf{x}_l^i) + S_2(P_i(\mathbf{x}_l^i)) \frac{\partial P_i^H}{\partial y}(\mathbf{x}_l^i) \right).$$

Next, the reconstructed values $U_{ij,l}$ at the quadrature points of each edge of the cell $V_i$ are also computed. Again, a third-order Gaussian quadrature formula is used. Therefore, two values must be computed at each edge of $V_i$.

(2) **Edge-based calculations:** The following computations must be performed at each edge $E_{ij}$ common to cells $V_i$ and $V_j$, using the reconstructed values $U_{ij,l}^-$ and $U_{ij,l}^+$ previously calculated:

$$\Sigma_{ij}^{\pm} = |E_{ij}| \sum_{l=1}^{n(\bar{r})} w_l \mathcal{D}^{\pm}(U_{ij,l}^-(\gamma,t), U_{ij,l}^+(\gamma,t), H_{ij,l}^-(\gamma), H_{ij,l}^+(\gamma), \boldsymbol{\eta}_{ij}).$$

(3) **Volume-based calculations:** At each cell $V_i$, the following computations must be performed:
   a) Computation of the local $\Delta t_i$ for each volume.
   b) Computation of $U_i^{n+1,s}$: The $n+1, s$-th state of each volume must be approximated from the $n$-th and the $n+1, s-1$-th states using the data computed at the previous steps.

Several remarks can be made related to the description of the parallel algorithm. The computation steps required by the problem addressed here can be classified into two groups: computations associated to edges and computations associated to volumes. The scheme exhibits a high degree of data parallelism because the computation at each edge/volume is independent with respect to the computation performed at the rest of edges/volumes. Moreover, the scheme presents a high arithmetic intensity and the computation exhibits a high degree of locality. These remarks indicate that this problem is suitable for being implemented on GPUs using CUDA.

### *4.1.2 Implementation details*

We consider problems consisting in a bidimensional regular finite volume mesh. Each processing step executed on the GPU is assigned to a CUDA kernel. A kernel is a function executed on the GPU, which is executed forming a grid of thread blocks that run logically in parallel (see [15] for more details). Next, we describe each step.

- **Build the data structure:** For each volume, we store its state ($h$, $q_x$ and $q_y$) and its depth $H$. We define an array of `float4` elements, where

each element represents a volume and contains the former parameters. This array is stored as a 2D texture since each edge (thread) only needs the data of its two adjacent volumes, and texture memory is especially suited for each thread to access its closer environment in texture memory. The per-block shared memory, on the other hand, is more suitable when each thread needs to access many elements located in global memory, and each thread of a block loads a small part of these elements into shared memory. We first implemented a CUDA program using shared memory instead of a texture, where each thread of a block loaded the data of a volume into shared memory, but later we got better execution times by using a texture.

The area of the volumes and the length of the vertical and horizontal edges are precalculated and passed to the CUDA kernels that need them.

We can know at runtime if an edge or volume is frontier or not and the value of $\boldsymbol{\eta}_{ij}$ at an edge by checking the position of the thread in the grid.

- **Reconstruction and integral computation:** In this step, the reconstruction values $U^{\pm}_{ij,l}$, $l = 1, 2$, are computed and stored in four arrays located in global memory, each one being an array of `float3` elements. The size of each array is twice the number of volumes and they are associated to the four edges of a cell (south, north, east and west). Moreover, the integral term $\Sigma_i$ is also computed and stored in an accumulator placed in global memory. This accumulator is an array of `float4` elements and its size is the number of volumes. This accumulator is also used to store the contributions of the vertical edges. In this process, each thread represents a finite volume cell.
- **Process vertical and horizontal edges:** We divide the edge processing into vertical and horizontal edge processing. For vertical edges $\eta_{ij,y} = 0$, and for horizontal edges $\eta_{ij,x} = 0$. Therefore, all the operations where these terms take part can be avoided, thus increasing efficiency.

  Here, each thread represents a vertical or a horizontal edge, and computes the contribution to their adjacent volumes as described in section 4.1.1.

  The edges (i.e., threads) synchronize each other when contributing to a particular volume by means of two accumulators stored in global memory, each one being an array of `float4` elements. Note that one of them has been previously used to store the integral cell computation. The size of each accumulator is the number of volumes. Each element of the accumulators stores the edge contributions to the volume (a $3 \times 1$ vector, $\Sigma^{\pm}_{ij}$, and a `float` value storing $\|\Lambda_{ij}\|_{\infty}$). In the processing of vertical edges, each edge writes the contribution to its right-side volume in the first accumulator, and the contribution to its left-side volume in the second accumulator. Next, the processing of horizontal edges is performed in an analogous way, with the difference that the contribution is added to the accumulators.
- **Compute $\Delta t_i$ for each volume:** In this step, each thread represents a volume and the local $\Delta t_i$ of the volume $V_i$ is computed using the CFL condition (6).
- **Get the minimum $\Delta t$:** This step finds the minimum of the local $\Delta t_i$ of the volumes by applying a reduction algorithm on the GPU. The reduction

algorithm applied is the kernel 7 (the most optimized one) of the reduction sample included in the CUDA Software Development Kit [15].

- **Compute $U_i^{n+1,s}$ for each volume:** In this step, each thread represents a volume and the state $U_i$ of the volume $V_i$ is updated. The final value is obtained by adding up the two $3 \times 1$ vectors stored in the positions corresponding to the volume $V_i$ in both accumulators. Since a CUDA kernel cannot directly write into textures, the texture is initially updated by writing the results into a temporal array, which is then copied to the CUDA array bound to the texture.

### *4.2 Numerical experiments*

In this section, the reconstructions in space are performed using the third-order operator introduced in Section 3, while for time stepping an optimal TVD Runge-Kutta scheme is applied (see [20]). Also, a Gaussian quadrature of order three is used to approximate the integral terms.

Two different codings of the scheme have been implemented: the CPU code was written in C++, while the GPU code was programmed in CUDA, following the guidelines stated in Section 4.1. The CPU was an Intel Xeon E5430 (2.66 GHz 12MB L2 Cache), while two different GPUs have been used: a NVIDIA GeForce 9800GTX (128 stream processors with 512MB) and a NVIDIA GeForce GTX260 (192 stream processors with 869MB).

#### *4.2.1 Verification of the C-property*

In this test the C-property (see [1], for example) of the scheme is numerically verified. Consider a depth function of the form

$$H(\mathbf{x}) = 2 - (x - 0.5)^2 - (y - 0.5)^2, \quad \mathbf{x} \in [0, 1] \times [0, 1]$$

with a random perturbation (see figure 3). The initial conditions are given by $h(\mathbf{x}, 0) = H(\mathbf{x})$ and $q_x(\mathbf{x}, 0) = q_y(\mathbf{x}, 0) = 0$. A rectangular mesh with $\Delta x = \Delta y = 0.01$ has been considered, and CFL=0.9. As it can be seen in table 1, the stationary solution is exactly preserved up to machine accuracy.

Table 1
Test case 4.2.1. Verification of the C-property. $L^1$ errors.

| Precision | Error $h$ | Error $q_x$ | Error $q_y$ |
|---|---|---|---|
| Single | 1.35E–08 | 2.63E–07 | 3.28E–07 |
| Double | 5.56E–17 | 6.48E–18 | 6.43E–18 |

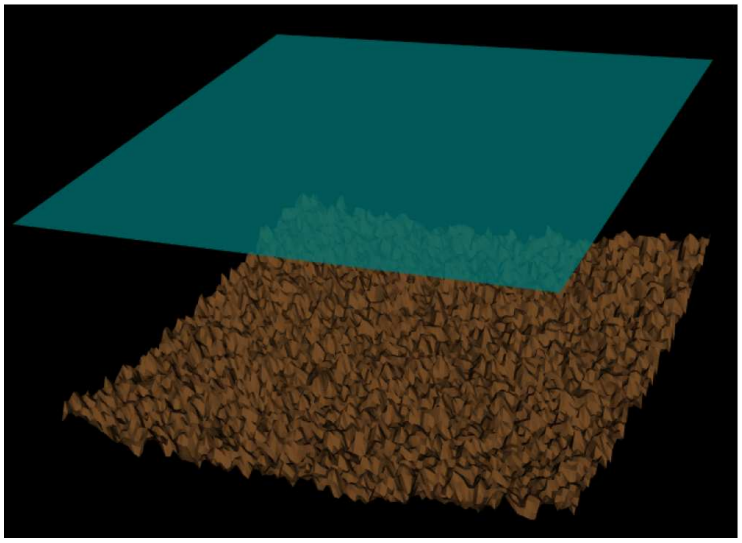

Fig. 3. Finite volume discretization.

*4.2.2 Subcritical stationary solution*

The depth function for this experiment is taken as

$$H(\mathbf{x}) = 2 - 0.2\exp(-0.16(x-10)^2), \quad \mathbf{x} \in [0,20] \times [0,20].$$

The initial conditions correspond to the two-dimensional extension of a one-dimensional subcritical stationary solution (see [2] for the details). As boundary conditions, $q_x(0,y,t) = 0.15$ is fixed at $x = 0$, and $h(20,y,t) = 0.5$ is imposed at $x = 20$. Solid walls are considered at $y = 0$ and $y = 20$. The CFL is set to 0.9.

The $L^1$ errors obtained with several meshes are shown in table 2. As expected, the scheme achieves third order of accuracy.

Table 2
Test case 4.2.2. $L^1$ errors and orders.

| N. cells | Error $h$ | Order $h$ | Error $q_x$ | Order $q_x$ |
|---|---|---|---|---|
| $10 \times 10$ | 1.22E–02 | – | 7.76E–02 | – |
| $20 \times 20$ | 3.93E–03 | 1.63 | 1.04E–02 | 2.90 |
| $40 \times 40$ | 5.71E–04 | 2.78 | 8.42E–04 | 3.63 |
| $80 \times 80$ | 4.87E–05 | 3.55 | 3.89E–05 | 4.44 |
| $160 \times 160$ | 3.32E–06 | 3.87 | 1.86E–06 | 4.39 |

### *4.2.3 Circular dam-break problem*

This experiment concerning a circular dam-break problem has been taken from [2]. The bottom is defined by the function

$$H(\mathbf{x}) = \begin{cases} 0.6 - b(\mathbf{x}) & \text{if } (x-1.5)^2 + (y-1)^2 \leq 0.5^2, \\ 0.6 & \text{otherwise,} \end{cases}$$

where $b(\mathbf{x}) = \frac{1}{8}(\cos(2\pi(x-0.5)) + 1)(\cos(2\pi y) + 1)$, on the computational domain $[0,2] \times [0,2]$.

The initial conditions are given by

$$h(\mathbf{x}) = \begin{cases} H(\mathbf{x}) + 0.5 & \text{if } (x-1.25)^2 + (y-1)^2 \leq 0.1^2, \\ H(\mathbf{x}) & \text{otherwise,} \end{cases}$$

and $q_x(\mathbf{x}, 0) = q_y(\mathbf{x}, 0) = 0$. Boundary conditions of absorbing type at $x = 0$ and $x = 2$, and solid walls at $y = 0$ and $y = 2$, have been considered. The CFL is set to 0.9.

The computed solutions at different times are represented in figure 4, where the reference solution has been calculated on a mesh of $800 \times 800$ cells. The results obtained are comparable to those found in [2], where third-order bi-hyperbolic reconstructions were applied.

A comparison between the CPU and GPUs times is presented in table 3. Finally, figure 5 shows the speedup curves obtained with both GPUs.

Table 3
Test case 4.2.3. CPU versus GPU times (in seconds).

| N. cells | CPU | 9800GTX | GTX260 |
|---|---|---|---|
| $50 \times 50$ | 2.23 | 0.0858 | 0.046 |
| $100 \times 100$ | 18.21 | 0.461 | 0.20 |
| $200 \times 200$ | 147.78 | 3.362 | 1.38 |
| $400 \times 400$ | 1175.28 | 25.596 | 10.54 |
| $800 \times 800$ | 9414 | 201.896 | 84.08 |
| $1200 \times 1200$ | 31405 | 673.30 | 279.20 |

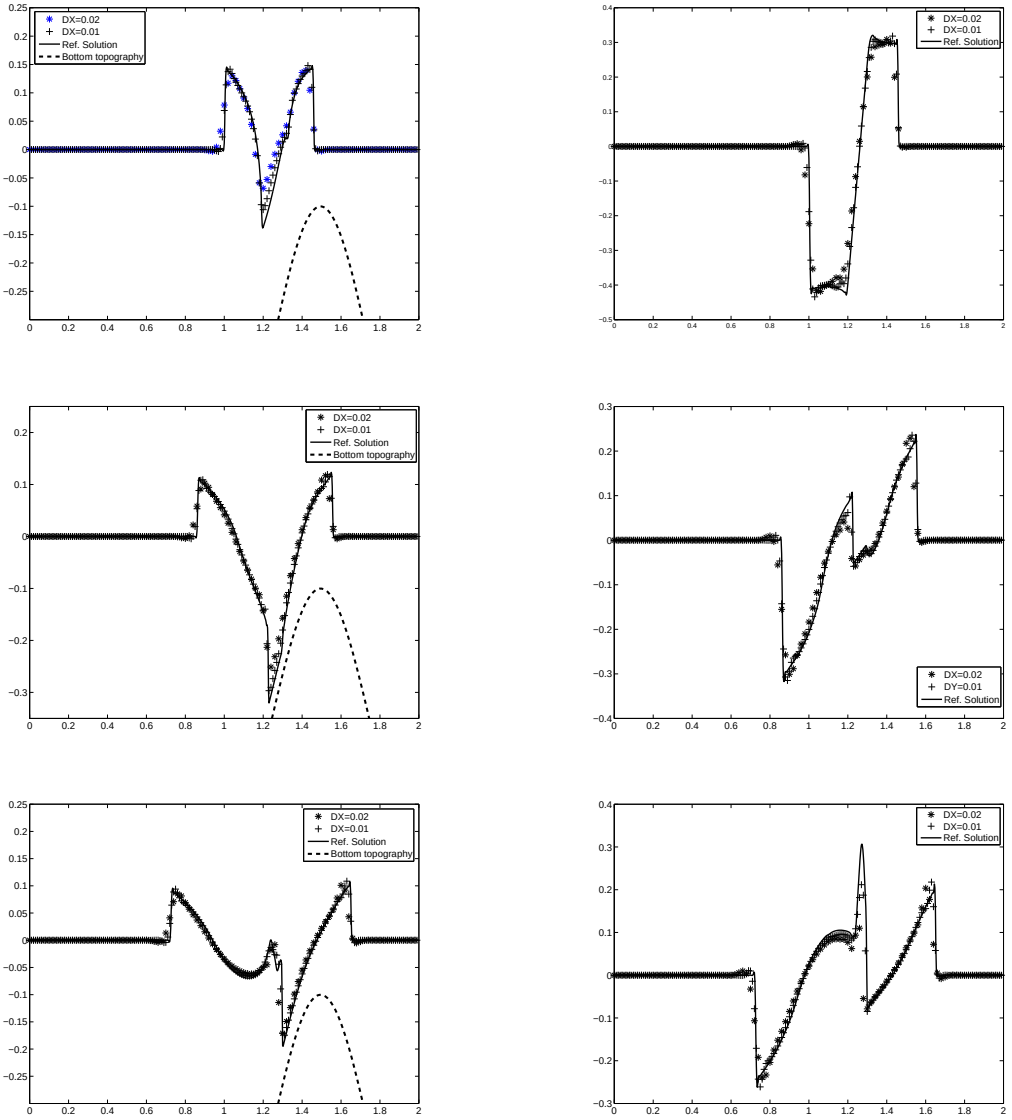


Fig. 4. Circular dam-break problem: results obtained at times $t = 0.05$, 0.1 and 0.15 (top to bottom) at the longitudinal section $y = 1$, with $\Delta x = 0.02$ and $\Delta x = 0.01$. Free surface (left) and discharge $q_x$ (right).

## 5 Conclusions

A new kind of reconstruction operator of polynomial type, that is third-order accurate on each computational cell, has been developed. The method can be extended to general nonuniform quadrilateral meshes. An efficient high order well-balanced finite volume solver for one-layer shallow water systems has been derived and implemented using the CUDA framework. This solver implements optimization techniques to efficiently parallelize the numerical scheme on the CUDA architecture. Simulations carried out on GeForce 9800GTX and

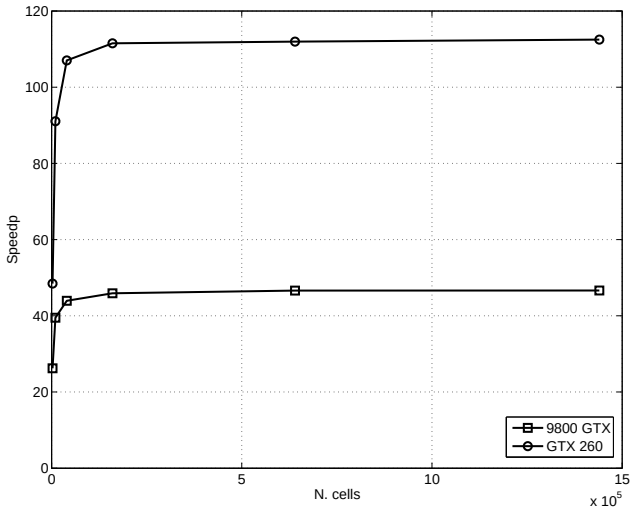


Fig. 5. Test case 4.2.3. GPUs speedup curves.

GeForce GTX260 cards using single precision were found to be up to two orders of magnitude faster than a monocore version of the solver for big-size uniform problems. These simulations also show that the numerical solutions obtained with the solver are accurate enough for practical applications. As further work, we propose to extend the strategy to enable efficient simulations on non-structured meshes and to extend the CUDA implementation to other models like two-layer shallow water systems.

## Acknowledgments

The authors would like to thank Professors Manuel J. Castro (University of Málaga) and Enrique D. Fernández (University of Seville) for many valuable suggestions and discussions.

## References

[1] M.J. Castro, J.M. Gallardo, C. Parés (2006). High order finite volume schemes based on reconstruction of states for solving hyperbolic systems with nonconservative products. Applications to shallow-water systems. Math. Comp. 75, 1103–1134.

[2] M.J. Castro, E.D. Fernández, A.M. Ferreiro, A. García, C. Parés (2009). High order extension of Roe schemes for two dimensional nonconservative hyperbolic systems. J. Sci. Comput. 39, 67–114.

[3] M. Dumbser, M. Käser (2007). Arbitrary high order non-oscillatory finite volume schemes on unstructured meshes for linear hyperbolic systems. J. Comput. Phys. 221, 693–723.

[4] T.R. Hagen, J.M. Hjelmervik, K.A. Lie, J.R. Natvig, M. Ofstad (2005). Visual simulation of shallow-water waves. Sim. Modelling Pract. and Th. 13, 716–726.

[5] A. Harten, J.M. Hyman (1983). Self-adjusting grid methods for one-dimensional hyperbolic conservation laws. J. Comp. Phys. 50, 235–269.

[6] C. Hu, C.-W. Shu (1999). Weighted essentialy non-oscillatory schemes on triangular meshes. J. Comput. Phys. 150, 97–127.

[7] G. Jiang, C.-W. Shu (1996). Efficient implementation of weighted ENO schemes. J. Comput. Phys. 126, 202–228.

[8] M. Lastra, J.M. Mantas, C. Ureña, M.J. Castro, J.A. García (2009). Simulation of shallow-water systems using graphics processing units. Math. Comp. Simul. 80, 598–618.

[9] X.D. Liu, S. Osher, T. Chan (1994). Weighted essentially nonoscillatory schemes. J. Comput. Phys. 115, 200–212.

[10] M. de la Asunción, J.M. Mantas, M.J. Castro (2009). Simulation of one-layer shallow water systems on multicore and CUDA architectures. Submitted to J. Supercomputing.

[11] A. Marquina (1994). Local piecewise hyperbolic reconstructions for nonlinear scalar conservation laws. SIAM J. Sci. Comput. 15, 892–915.

[12] R.J. LeVeque (1998). Balancing source terms and flux gradients in high-resoluton Godunov methods: the quasi-steady wave-propagation algorithm. J. Comput. Phys. 146, 346–365.

[13] S. Noelle, N. Pankratz, G. Puppo, J. Natvig (2006). Well-balanced finite volume schemes of arbitrary order of accuracy for shallow water flows. J. Comput. Phys. 213, 474–499.

[14] http://www.nvidia.com,

[15] NVIDIA. CUDA Zone. http://www.nvidia.com/object/cuda_home.html. Accessed November 2009.

[16] C. Parés (2006). Numerical methods for nonconservative hyperbolic systems: a theoretical framework. SIAM J. Num. Anal. 44, 300– 321.

[17] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. Purcell (2005). A Survey of General-Purpose Computation on Graphics Hardware, Eurographics 2005 State of the Art Report.

[18] M. Rumpf, R. Strzodka (2006). Graphics Processor Units: New Prospects for Parallel Computing, L. N. in Computational Science and Engineering 51, 89–121.

[19] H.J. Schroll, F. Svensson (2006). A bi-hyperbolic finite volume method on quadrilateral meshes. J. Sci. Comp. 26, 237-260.

[20] C.-W. Shu (1997). Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. ICASE Report n. 97–65.

[21] C.-W. Shu, S. Osher (1998). Efficient implementation of essentially non-oscillatory shock capturing schemes. J. Comput. Phys. 77, 439–71.

[22] G. Walz (1997). Romberg Type Cubature over Arbitrary Triangles. Mannheimer Mathem. Manuskripte Nr.225, Mannhein.