Efficient multilayer shallow-water simulation system based on GPUs

Miguel Lastra[a,*], Manuel J. Castro Díaz[b], Carlos Ureña[a], Marc de la Asunción[a]

[a]*Depto. Lenguajes y Sistemas Informáticos. E.T.S Ingeníeria Informática. Periodista Daniel Saucedo Aranda s/n. Universidad de Granada.*
[b]*Depto. Análisis Matemático. Facultad de Ciencias.Campus de Teatinos s/n. Universidad de Málaga.*

Abstract

The computational simulation of shallow stratified fluids is a very active research topic because these types of systems are very common in a variety of natural environments. The simulation of such systems can be modelled using multilayer shallow-water equations but do impose important computational requisites especially when applied to large domains.

General Purpose Computing on Graphics Processing Units (GPGPU) has become a vivid research field due to the arrival of massively parallel hardware platforms (based on graphics cards) and adequate programming frameworks which have allowed important speed-up factors with respect to not only sequential but also parallel CPU based simulation systems.

In this work we present a proposal for the simulation of shallow stratified fluids with an arbitrary number of layers using GPUs. The designed system does fully adapt to the many-core architecture of modern GPUs and several experiments have been carried out to illustrate its scalability and behavior on different GPU models. We propose a new elaborated 3D computational scheme for an underlying 2D mathematical model. This scheme allowed implementing a system capable of handling an arbitrary number of layers. The system adds no overhead when used for two-layer scenarios, compared to an existing 2D system specifically designed for just two layers.

Our proposal is aimed at creating a GPU-based computational scheme suitable for the simulation of multilayer large-scale real-world scenarios.

Keywords: Shallow-water, finite volumes, simulation, GPU, CUDA

1. Introduction

The simulation of free surface or internal waves in shallow stratified fluids are commonly modelled by the multilayer shallow-water equations formulated as a conservation law with non-conservative products and source terms. Stratified fluids are ubiquitous in nature: they appear in atmospheric flows, ocean currents, estuarine systems,... This is the situation, for instance, in the Strait of Gibraltar, where surface water from the Atlantic inflows over saltier westwards-flowing Mediterranean water. Simulating those phenomena requires very long lasting simulations in big computational domains which is why very efficient implementations are needed to be able to analyze those problems in affordable computational times.

A 2D multilayer shallow-water system can be discretized by the natural extension to 2D domains of a first order PVM path-conservative type finite volume scheme introduced in [8]. PVM schemes have been introduced in [8] in the framework of balance laws and non-conservative hyperbolic systems. They are defined in terms of viscosity matrices computed by a suitable polynomial evaluation of a Roe Matrix. These

[*]Corresponding author
Email addresses: mlastral@ugr.es (Miguel Lastra), castro@anamat.cie.uma.es (Manuel J. Castro Díaz), curena@ugr.es (Carlos Ureña), marc@correo.ugr.es (Marc de la Asunción)
[1]Corresponding author. Tel: +34958246144

methods have the advantage that they only need some information about the eigenvalues of the system and no spectral decomposition of the Roe Matrix is required. As consequence, they are much faster than Roe schemes for systems with an increasing number of unknowns. In this work the PVM-2U method has been chosen among the PVM schemes introduced in [8] as it provides the best results, concerning computational time and accuracy. This method can be seen as a the natural extension of the scheme introduced by Degond et al. in [7] for non-conservative systems.

Since the appearance of computing frameworks like Cuda [14] and openCL [12] the huge computational capabilities of modern Graphics Processing Units (GPUs) have been unveiled for computationally intensive physically based simulations. Nevertheless, elaborated software designs are required to take full advance of the processing power offered by GPUs.

The Cuda computational model, provided by Nvidia [15], requires the computational tasks to be mapped to a set of threads that will be run in parallel. Each of these threads will run the same program (kernel) where the number of threads running at a point in time will depend on the hardware requirements of each thread. At the lower level, threads are run in groups of 32 threads called warps and it is necessary to avoid code divergences inside a warp to avoid serialization. On the other hand, coalesced memory accesses are also essential to allow several threads access the required data stored in memory by only issuing one physical memory access for a group of threads. When a warp requires a memory access operation, the whole warp is stopped until that memory operation has finished and another warp is run. Modern GPUs provide a large amount of registers to avoid costly context switches by keeping a large amount of threads in a ready to run state. Taking into account GPUs currently provide in the order of thousands of computing cores, a large amount of threads is required to avoid computing cores not receiving enough workload and idling. This means any simulation scheme must be designed to provide a high degree of thread level parallelism. The higher the number of threads that can be kept active, the higher the value of a metric called occupancy.

On the other hand, as in many architectures, computing cores have pipelines which should be kept as full as possible to maximize the performance. This requires an adequate level of instruction level parallelism which means having certain degree of independence between the instructions run by each thread. This fact requires that each thread must also receive an adequate level of workload and as much independence as possible from memory accesses (arithmetic intensity).

The computing cores in Nvidia branded hardware are grouped into multiprocessors (SMXs) and threads are also grouped into thread blocks. Each thread block is run on the same SMX and this allows using an intra-block synchronization mechanism and also intra-block collaboration through the use of a high-speed shared memory (shared among the threads of each block).

Dividing the computations that make up the simulation program into independent threads and grouping them into blocks requires also to consider the resources required by each thread and several hardware level limits like the maximum resident threads and blocks per SMX to keep the occupancy level high (although a higher occupancy does not automatically translate into a higher performance).

There are many examples in literature of successfully using GPUs for shallow-water simulation [5, 4, 2, 10, 3], even before frameworks like Cuda were available, or widely used, and physical simulations were run using the standard graphics pipeline [11, 13]. In this work we concentrate on multilayer environments with an arbitrary number of layers simulated using Cuda on Nvidia GPUs.

This paper is organized as follows: in Section 2 the mathematical model that has been used is described, Section 3 describes de numerical model, Section 4 addresses the multilayer computational model proposed in this work and Section 5 presents the set of experiments that have been performed. Finally, Section 6 presents the conclusions that were reached.

2. Mathematical model

Let us consider the system of equations governing the 2D flow of m superposed immiscible layers of shallow homogeneous fluids with constant densities in a subdomain $D \subset \mathbb{R}^2$:

$$\begin{cases} \partial_t h_l + \partial_x q_{x,l} + \partial_y q_{y,l} = 0, \\ \partial_t q_{x,l} + \partial_x \left(\dfrac{q_{x,l}^2}{h_l} + \dfrac{1}{2} g h_l^2 \right) + \partial_y \left(\dfrac{q_{x,l} q_{y,l}}{h_l} \right) + g h_l \partial_x \left(\displaystyle\sum_{k>l} h_k + \sum_{k<l} \frac{\rho_k}{\rho_l} h_k - H \right) = 0. \\ \partial_t q_{y,l} + \partial_x \left(\dfrac{q_{x,l} q_{y,l}}{h_l} \right) + \partial_y \left(\dfrac{q_{y,l}^2}{h_l} + \dfrac{1}{2} g h_l^2 \right) + g h_l \partial_y \left(\displaystyle\sum_{k>l} h_k + \sum_{k<l} \frac{\rho_k}{\rho_l} h_k - H \right) = 0 \end{cases} \tag{1}$$

where $l = 1, \cdots, m$ being m is the number of layers, $h_l(\boldsymbol{x}, t)$ and $\boldsymbol{q}_l(\boldsymbol{x}, t) = (q_{x,l}(\boldsymbol{x}, t), q_{y,l}(\boldsymbol{x}, t))$ are, respectively, the thickness and the mass-flow of the l-th layer at point $\boldsymbol{x}$ at time t, and they are related to the mean velocities $\boldsymbol{u}_l(\boldsymbol{x}, t) = (u_{x,l}(\boldsymbol{x}, t), u_{y,l}(\boldsymbol{x}, t))$, by the equalities: $\boldsymbol{q}_l(\boldsymbol{x}, t) = \boldsymbol{u}_l(\boldsymbol{x}, t) h_l(\boldsymbol{x}, t)$; $H(\boldsymbol{x})$ is the basin depth measured from a fixed reference level, g is the gravity constant and ρ_j the densities of each layer verifying

$$0 < \rho_1 < \cdots < \rho_m.$$

Notice that h_1 is the height of the layer of fluid on the top and h_m is the height of the layer of fluid over the bottom (See Figure 1).

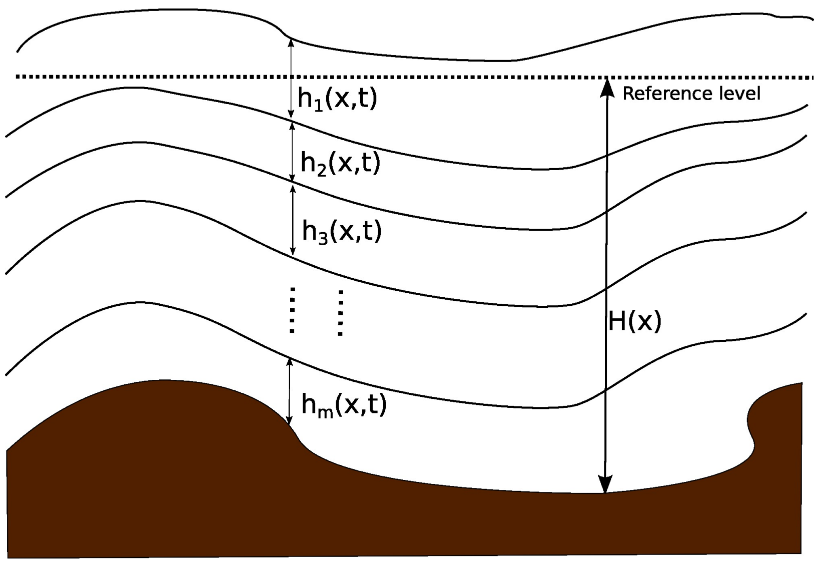

Figure 1: Multilayer stratified flow

This system can be written under the structure of a system of balance laws with non-conservative products

$$\partial_t w + \partial_x F_x(w) + \partial_y F_y(w) + B_x(w) \partial_x w + B_y(w) \partial_y w = G_x(w) \partial_x H + G_y(w) \partial_y H \tag{2}$$

where

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix}, \quad w_l = \begin{pmatrix} h_l \\ q_{x,l} \\ q_{y,l} \end{pmatrix} = \begin{pmatrix} h_l \\ u_{x,l} h_l \\ u_{y,l} h_l \end{pmatrix},$$

$$F_x(w) = \begin{pmatrix} F_{x,1} \\ \vdots \\ F_{x,m} \end{pmatrix}, \quad F_{x,l} = \begin{pmatrix} u_{x,l}h_l \\ u_{x,l}^2 h_l + \dfrac{1}{2} g h_l^2 \\ u_{x,l}u_{y,l}h_l \end{pmatrix},$$

$$F_y(w) = \begin{pmatrix} F_{y,1} \\ \vdots \\ F_{y,m} \end{pmatrix}, \quad F_{y,l} = \begin{pmatrix} u_{y,l}h_l \\ u_{x,l}u_{y,l}h_l \\ u_{y,l}^2 h_l + \dfrac{1}{2} g h_l^2 \end{pmatrix}$$

$$G_x(w) = \begin{pmatrix} G_{x,1} \\ \vdots \\ G_{x,m} \end{pmatrix}, \quad G_{x,l} = \begin{pmatrix} 0 \\ gh_l \\ 0 \end{pmatrix}, \quad G_y(w) = \begin{pmatrix} G_{y,m} \\ \vdots \\ G_{y,m} \end{pmatrix}, \quad G_{y,l} = \begin{pmatrix} 0 \\ 0 \\ gh_l \end{pmatrix}.$$

$B_x(w)$ is the $3m \times 3m$ matrix defined by

$$B_x(w) = \begin{pmatrix} \mathbf{0} & B_x^{1,2} & B_x^{1,3} & \cdots & B_x^{1,m} \\ B_x^{2,1} & \mathbf{0} & B_x^{2,3} & \cdots & B_x^{2,m} \\ \vdots & \vdots & \mathbf{0} & \cdots & \vdots \\ B_x^{m,1} & B_x^{m,2} & B_x^{m,3} & \cdots & \mathbf{0} \end{pmatrix}$$

where $B_x^{l,k}$, $k, l = 1, \cdots, m$, $k \neq l$ are 3×3 matrices defined by

$$B_x^{l,k} = \begin{cases} \begin{pmatrix} 0 & 0 & 0 \\ \rho_k/\rho_l h_l & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \text{if } k < l \\ \begin{pmatrix} 0 & 0 & 0 \\ h_l & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \text{otherwise.} \end{cases}$$

$B_y(w)$ is the $3m \times 3m$ matrix defined by

$$B_y(w) = \begin{pmatrix} \mathbf{0} & B_y^{1,2} & B_y^{1,3} & \cdots & B_y^{1,m} \\ B_y^{2,1} & \mathbf{0} & B_y^{2,3} & \cdots & B_y^{2,m} \\ \vdots & \vdots & \mathbf{0} & \cdots & \vdots \\ B_y^{m,1} & B_y^{m,2} & B_y^{m,3} & \cdots & \mathbf{0} \end{pmatrix}$$

where $B_y^{l,k}$, $k, l = 1, \cdots, m$, $k \neq l$ are 3×3 matrices defined by

$$B_y^{l,k} = \begin{cases} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \rho_k/\rho_l h_l & 0 & 0 \end{pmatrix} & \text{if } k < l \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ h_l & 0 & 0 \end{pmatrix} & \text{otherwise.} \end{cases}$$

Let us define the matrices $A_\alpha(w) = J_\alpha(w) + B_\alpha(w)$, $\alpha = x, y$ where $J_\alpha(w) = \frac{\partial F_\alpha}{\partial w}(w)$ are the Jacobians of the fluxes F_α, and we assume that $w \in \Omega \subset \mathbb{R}^{3m}$, so that (1) is strictly hyperbolic.

Let us also remark that the system (1) verifies the property of invariance by rotations. Effectively, given $\eta \in \mathbb{R}^2$ with $\|\eta\| = 1$, let us define

$$T_\eta = \begin{pmatrix} R_\eta^{1,1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & R_\eta^{2,2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & R_\eta^{m,m} \end{pmatrix}, \quad R_\eta^{l,l} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \eta_x & \eta_y \\ 0 & -\eta_y & \eta_x \end{pmatrix}, \quad l = 1, \cdots, m$$

and let us denote $F_\eta(w) = F_x(w)\eta_x + F_y(w)\eta_y$, $\boldsymbol{B}(w) = (B_x(w), B_y(w))$, and $\boldsymbol{G}(w) = (G_x(w), G_y(w))$ then

$$F_\eta(w) = T_\eta^{-1} F_x(T_\eta w), \; T_\eta \boldsymbol{B}(w) \cdot \eta = B_x(T_\eta w), \; T_\eta \boldsymbol{G}(w) \cdot \eta = G_x(T_\eta w). \tag{3}$$

Moreover, it is easy to check that $T_\eta w$ verifies the system

$$\partial_t(T_\eta w) + \partial_\eta F_x(T_\eta w) + B_x(T_\eta w)\partial_\eta w = G_x(T_\eta w)\partial_\eta H - R_{\eta^\perp}, \tag{4}$$

where $R_{\eta^\perp} = T_\eta \Big(\partial_{\eta^\perp} F_{\eta^\perp}(w) + \boldsymbol{B}(w) \cdot \eta^\perp \partial_{\eta^\perp} w - \boldsymbol{G}(w) \cdot \eta^\perp \partial_{\eta^\perp} H \Big)$.

Finally, let us remark that there is no explicit formula for the eigenvalues and eigenvectors of matrix $A_\eta(w) = A_x(w)\eta_x + A_y(w)\eta_y$, but it is well known in oceanography that the fastest and the slowest wave of system (1) could be approximated by the expressions

$$\lambda_L^\eta = \bar{u}_\eta - \sqrt{g \sum_{l=1}^m h_l}, \quad \lambda_R^\eta = \bar{u}_\eta + \sqrt{g \sum_{l=1}^m h_l} \tag{5}$$

where $\bar{u}_\eta$ is defined by

$$\bar{u}_\eta = \frac{\sum_{l=1}^m \boldsymbol{u}_l \cdot \eta \, h_l}{\sum_{l=1}^m h_l}. \tag{6}$$

3. Numerical Scheme

To discretize (1) the computational domain D is decomposed into subsets with a simple geometry, called cells or finite volumes: $V_i \subset \mathbb{R}^2$. Here, it is assumed that the cells are rectangular with edges parallel to the Cartesian axes. Let us denote by T the mesh 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 (Δx (respectively Δy) the length of the horizontal (respectively vertical) edges); $\eta_{ij} = (\eta_{ij,x}, \eta_{ij,y})$ the normal unit vector at edge E_{ij} pointing towards the cell V_j and w_i^n the approximation to the average of the solution in the cell V_i at time t^n provided by the numerical scheme:

$$w_i^n \cong \frac{1}{|V_i|} \int_{V_i} w(\boldsymbol{x}, t^n) \, d\boldsymbol{x}.$$

Let us now briefly describe the numerical scheme that we use to approximate the solution of the system (1). Here, we use the natural extension to 2D problems of the PVM-2U scheme introduced in [8] :

$$w_i^{n+1} = w_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \mathcal{N}_i} |E_{ij}| \mathcal{F}_{ij}^- \tag{7}$$

where $\mathcal{F}_{ij}^-$ is defined taking into account the property of invariance by rotations of system (1). The following steps are performed to define $\mathcal{F}_{ij}^-$:

1. We define

$$w^{\eta_{ij}} = \left(h_1,\ q_1^{\eta_{ij}},\ h_2,\ q_2^{\eta_{ij}}, \cdots,\ h_m,\ q_{m,}^{\eta_{ij}}\right)^T = T_{\eta_{ij}}(w)_{[1,2,4,5\cdots,3m-2,3m-1]},$$

and

$$w^{\eta_{ij}^{\perp}} = \left(q_1^{\eta_{ij}^{\perp}},\ q_2^{\eta_{ij}^{\perp}}, \cdots,\ q_m^{\eta_{ij}^{\perp}}\right)^T = T_{\eta_{ij}}(w)_{[3,6,\cdots,3m]},$$

where $w_{[i_1,\cdots,i_s]}$ is the vector defined from vector w, using its i_1-th, $\dots$, i_s-th components, $q_l^{\eta_{ij}} = \boldsymbol{q}_l \cdot \eta_{ij}$ and $q_l^{\eta_{ij}^{\perp}} = \boldsymbol{q}_l \cdot \eta_{ij}^{\perp}$, $l = 1, \cdots, m$.

2. Let $\Phi_{\eta_{ij}}^{-}$ be the 1D numerical PVM-2U flux associated to the 1D multilayer shallow-water system defined using the 1-st, 2-nd, 4-th and 5-th, $\cdots$, $(3m-2)$-th and $(3m-1)$-th equations of system (4) where the term $R_{\eta_{ij}^{\perp}}$ has been neglected:

$$\begin{aligned}\Phi_{\eta_{ij}}^{-} &= \frac{1}{2}(\mathcal{F}(w_j^{\eta_{ij}}) - \mathcal{F}(w_i^{\eta_{ij}}) + \mathcal{B}_{ij}(w_j^{\eta_{ij}} - w_i^{\eta_{ij}}) - \mathcal{G}_{ij}(H_j - H_i) \quad (8)\\ &\quad -\mathcal{Q}_{ij}((w_j^{\eta_{ij}} - w_i^{\eta_{ij}}) - (\mathcal{A}_{ij}^{*})^{-1}\mathcal{G}_{ij}(H_j - H_i)) + \mathcal{F}_C(w_i^{\eta_{ij}}) \quad (9)\end{aligned}$$

where

$$\mathcal{F}(w^{\eta_{ij}}) = \begin{pmatrix} u_1^{\eta_{ij}} h_1 \\ (u_1^{\eta_{ij}})^2 h_1 + \frac{g}{2} h_1^2 \\ u_2^{\eta_{ij}} h_2 \\ (u_2^{\eta_{ij}})^2 h_2 + \frac{g}{2} h_2^2 \\ \vdots \\ u_m^{\eta_{ij}} h_m \\ (u_m^{\eta_{ij}})^2 h_m + \frac{g}{2} h_m^2 \end{pmatrix},\ \mathcal{G}_{ij} = \begin{pmatrix} 0 \\ g h_1^{ij} \\ 0 \\ g h_2^{ij} \\ \vdots \\ 0 \\ g h_m^{ij} \end{pmatrix},\ \mathcal{F}_C(w_{\eta_{ij}}) = \begin{pmatrix} u_1^{\eta_{ij}} h_1 \\ (u_1^{\eta_{ij}})^2 h_1 \\ u_2^{\eta_{ij}} h_2 \\ (u_2^{\eta_{ij}})^2 h_2 \\ \vdots \\ u_m^{\eta_{ij}} h_m \\ (u_m^{\eta_{ij}})^2 h_m \end{pmatrix},$$

$$\mathcal{B}_{ij}(w_j^{\eta_{ij}} - w_i^{\eta_{ij}}) = \begin{pmatrix} 0 \\ g h_1^{ij} \left(\sum_{k=2}^{m} (h_{k,j} - h_{k,i}) \right) \\ 0 \\ g h_2^{ij} \left(\sum_{k=3}^{m} (h_{k,j} - h_{k,i}) + \sum_{k=1}^{1} \frac{\rho_k}{\rho_2} (h_{k,j} - h_{k,i}) \right) \\ \vdots \\ 0 \\ g h_m^{ij} \left(\sum_{k=1}^{m-1} \frac{\rho_k}{\rho_m} (h_{k,j} - h_{k,i}) \right) \end{pmatrix}$$

and

$$h_l^{ij} = \frac{h_{l,j} + h_{l,i}}{2}, \quad l = 1, \cdots, m.$$

Matrix $\mathcal{Q}_{ij}$ is defined by

$$\mathcal{Q}_{ij} = \alpha_0^{ij} Id + \alpha_1^{ij} \mathcal{A}_{ij} + \alpha_2^{ij} \mathcal{A}_{ij}^2 \qquad (10)$$

with

$$\begin{aligned}
\alpha_0^{ij} &= \frac{(S_M)^2 S_m(\text{sign}(S_m) - \text{sign}(S_M))}{(S_m - S_M)^2}, \\
\alpha_1^{ij} &= \frac{S_M(|S_M| - |S_m|) + S_m(\text{sign}(S_M)S_m - S_M\text{sign}(S_m))}{(S_m - S_M)^2}, \\
\alpha_2^{ij} &= \frac{S_m(\text{sign}(S_m) - \text{sign}(S_M))}{(S_m - S_M)^2},
\end{aligned} \tag{11}$$

where

$$S_M = \begin{cases} S_L^{ij} & \text{if } |S_L^{ij}| \geq |S_R^{ij}|, \\ S_R^{ij} & \text{otherwise} \end{cases} \tag{12}$$

and

$$S_m = \begin{cases} S_R^{ij} & \text{if } |S_L^{ij}| \geq |S_R^{ij}|, \\ S_L^{ij} & \text{otherwise.} \end{cases} \tag{13}$$

S_R^{ij} and S_L^{ij} are two estimations of the fastest and slowest waves, respectively, related to the Riemann problem associated to edge E_{ij}. Here, the following estimations are used:

$$S_R^{ij} = \max\left(\lambda_{R,j}^{\eta_{ij}}\ ,\ \lambda_R^{ij}\right), \quad S_L^{ij} = \min\left(\lambda_{L,i}^{\eta_{ij}}\ ,\ \lambda_L^{ij}\right),$$

being

$$\lambda_{R,j}^{\eta_{ij}} = u_j^{\eta_{ij}} + \sqrt{g\sum_{l=1}^{m} h_{l,j}}, \quad \lambda_R^{ij} = u_{ij}^{\eta_{ij}} + \sqrt{g\sum_{l=1}^{m} h_l^{ij}}$$

$$\lambda_{L,i}^{\eta_{ij}} = u_i^{\eta_{ij}} - \sqrt{g\sum_{l=1}^{m} h_{l,i}}, \quad \lambda_L^{ij} = u_{ij}^{\eta_{ij}} - \sqrt{g\sum_{l=1}^{m} h_l^{ij}}$$

where

$$u_k^{\eta_{ij}} = \frac{\sum_{l=1}^{m} h_{l,k}\, \boldsymbol{u}_{l,k} \cdot \eta_{ij}}{\sum_{l=1}^{m} h_{l,k}}, \quad k = i, j$$

and

$$u_{ij}^{\eta_{ij}} = \frac{\sum_{l=1}^{m} u_l^{ij} h_l^{ij}}{\sum_{l=1}^{m} h_l^{ij}},$$

with

$$u_l^{ij} = \frac{\sqrt{h_{l,j}}\,u_{l,j}^{\eta_{ij}} + \sqrt{h_{l,i}}\,u_{l,i}^{\eta_{ij}}}{\sqrt{h_{l,j}} + \sqrt{h_{l,i}}}, \quad l = 1, \cdots, m \text{ and } u_{l,k}^{\eta_{ij}} = \boldsymbol{u}_{l,k} \cdot \eta_{ij},\ k = i, j.$$

$\mathcal{A}_{ij}$ is the $2m \times 2m$ matrix defined by

$$\mathcal{A}_{ij} = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ gh_1^{ij} - (u_1^{ij})^2 & 2u_1^{ij} & gh_1^{ij} & 0 & \cdots & gh_1^{ij} & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \frac{\rho_1}{\rho_2} gh_2^{ij} & 0 & gh_2^{ij} - (u_2^{ij})^2 & 2u_2^{ij} & \cdots & gh_2^{ij} & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ \frac{\rho_1}{\rho_m} gh_m^{ij} & 0 & \frac{\rho_2}{\rho_m} gh_m^{ij} & 0 & \cdots & gh_m^{ij} - (u_m^{ij})^2 & 2u_m^{ij} \end{pmatrix} \tag{14}$$

and $\mathcal{A}_{ij}^*$ is defined from $\mathcal{A}_{ij}$ by setting $u_l^{ij} = 0$, $l = 1, \cdots, m$.
Taking into account the expression of $\mathcal{Q}_{ij}$ (10) and $\mathcal{A}_{ij}^*$, $\Phi_{\eta_{ij}}^-$ can be written as

$$\Phi_{\eta_{ij}}^- = \frac{1}{2}(\mathcal{E}_{ij} - (\alpha_0^{ij} \widetilde{\mathcal{I}}_{ij} + \alpha_1^{ij} \mathcal{E}_{ij} + \alpha_2^{ij} \mathcal{A}_{ij} \mathcal{E}_{ij})) + \mathcal{F}_C(w_i^{\eta_{ij}}) \tag{15}$$

where

$$\mathcal{E}_{ij} = \mathcal{F}(w_j^{\eta_{ij}}) - \mathcal{F}(w_i^{\eta_{ij}}) + \mathcal{B}_{ij}(w_j^{\eta_{ij}} - w_i^{\eta_{ij}}) - \mathcal{G}_{ij}(H_j - H_i)$$

and

$$\widetilde{\mathcal{I}_{ij}} = \begin{pmatrix} f_j^1 - f_i^1 - (f_j^2 - f_i^2) \\ q_{1,j}^{\eta_{ij}} - q_{1,i}^{\eta_{ij}} \\ f_j^2 - f_i^2 - (f_j^3 - f_i^3) \\ q_{2,j}^{\eta_{ij}} - q_{2,i}^{\eta_{ij}} \\ \vdots \\ f_j^m - f_i^m \\ q_{m,j}^{\eta_{ij}} - q_{m,i}^{\eta_{ij}} \end{pmatrix}$$

with

$$f_\alpha^l = \sum_{k=l}^m h_{k,\alpha} - H_\alpha, \quad \alpha = i, j.$$

Finally, let us remark that $\mathcal{E}_{ij}$ can be written in an equivalent way as

$$\mathcal{E}_{ij} = \mathcal{F}_C(w_j^{\eta_{ij}}) - \mathcal{F}_C(w_i^{\eta_{ij}}) + \mathcal{P}_{ij}$$

where $\mathcal{P}_{ij}$ is a discretization of the pressure terms given by

$$\mathcal{P}_{ij} = \begin{pmatrix} 0 \\ gh_1^{ij}\left(f_j^1 - f_i^1\right) \\ 0 \\ gh_2^{ij}\left(\dfrac{\rho_1}{\rho_2}\left((f_j^1 - f_i^1) - (f_j^2 - f_i^2)\right) + (f_j^2 - f_i^2)\right) \\ \vdots \\ 0 \\ gh_m^{ij}\left(\displaystyle\sum_{k=1}^{m-1}\frac{\rho_k}{\rho_m}\left((f_j^k - f_i^k) - (f_j^{k+1} - f_i^{k+1})\right) + (f_j^m - f_i^m)\right) \end{pmatrix}.$$

Equivalently the flux corresponding to cell w_j can be defined as:

$$\Phi^+_{\eta_{ij}} = \frac{1}{2}(\mathcal{E}_{ij} + (\alpha_0^{ij}\widetilde{\mathcal{I}}_{ij} + \alpha_1^{ij}\mathcal{E}_{ij} + \alpha_2^{ij}\mathcal{A}_{ij}\mathcal{E}_{ij})) - \mathcal{F}_C(w_j^{\eta_{ij}}). \tag{16}$$

3. Let us define

$$\Phi^-_{\eta_{ij}^\perp} = \begin{pmatrix} (\Phi^-_{\eta_{ij}})_{[1]}\, u_1^{\eta_{ij}^\perp,*} \\ (\Phi^-_{\eta_{ij}})_{[3]}\, u_2^{\eta_{ij}^\perp,*} \\ \vdots \\ (\Phi^-_{\eta_{ij}})_{[2m-1]}\, u_m^{\eta_{ij}^\perp,*} \end{pmatrix}, \tag{17}$$

where $u_l^{\eta_{ij}^\perp,*}$ is defined as follows

$$u_l^{\eta_{ij}^\perp,*} = \begin{cases} \dfrac{q_{l,i}^{\eta_{ij}^\perp}}{h_{l,i}} & \text{if } (\Phi^-_{\eta_{ij}})_{[2l-1]} > 0 \\ \dfrac{q_{l,j}^{\eta_{ij}^\perp}}{h_{l,j}} & \text{otherwise} \end{cases} \qquad l = 1, \cdots, m.$$

Let us remark that $\Phi^-_{\eta_{ij}^\perp}$ is the numerical flux associated to the 1-st, 6-th, $\cdots$, $3m$-th equations of system (4) where, again, the term $R_{\eta_{ij}^\perp}$ has been neglected. Its derivation has been done following the main ideas of the HLLC method for the shallow-water system introduced in [9] as $q_l^{\eta_{ij}^\perp}$, $l = 1, \cdots, m$ can be seen as a passive scalar that is convected by the flow. Equivalently, the flux corresponding to cell w_j can be defined as

$$\Phi^+_{\eta_{ij}^\perp} = -\Phi^-_{\eta_{ij}^\perp}.$$

4. Finally, the global numerical flux is defined by $\mathcal{F}_{ij}^- = T_{\eta_{ij}}^{-1} F_{ij}^-$, where

$$F_{ij}^- = \begin{pmatrix} (\Phi_{\eta_{ij}}^-)_{[1]} \\ (\Phi_{\eta_{ij}}^-)_{[2]} \\ (\Phi_{\eta_{ij}^\perp}^-)_{[1]} \\ (\Phi_{\eta_{ij}}^-)_{[3]} \\ (\Phi_{\eta_{ij}}^-)_{[4]} \\ (\Phi_{\eta_{ij}^\perp}^-)_{[2]} \\ \vdots \\ (\Phi_{\eta_{ij}}^-)_{[2m-1]} \\ (\Phi_{\eta_{ij}}^-)_{[2m]} \\ (\Phi_{\eta_{ij}^\perp}^-)_{[m]} \end{pmatrix}.$$

Equivalently, $\mathcal{F}_{ij}^+ = T_{\eta_{ij}}^{-1} F_{ij}^+$.

A CFL condition must be imposed to ensure stability of the scheme:

$$\Delta t = \min_{i=1\ldots NV} \left\{ \left[\frac{\sum_{j\in\mathcal{N}_i} |E_{ij}|\, \lambda_{\max}^{ij}}{2\gamma\, |V_i|} \right]^{-1} \right\} \tag{18}$$

with $0 < \delta \leq 1$ and $\lambda_{\max}^{ij} = \max\left(|\lambda_R^{ij}|, |\lambda_L^{ij}|\right)$.

Let us remark that this scheme is path-conservative in the sense introduced by Pares in [17] and extended to 2D by Castro et al. in [6]. Moreover, it is exactly well-balanced for stationary solutions corresponding to water at rest. More general results concerning the consistency and well-balanced properties have been studied in [6] and [18].

4. Computational scheme

At the computational level, the domain will be treated as a 3D domain in the sense that for each volume V_i the state data related to each layer will be used and stored independently. This means that each volume V_i will be decomposed in a set of sub-volumes $\{V_{l,i}\,|\, l = 1, \cdots, m\}$ where m is the number of layers. The same logic is applied to the edges so that $E_{l,ij}$ is defined as the common edge of two neighboring volumes $V_{l,i}$ and $V_{l,j}$. This scheme is graphically shown in Figure 2. This software design decision adds a third dimension to the sources of parallelism as an important subset of the computations associated to cells and edges can be performed independently across layers.

The storage of each mathematical volume is also spread over the computational volumes. The state associated to $V_{l,i}$ is $w_{l,i}$ and the rotated state: $T_{\eta_{ij}}(w)_{[3l-2,3l-1]} = \left(h_l, q_l^{\eta_{ij}}\right)^T$ and $T_{\eta_{ij}}(w)_{[3l]} = q_l^{\eta_{ij}^\perp}$.

As shown in equation (7), the computational cost of the simulation process is mainly associated to the computation of $\mathcal{F}_{ij}^-$. This computation is split over the existing layers and $\mathcal{F}_{ij}^-$ is obtained by composing $\mathcal{F}_{l,ij}^-$ $l = 1, \cdots, m$

$$\mathcal{F}_{ij}^- = (R_{\eta_{ij}}^{l,l})^{-1} F_{l,ij}^- = \begin{pmatrix} (\Phi_{\eta_{ij}}^-)_{[2l-1]} \\ (\Phi_{\eta_{ij}}^-)_{[2l]} \\ (\Phi_{\eta_{ij}^\perp}^-)_{[l]} \end{pmatrix}$$

which allows the further parallelization of this computational step.

The whole set of operations to compute $\mathcal{F}_{l,ij}^-$ $l = 1, \cdots, m$ has been divided into several steps with two precomputation steps (Figure 3, steps 1 and 2) that are run before the effective $\mathcal{F}_{l,ij}^-$ computation. The

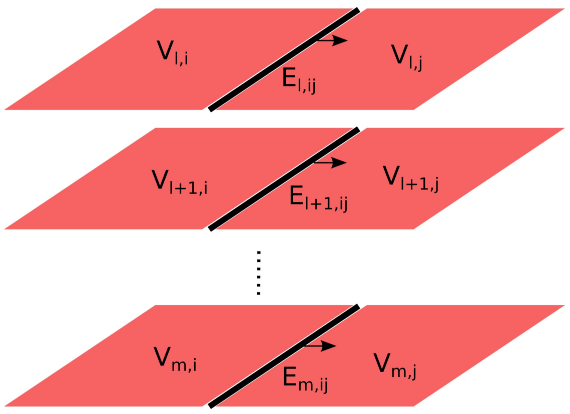

Figure 2: Finite volumes scheme

first precomputation step is associated to each volume as the computation of data is not dependent on the neighboring volume. The second and third steps are associated to each edge.

The precomputation steps increase the parallelism level of the computation of $\mathcal{F}^{-}_{l,ij}$ (Figure 3, step 3) because it moves dependencies between the layers to the precomputation steps. This allows achieving the maximum level of parallelism for the most computationally intensive part of the simulation process.

After the computation of $\mathcal{F}^{-}_{l,ij}$ and $\lambda^{ij}_{\max}$, there is another kernel, associated to each volume, used to compute the next state of each volume, but this is a lightweight computation (Figure 3, step 4).

In the following sections each computational phase is explained in detail.

4.1. Volume Precomputations

The first kernel that is run at each simulation step performs the computation of f^l_i for each volume $V_{l,i}$.

$$f^l_i = \sum_{k=l}^{m} h_{k,i} - H_i$$

This computation is carried out layer-wise starting from the deepest layer m because this allows to reuse the computations of layer $l+1$ for layer l ($f^l_i = f^{l+1}_i + h_{l,i}$). The values f^l_i are stored in a float array and are used for the pressure terms matrix $\mathcal{P}_{ij}$ computation in the next precomputation step. The f^l_i values are also required to compute matrix $\widetilde{\mathcal{I}_{ij}}$.

In the same kernel the local contribution of the volume to the following expressions

$$\sum_{l=1}^{m} h_l$$

and

$$\sum_{l=1}^{m} \boldsymbol{u}_l \cdot \eta\, h_l$$

is computed and accumulated in a float2 array. These values are used for the computation of $u^{\eta_{ij}}_i$ which is required for the computation of S^{ij}_R, S^{ij}_L, $\lambda^{\eta_{ij}}_{L,i}$ and $\lambda^{\eta_{ij}}_{R,j}$ (and $\lambda^{ij}_{\max}$ in the third step).

4.2. Edge Precomputations

In this stage, for each edge $E_{l,ij}$ data related to the 1D Roe state is precomputed. For each edge:

$$h^{ij}_l = \frac{h_{l,j} + h_{l,i}}{2}$$

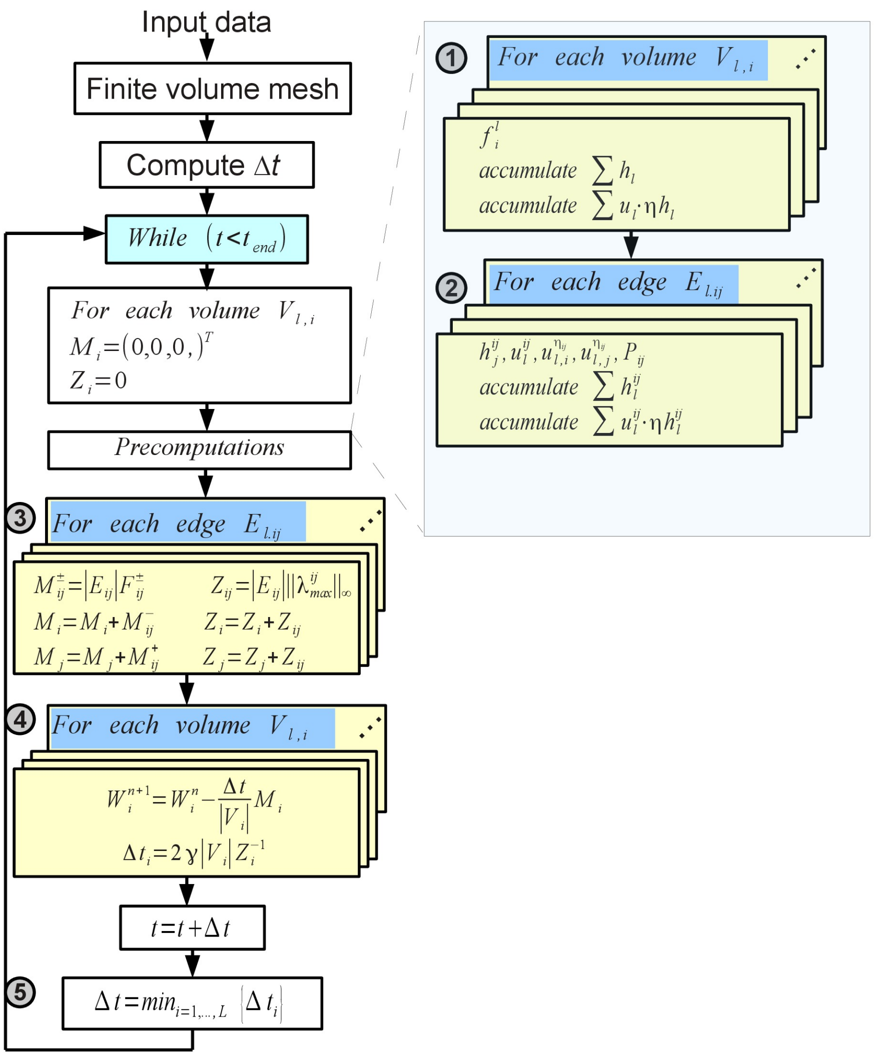

Figure 3: Block diagram

$$u_l^{ij} = \frac{\sqrt{h_{l,j}}\, u_{l,j}^{\eta_{ij}} + \sqrt{h_{l,i}}\, u_{l,i}^{\eta_{ij}}}{\sqrt{h_{l,j}} + \sqrt{h_{l,i}}}$$

are computed. The h_l^{ij}, u_l^{ij}, $u_{l,i}^{\eta_{ij}}$ and $u_{l,j}^{\eta_{ij}}$ values are stored in a float4 array for the later computation of matrix $\mathcal{A}_{ij}$ and $\mathcal{F}_C(w_i^{\eta_{ij}})$.

In the same kernel, the local contribution of the edge to the following expressions:

$$\sum_{l=1}^{m} h_l^{ij}$$

and

$$\sum_{l=1}^{m} u_l^{ij} h_l^{ij}$$

are computed and accumulated in a float2 array. These values are required for the computation of $u_{ij}^{\eta_{ij}}$

which is required for the computation of S_R^{ij}, S_L^{ij} λ_L^{ij}, λ_R^{ij} (and $\lambda_{\max}^{ij}$ in the third step).

Finally, the pressure terms $\mathcal{P}_{ij}$ are also computed in this kernel using the f_i^l values precomputed for each volume in the previous step. Pressure terms are stored in a float2 array. Each kernel instance performs only the computation involving row l of $\mathcal{P}_{ij}$ (rows $2l-1$ and $2l$), which means the computation of the matrix is split over the kernels instances associated to the edge set.

4.3. $\mathcal{F}_{ij}^-$ computation

In the third step, the kernel instance associated to each edge $E_{l,ij}$ produces a partial result of the term $\sum_{j\in\mathcal{N}_i}|E_{ij}|\mathcal{F}_{ij}^-$ required to compute the next state of each volume. Each partial result is stored in an accumulator. The reason for using only one accumulator M_i is to be able to control the positivity of the partial result as the computations of the final result are carried out and also to reduce the storage requirements of the simulation process. In order to avoid race conditions when accessing each accumulator, edges are processed in sets:

- even vertical edges
- odd vertical edges
- even horizontal edges
- odd horizontal edges

As shown in Figure 4, the computation associated to four edges contributes to the computation of

$$\sum_{j\in\mathcal{N}_i}|E_{ij}|\mathcal{F}_{ij}^- = |E_{ia}|\mathcal{F}_{ia}^- + |E_{id}|\mathcal{F}_{id}^- + |E_{bi}|\mathcal{F}_{bi}^+ + |E_{ci}|\mathcal{F}_{ci}^+$$

where each edge belongs to a different edge set of the before mentioned edge classification scheme.

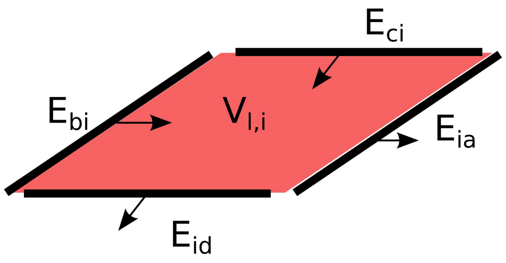

Figure 4: Edges that contribute to the state computation of a volume

The computation of $\mathcal{F}_{ij}^-$, and therefore $\Phi_{\eta_{ij}}^-$ requires the computation of several matrices: $\mathcal{F}_C(w_i^{\eta_{ij}})$, $\mathcal{F}_C(w_j^{\eta_{ij}})$, $\widetilde{\mathcal{I}}_{ij}$, $\mathcal{E}_{ij}$ and $\mathcal{A}_{ij}$.

It is important to note that each kernel instance performs only the computations related to layer l which are two rows of each of the before mentioned matrices. This way the whole computation is distributed across the kernel instances of the edges of all layers. The kernel instance associated to edge $E_{l,ij}$ computes only 3 components of $\mathcal{F}_{ij}^-$.

The same kernel also computes $\lambda_{\max}^{ij}$ (stores the partial results in Z_i) to allow the computation of Δt^n which requires values already precomputed in the two precomputation kernels.

Finally, $\mathcal{F}_{ij}^+$ is computed at the same time at each edge. This way the same kernel instance contributes to the computation of $w_i^{l,n+1}$ and $w_j^{l,n+1}$, and the new states of two volumes can be obtained requiring almost the same computational effort as for one of them.

All the computation steps of this phase can be performed in parallel as there are no dependencies between layers. Edge related kernels are therefore organized in 3D thread blocks where the z coordinate of the thread block dimension is used to represent the layer of the kernel instance.

4.4. Next state computation

After $\mathcal{F}^{-}_{ij}$ and $\lambda^{ij}_{\max}$ have been computed, a reduction operation is required to compute the minimum of the local result of $\lambda^{ij}_{\max}$ to obtain Δt^{n+1}. This minimum is computed with a dedicated kernel.

Finally, using the accumulator associated to each volume (M_i), the next state can obtained using Δt^n (value of the previous simulation step). A dedicated kernel per volume $V_{l,i}$, using the previous state w^n_i computes the next state w^{n+1}_i layer-wise.

Both the reduction and state update kernels are very lightweight and account only for a small portion of the total computing time of each simulation step.

The next state computations could be integrated in the kernel that processes each edge, after processing the odd horizontal edges. This would make the last kernel unnecessary but would increase the number of GPU registers required by the edge processing kernel (from 42 to 52 registers). This effect reduces occupancy and increases the simulation time. Therefore, this option was discarded.

4.5. Volume reordering

The restriction imposed of alternatively processing even and odd edges creates non fully coalesced memory access patterns when vertical edges are processed. An example of this fact in shown in Figure 5A. Considering, for instance, the processing step of even vertical edges, and taking into account the left and right volume data of each edge has to be accessed, first memory read instructions for volumes $1, 3, 5\ldots$ will be issued and then memory read instructions for volumes $0, 2, 4, \ldots$. As volumes are stored in consecutive memory positions, these access patterns have gaps between each element pair. Figure 5 uses horizontal arrows to mark the volumes accessed by the kernel associated to each edge and a number indicating in which sequence these accesses are issued.

By reordering the volumes in a such a way that all even volumes are stored consecutively and then all odd volumes are also stored consecutively, volumes $1, 3, 5\ldots$ can be accessed in a fully coalesced way and also volumes $0, 2, 4, \ldots$ (see Figure 5B). This Figure shows that the first memory access instruction will fetch all the volume data of the right part of the grid and the second memory access instruction the left one.

This volume reordering scheme does also make memory writes coalesced. Each kernel associated to each edge writes to the accumulator of each volume using the same pattern as the read operations which means writes do benefit from the reordering scheme in the same way.

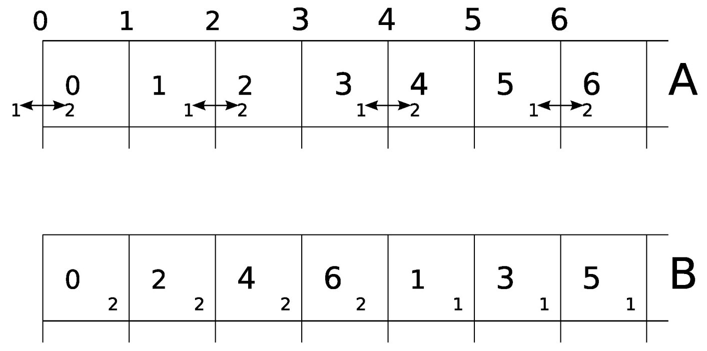

Figure 5: Volume reordering scheme

When horizontal edges are processed, as each type of edge is in the same row, no gaps are generated when accessing the upper and the lower volumes associated to each edge. This means volumes are not reordered vertically but only horizontally.

5. Numerical Experiments

Several experiments have been carried out in order to test the efficiency of the proposed simulation software design. First, a comparison with an existing two-layer simulation software was performed with

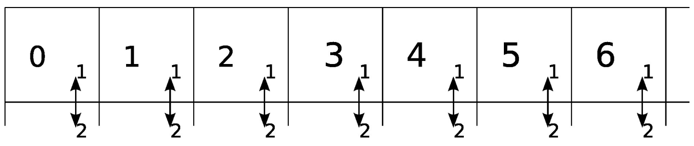

Figure 6: Horizontal edges

both its GPU and its CPU version. Additionally, the multilayer simulation software was run using different mesh sizes and different number of layers. The GPU used for the experiments was an Nvidia GTX680 which uses the Kepler [16] architecture.

5.1. Comparison with a two-layer system

In order to study the efficiency of the multilayer system design, a comparison with the two-layer GPU system [1, 3] was performed, but using the PVM-2U scheme instead of the Roe scheme, a structured regular mesh and a single GPU . Three experiments were carried out using different mesh sizes. Each experiment simulates a dam break on the internal layer during 10 seconds of simulation time.

The test scenario is similar to the one shown in Figure 7 but with only two layers. The domain consisted in a 10×10 square and a flat basin, $H(\boldsymbol{x}) = 1, \forall \boldsymbol{x}$. The initial conditions where defined by

- The dam height is 4.5 and the distance to the top neighboring layer is 0.5. The region of the second layer not affected by the presence of the simulated dam has a thickness value of 1.
- $\boldsymbol{q}_l(\boldsymbol{x}, 0) = \boldsymbol{0}, l = 1, 2$

Therefore, the function that sets the thickness of each layer has the following analytical expression:

$$h_l(\boldsymbol{x}) = \begin{cases} (0.5\,\boldsymbol{DA}(\boldsymbol{x})) + 5\,(1 - \boldsymbol{DA}(\boldsymbol{x})) & \text{if } l = 1 \\ (5.5\,\boldsymbol{DA}(\boldsymbol{x})) + (1 - \boldsymbol{DA}(\boldsymbol{x})) & \text{if } l = 2 \end{cases} \tag{19}$$

where $\boldsymbol{DA}(\boldsymbol{x})$ is a function with the following expression:

$$\boldsymbol{DA}(\boldsymbol{x}) = \begin{cases} 1 & \text{if distance}(\boldsymbol{x}, \boldsymbol{c}) < 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

where $\boldsymbol{c} = (5, 5)$ is the 2D position of the center of the circular dam.

The CFL value used was 0.9 and $\rho_2 = 2\,\rho_1$. Wall boundary conditions were imposed.

As the two-layer code was specifically designed to handle two layers, it was expected to perform better than the general multilayer version, but this was not the case as it can be seen in Table 1.

#Volumes	Two-layer code	Multilayer code
256×256	7.2s	5.8s
512×512	53.1s	40.8s
1024×1024	417.2s	311.5s

Table 1: Two-layer and multilayer comparison. Time expressed in seconds

The two-layer code does also split the computations using the aforementioned edge groups, but requires no precomputation steps as each kernel instance processes both layers when processing volume and edges. This fact makes the design more monolithic and the kernel associated to the computations for each edge requires a higher amount of registers (63) than in the multilayer code. This reduces the occupancy and explains why the multilayer system offers a better performance despite the overhead produced by the adaptation to an arbitrary number of layers. In the multilayer system the bigger kernel is the one associated to the computation of $\mathcal{F}^-_{l,ij}$ and $\lambda^{ij}_{\max}$ which requires 42 registers. It can therefore be concluded that the strategy of a finer-grained design of the multilayer system offers a better performance for this type of problem as it increases the thread level parallelism while achieving a sufficient instruction level parallelism for each kernel instance. This increased performance comes at the cost of a larger memory footprint because of the intermediate data storage requirements.

5.1.1. Comparison with a CPU two-layer system

The multilayer code was also compared to a native CPU two-layer implementation. The same 10 seconds of simulation of the dam break mentioned in the previous section with 256 × 256 volumes, requires 1489 seconds on the CPU (Intel Core i7-2600 CPU @ 3.40GHz). Using four OpenMP threads, the run time is 412 seconds. This results in a speed-up of over 250× with respect to the single-threaded CPU version and over 70× with respect to the multithreaded CPU version. This CPU version includes no special optimizations apart from the OpenMP parallelization and the compiler optimizations produced by the -O3 compiler flag.

5.2. Multilayer experiments

Two experiments were performed to test the performance of our work. The first experiment used a domain consisting in a 10×10 square. The basin depth is defined by the function: $H(\boldsymbol{x}) = 1 - 0.5\ \exp\left(-10\,(x-5)^2\right)$ and all layers have a thickness value of 1, except for three layers:

- The two layers affected by the presence of the simulated dam. The dam height is 4.5 and the distance to the top neighboring layer is 0.5.
- The height of layer m is defined by $H(\boldsymbol{x})$.

The function that defines the thickness of each layer is shown in equation 21.

$$h_l(\boldsymbol{x}) = \begin{cases} H(\boldsymbol{x}) & \text{if } l = m \\ 1 & \text{if } l = 1, \ldots, (d-2), (d+1), \ldots, (m-1) \\ (5.5\,\boldsymbol{DA}(\boldsymbol{x})) + (1 - \boldsymbol{DA}(\boldsymbol{x})) & \text{if } l = d \\ (0.5\,\boldsymbol{DA}(\boldsymbol{x})) + 5\,(1 - \boldsymbol{DA}(\boldsymbol{x})) & \text{if } l = (d-1) \end{cases} \tag{21}$$

where $d = \frac{m}{2}$ is the layer on which the dam is defined. In this experiment the center of the circular dam is located at $\boldsymbol{c} = (2, 5)$.

Regarding the initial conditions, $\boldsymbol{q}_l(\boldsymbol{x}, 0) = \boldsymbol{0}, l = 1, \ldots m$. The CFL value used was 0.9 and $\rho_l = 2\,\rho_{l-1}$. Wall boundary conditions were imposed.

The test scenario scheme with six layers and basin is shown in Figure 7. There are also video captures of different simulation scenarios available on the complementary material website: `http://dicits.ugr.es/software/MultiLayerSW/`

In Table 2 the results for different number of layers and 2D sizes are shown. The number of volumes (millions of) processed per second is also shown considering the number of simulation iterations performed, and the total number of computational volumes.

In the first test, the total thickness of the water increases as layers are added, and so does the density difference between the first and the last layer. This results in a decreasing Δt value and therefore an increasing number of iterations as the number of layers increases.

For the second experiment, the total thickness of the water was set to 10. This means that as layers are added, the thickness of each layer decreases. Additionally the density values were set so that $\rho_1 = 0.1$ and $\rho_m = 1.0$ and all the intermediate ρ_i values are interpolated lineally. The rest of the experiment parameters are the same as for the previous one. This test scenario produces a fixed Δt value at each iteration step and a constant number of iterations. In this case, as layers as added, the simulation scenario remains fixed but more resolution is added to the dimension represented by the number of layers.

The simulation times for the second experiment are shown for a 512 × 512 domain together with the results of the previous experiment in Figure 8. This plot shows that when keeping the number of iterations

#Volumes per layer	#Layers	Time	#Vols/s (millions)
256 × 256	4	12.7s	110
256 × 256	8	34.1s	100
256 × 256	16	112.7s	79
512 × 512	4	93.7s	121
512 × 512	8	254.5s	109
512 × 512	16	865.6s	84
1024 × 1024	4	732.7s	125
1024 × 1024	8	1981.6s	113
1024 × 1024	16	6848.1s	86

Table 2: GTX680 - Multilayer experiments. Time expressed in seconds

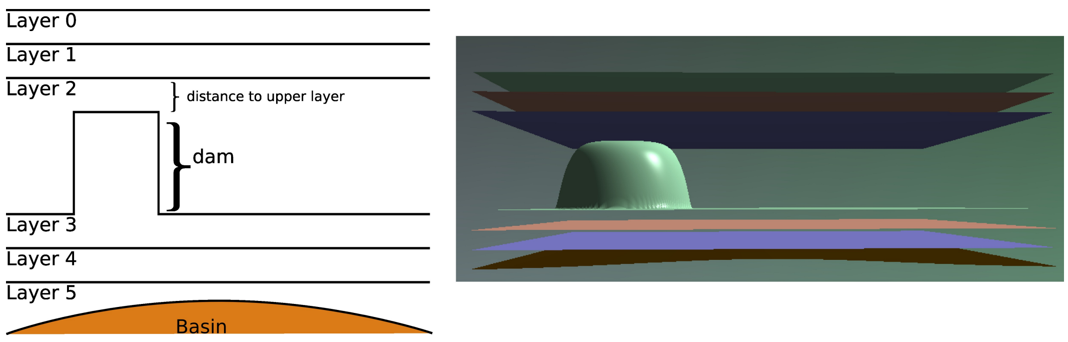

Figure 7: Test scenario with six layers

constant, doubling the number of layers does only increase the run time by a factor between 2 and 3. This is a very good result taking into account the inter-layer dependencies become more important as the number of layers increases.

A comparison using a Tesla K20 GPU (2495 Cuda cores @ 706Mhz) was also performed. This GPU offers a greater amount of Cuda cores (60% more) than the GTX 680 (1536 Cuda cores @ 1058Mhz) but at a lower clock speed (33% less). Table 3 shows the results obtained. Finally, a single experiment was performed using a GTX Titan GPU (2688 Cuda cores @ 876Mhz). This GPU provides 75% more cores than the GTX680 at a 15% lower clock rate and the results obtained are shown in table 4

#Volumes per layer	#Layers	GTX 680 time	Tesla K20 time
256 × 256	8	34,1s	29s
512 × 512	16	865.6s	802s

Table 3: GTX 680-Tesla K20 multilayer experiment. Time expressed in seconds

Taking into account the 1.5× speed-up factor obtained using the GTX Titan and the up to 1.11× factor obtained with the Tesla K20 with respect to the results of the GTX 680 model, we believe the system shows a good scalability with respect to the number of available Cuda cores. The important clock rate difference of the K20 model explains that the results obtained with this GPU are worse than the ones that could be expected by only considering its number of Cuda cores.

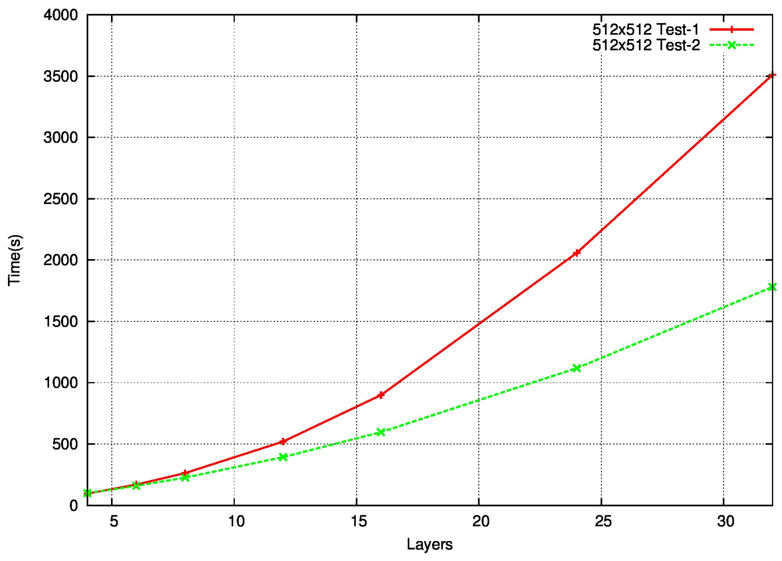

Figure 8: 512 × 512 simulation times

#Volumes per layer	#Layers	GTX 680 time	GTX Titan time
256 × 256	6	23s	15,2s

Table 4: GTX 680-GTX Titan multilayer experiments. Time expressed in seconds

5.3. Volume reordering

We have also analyzed the benefits of the volume reordering scheme with respect to the ad-hoc volume/data structure mapping using the first of the two test scenarios introduced in section 5.2.

#Layers	Reordered volumes	Ad-hoc order
2	6.1s	5.8s
6	25.3s	26.1s
12	75.6s	82.37s

Table 5: Volume reordering benefits. 65 535 (256 × 256) volumes per layer. Time expressed in seconds

Table 5 shows that the reordering scheme does produce some overhead because it requires a higher amount of operations to compute volume and edge indices and this fact produces a slightly higher run time for a low number of layers. As the number of edges increases, which also increases the number of memory accesses, the benefit of the reordering is clear.

6. Conclusions

An efficient GPU based multilayer simulation scheme has been presented which allows an arbitrary number of layers as it is fully parameterized. The fact that it is based on a 2D mathematical model that has been implemented by using a 3D approach, a higher level of thread level parallelism is achieved which offers a good scalability as the number of layers increases.

The efficiency of the proposed simulation scheme has been shown in terms of the number of volumes which are processed per second and the speed-up values with respect to previous two-layer systems, using both their GPU and their single and multi-threaded CPU version. The proposed general scheme introduces no overhead when compared to existing GPU-based systems which were specifically designed for two layers. Moreover, it achieves a run time reduction from one to two orders of magnitude when compared to multi and single-threaded CPU implementations. Our simulation software processes around 100 million of 3D volumes per second for scenarios with between 4 and 16 layers on a single GPU.

The results show that the simulation scheme that has been presented can be applied to large-scale problems to simulate real-world phenomena like lake or oceanic currents.

Acknowledgments

The work of M. J. Castro was supported in part by the Spanish Government and FEDER through the Research project MTM2012-38383-C02-01, and by the Andalusian Government through the projects P11-FQM8179 and P11-RNM7069

References

[1] M. de la Asunción, J.M. Mantas, M.J. Castro, Programming CUDA-based GPUs to simulate two-layer shallow water flows, in: P. D'ambra, M. Guarracino, D. Talia (Eds.), Euro-Par 2010, volume 6272 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 353–364.

[2] M. de la Asunción, J.M. Mantas, M.J. Castro, Simulation of one-layer shallow water systems on multicore and CUDA architectures, Journal of Supercomputing 58 (2011) 206–214.

[3] M. de la Asunción, J.M. Mantas, M.J. Castro, E.D. Fernández-Nieto, An MPI–CUDA implementation of an improved roe method for two-layer shallow water systems, Journal of Parallel and Distributed Computing. Special Issue on Accelerators for High–Performance Computing 72 (2012) 1065–1072.

[4] A.R. Brodtkorb, T.R. Hagen, K.A. Lie, J.R. Natvig, Simulation and visualization of the Saint-Venant system using GPUs, Computing and Visualization in Science 13 (2010) 341–353.

[5] A.R. Brodtkorb, M.L. Sætra, Shallow water simulations on multiple GPUs, in: PARA 2010: State of the Art in Scientific and Parallel Computing, Reykjavik (Iceland), pp. 56–66.

[6] M.J. Castro, E.D. Fernández-Nieto, A.M. Ferreiro, J.A. García, C. Parés, High order extensions of Roe schemes for two-dimensional nonconservative hyperbolic systems, Journal of Scientific Computing 39 (2009) 67–114.

[7] P. Degond, P.F. Peyrard, G. Russo, P. Villedieu, Polynomial upwind schemes for hyperbolic systems, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 328 (1999) 479 – 483. doi:http://dx.doi.org/10.1016/S0764-4442(99)80194-3.

[8] M.J.C. Díaz, E.D. Fernández-Nieto, A class of computationally fast first order finite volume solvers: PVM methods., SIAM J. Scientific Computing 34 (2012).

[9] E.D. Fernández-Nieto, D. Bresch, J. Monnier, A consistent intermediate wave speed for a well-balanced HLLC solver, Comptes Rendus Mathematique 346 (2008) 795 – 800. doi:http://dx.doi.org/10.1016/j.crma.2008.05.012.

[10] M. Geveler, D. Ribbrock, S. Mallach, D. Göddeke, A simulation suite for Lattice-Boltzmann based real-time CFD applications exploiting multi-level parallelism on modern multi- and many-core architectures, Journal of Computational Science 2 (2011) 113–123.

[11] T.R. Hagen, J.M. Hjelmervik, K.A. Lie, J.R. Natvig, M.O. Henriksen, Visual simulations of shallow-water waves, Simulation Modelling Practice and Theory 13 (2005) 716–726.

[12] Khronos OpenCL Working Group, The OpenCL Specification, Accessed November 2012.

[13] M. Lastra, J.M. Mantas, C. Ureña, M.J. Castro, J.A. García, Simulation of shallow-water systems using graphics processing units, Mathematics and Computers in Simulation 80 (2009) 598–618.

[14] NVIDIA, Cuda, http://www.nvidia.com/object/cuda_home_new.html, Accessed 2013.

[15] NVIDIA, Nvidia, http://www.nvidia.com/, Accessed 2013.

[16] NVIDIA, Kepler-The world's fastest, most efficient HPC arquitecture, http://www.nvidia.com/object/nvidia-kepler.html, Accessed August 2012.

[17] C. Parés, Numerical methods for nonconservative hyperbolic systems: A theoretical framework, SIAM Journal on Numerical Analysis 44 (2006) pp. 300–321.

[18] C. Parés, M.J. Castro, On the well-balance property of Roe's method for nonconservative hyperbolic systems. Applications to shallow-water systems, ESAIM: Mathematical Modelling and Numerical Analysis 38 (2004) 821–852.