To appear in the *Journal of Hydraulic Research*
Vol. 00, No. 00, Month 20XX, 1–20

Research paper

An efficient GPU implementation for a faster simulation of unsteady bed-load transport

Carmelo Juez, Research Scientist, *LIFTEC, CSIC-Universidad de Zaragoza, Spain*
Email: carmelo@unizar.es (author for correspondence)

Asier Lacasta, PhD Student, *LIFTEC, CSIC-Universidad de Zaragoza, Spain*
Email: alacasta@unizar.es

Javier Murillo, Research and Teaching Associate, *LIFTEC, CSIC-Universidad de Zaragoza, Spain*
Email: Javier.Murillo@unizar.es

Pilar García-Navarro (IAHR Member), Full Professor, *LIFTEC, CSIC-Universidad de Zaragoza, Spain*
Email: pigar@unizar.es

ABSTRACT
Computational tools may help engineers in the assessment of sediment transport during the decision-making processes. The main requirements are that the numerical results have to be accurate and simulation models must be fast. The present work is based on the 2D shallow water equations in combination with the 2D Exner equation. The resulting numerical model accuracy was already discussed in previous work. Regarding the speed of the computation, the Exner equation slows down the already costly 2D shallow water model as the number of variables to solve is increased and the numerical stability is more restrictive. In order to reduce the computational effort required for simulating realistic scenarios, the authors have exploited the use of Graphics Processing Units (GPUs) in combination with non-trivial optimization procedures. The gain in computing cost obtained with the graphic hardware is compared against single-core (sequential) and multi-core (parallel) CPU implementations in two unsteady cases.

Keywords: Finite Volume, flood simulation, GPU, parallel computing, sediment transport, 2D shallow water

1 Introduction

Traditionally, 1D models based on de St. Venant equations (Burguete & García-Navarro, 2001; Chang, 1982; Liu, Quin, Zhang, & Li, 2015; Petaccia et al., 2013) have been considered in hydraulic applications due to their low computational cost and data requirement. However, under the presence of complex topography or the presence of hydraulic structures, the use of 2D or 3D hydrodynamic models may be required. 2D depth averaged models are widely accepted for most practical purposes in complex cases. These models provide predictions for the water depth and the two-dimensional, depth averaged, flow velocity field at the cost of a fine topographic representation as it was pointed out in Caviedes-Voullieme, Morales-Hernandez, Lopez-Marijuan, and Garcia-Navarro (2014). The bed evolution is frequently computed through the Exner equation. The two models, hydrodynamic and morphodynamic, can be solved using asynchronous or synchronous methods (Aricò & Tucciarelli, 2008). Asynchronous techniques are based on the assumption that morphodynamic time scales are not relevant enough for altering the hydrodynamic variables within the interval of a computational time step. Therefore, the fluid mass and momentum equations are solved apart from (decoupled of) the Exner equation. Conversely, synchronous procedures assume that changes in the morphodynamic and hydrodynamic quantities take place within the same time

Received 07 Month 2015; accepted 01 Month 2016

scale, i.e. equations for both phases are solved at the same time and with the same time restriction. As stated in Juez, Murillo, and García-Navarro (2014), unsteady flows with a wide range of hydrodynamic and morphodynamic situations can only be properly tackled by means of a synchronous technique.

Focusing on the numerical techniques, the most widely used strategies are: Explicit Finite Volume (FV) schemes based on Riemann solvers (Begnudelli, Valiani, & Sanders, 2010; Canelas, Murillo, & Ferreira, 2013; Hou, Liang, Zhang, & Hinkelmann, 2015; Juez et al., 2014; Murillo & García-Navarro, 2010a; Siviglia et al., 2013; Soares-Frazao & Zech, 2010; Wu, 2004; Xia, Lin, Falconer, & Wang, 2010), or explicit Finite Element (FE) schemes (Villaret, Hervouet, Kopmann, Merkel, & Davies, 2013). The use of all these schemes for the extended system involves a higher number of algebraic operations and heavier restrictions in the stability criterion than their application to the fixed bed shallow water equations. Furthermore, the execution time required by the solver is increased when moving to realistic scenarios where large domains with high resolution meshes are required. Several authors have proposed strategies for improving their efficiency by relaxing the timestep selection (Juez et al., 2014; Serrano, Murillo, & García-Navarro, 2012) or by enlarging the CFL (Courant–Friedrichs–Lewy) condition (Murillo, García-Navarro, Brufau, & Burguete, 2008). Nevertheless, in all cases the gain in computing cost was moderate. In the search for reducing the simulation time, other authors have explored the possibility of using implicit (Bilaceri, Beux, Elmahi, Guillard, & Salvetti, 2012) or semi-implicit (Garegnani, Rosatti, & Bonaventura, 2013) methods which allow for larger time steps when comparing with explicit ones. However, the main problem is the convergence speed of the linear solver which can become the bottleneck of the simulation.

A reliable way to reduce significantly the computational effort has come in the last years through the implementation of parallelization techniques such as Multiprocessing (OpenMP) and Message Passing Interface (MPI), which allow to run simulations on cluster machines (Lacasta, García-Navarro, Burguete, & Murillo, 2013). Their drawback is the associated hardware cost and energy processor requirements which usually imply a limitation on their practical usage. Conversely, hardware accelerators, such as Graphics Processing Units (also called GPUs), emerge as a low cost strategy since they can be used on simple personal computers. It is important to emphasize that, while the computing capability of these accelerators reduces the computational effort required for large simulations, their optimal programming is not straightforward. The present paper is devoted to explain the details that should be payed attention to make the best of a GPU implementation in a sediment transport simulation model.

Previous works have developed strategies for implementing the pure shallow water equations on GPU (Kalyanapu, Siddharth, Pardyjak, Judi, & Burian, 2011; Vacondio, Dal Pal, & Mignosa, 2014). In this work, and following Lacasta, Morales-Hernández, Murillo, and García-Navarro (2014), an efficient GPU implementation for the hydrodynamic and also for the morphodynamic model is provided assuming unstructured meshes. The GPU techniques described in this paper have been incorporated into RiverFlow2D, a general purpose two-dimensional free-surface flow model as described in Garcia et al. (2015).

This work is organized as follows. In section 2 the mathematical model and the numerical scheme are described. Section 3 is devoted to outline the GPU implementation. Section 4 shows the capabilities of the tool in terms of results and speedup. Finally, in section 5, the authors draw the conclusions and propose future work.

2 Mathematical model & Numerical scheme

2.1 *Mathematical model*

The mathematical model is based on the 2D shallow water equations, SWE, and the 2D Exner equation. The SWE are derived from the Navier-Stokes equations by integrating the continuity and momentum equations over depth (Murillo & García-Navarro, 2010b). The resulting 2D system is written in conservative form as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = \mathbf{T}_\tau + \mathbf{T}_b \tag{1}$$

where the vector of conserved variables is:

$$\mathbf{U} = (h, q_x, q_y)^T \tag{2}$$

with h representing water depth, $q_x = hu$is the unit discharge in the x direction and $q_y = hv$ is the unit discharge in the y direction. The fluxes are expressed in terms of (u, v), the depth averaged components of the velocity field, as:

$$\begin{aligned}\mathbf{F} &= \left(hu, hu^2 + \frac{1}{2}gh^2, huv\right)^T \\ \mathbf{G} &= \left(hv, huv, hv^2 + \frac{1}{2}gh^2\right)^T\end{aligned} \tag{3}$$

The source terms $\mathbf{T}_\tau$ and $\mathbf{T}_b$ include, respectively, the information about the friction exerted over the bed, evaluated through the Manning formula, and the bed slopes:

$$\mathbf{T}_\tau = \left(0, -gh\frac{n^2u\sqrt{u^2+v^2}}{h^{4/3}}, -gh\frac{n^2v\sqrt{u^2+v^2}}{h^{4/3}}\right)^T \qquad \mathbf{T}_b = \left(0,\ -gh\frac{\partial z}{\partial x}, -gh\frac{\partial z}{\partial y}\right)^T \tag{4}$$

with n the Manning roughness parameter and z the bed elevation.

On the other hand, the bed evolution is modeled through the Exner equation, which is basically a movable bed continuity equation where the bed level time variations are due to the solid fluxes which cross the control volume. In this work the authors only focus on highly concentrated bed-load phenomena and, consequently, the 2D Exner equation is:

$$\frac{\partial z}{\partial t} + \xi\frac{\partial q_{s,x}}{\partial x} + \xi\frac{\partial q_{s,y}}{\partial y} = 0 \tag{5}$$

where $\xi = \frac{1}{1-p}$, p is the material porosity and $q_{s,x}$, $q_{s,y}$ are the solid fluxes. They are computed as a function of excess bed shear stress with respect to the critical value and taking into account the bed shear stress direction. This bedload transport is often expressed through the following dimensionless parameter:

$$\Phi = \frac{|\mathbf{q}_s|}{\sqrt{g(s-1)d_m^3}} \tag{6}$$

where $s = \rho_s/\rho_w$ is the ratio of solid material (ρ_s) over water (ρ_w) densities, and d_m is the grain median diameter. According to the numerical assessment performed in Juez, Murillo, and García-Navarro (2013) the empirical Smart (1984) formula is chosen for computing the dimensionless bedload discharge as follows:

$$\Phi = 4\ (d_{90}/d_{30})^{0.2}\ F\ S^{0.1}\theta^{1/2}(\theta - \theta_c^S) \tag{7}$$

where S is the velocity vector projected over the bed slope vector, as in Juez et al. (2013), for distinguishing between positive and negative sloping beds. On the other hand d_{90} and d_{30} are grain diameter values for which 90% and 30% of the weight of a nonuniform sample is finer respectively. F is the Froude number, θ is the dimensionless shear stress and θ_c^S is the critical shear stress according to Smart (1984). This formula is only applied when the shear stress is larger than the critical shear stress. Otherwise there is no sediment transport.

2.2 *Numerical scheme*

Hydrodynamic numerical scheme

System in (1) is integrated in a grid cell Ω_i and Gauss theorem is applied:

$$\frac{\partial}{\partial t}\int_{\Omega_i} \mathbf{U}\mathrm{d}\Omega + \oint_{\partial\Omega_i} \mathbf{E_n}\mathrm{dl} = \int_{\Omega_i} (\mathbf{T}_\tau + \mathbf{T}_b)\mathrm{d}\Omega \tag{8}$$

where $\mathbf{E_n} = \mathbf{F}n_x + \mathbf{G}n_y$ is the flux normal to a direction given by the outward pointing unit vector $\mathbf{n}$. Our formulation considers a piecewise representation per cell of the conserved variables, with A_i the cell area, so that:

$$\mathbf{U}_i^n = \frac{1}{A_i}\int_{\Omega_i} \mathbf{U}(x, y, t^n)\mathrm{d}\Omega \tag{9}$$

Using additionally that the second and the third integral in (8) can be explicitly expressed as a sum over the cell edges, (8) is written as:

$$A_i\frac{\partial \mathbf{U}_i}{\partial t} + \sum_{k=1}^{NE}(\mathbf{E_n})_k l_k = \sum_{k=1}^{NE}\mathbf{T}_{\tau\mathbf{n}} l_k + \sum_{k=1}^{NE}\mathbf{T}_{b\mathbf{n}} l_k \tag{10}$$

where NE is the number of edges in cell i and l_k is the edge length. On the other hand, $\mathbf{T}_{b\mathbf{n}}$ and $\mathbf{T}_{\tau\mathbf{n}}$ are suitable integrals of the bed slope and friction source terms (Murillo & García-Navarro, 2010a) projected over the outward pointing unit vector.

The numerical scheme is constructed by defining an approximate Jacobian matrix $\widetilde{\mathbf{J}}$ at each k edge between neighboring cells defined through the normal flux $\mathbf{E_n}$ so that:

$$\delta\mathbf{E}_{\mathbf{n},k} = \widetilde{\mathbf{J}}_{\mathbf{n},k}\delta\mathbf{U}_k \tag{11}$$

with $\delta\mathbf{E}_{\mathbf{n},k} = (\mathbf{E}_j - \mathbf{E}_i)\cdot\mathbf{n}_k$, $\delta\mathbf{U}_k = \mathbf{U}_j - \mathbf{U}_i$, and $\mathbf{U}_i$ and $\mathbf{U}_j$ the values at cells i and j sharing edge k.

From this approximate Jacobian matrix a set of three real eigenvalues $\widetilde{\lambda}_k^m$ and eigenvectors $\widetilde{\mathbf{e}}_k^m$ are obtained. The vector of conserved variables, $\mathbf{U}$, is then split onto the eigenvectors basis (Murillo & García-Navarro, 2010a) as:

$$\delta\mathbf{U}_k = \sum_{m=1}^{3}(\widetilde{\alpha}\widetilde{\mathbf{e}})_k^m \tag{12}$$

The source terms are also projected onto the eigenvectors basis to guarantee the exact equilibrium between fluxes and source terms (Murillo & García-Navarro, 2010a):

$$(\mathbf{T}_{b\mathbf{n}} + \mathbf{T}_{\tau\mathbf{n}})_k = \sum_{m=1}^{3} \left(\widetilde{\beta}\widetilde{\mathbf{e}}\right)_k^m \tag{13}$$

With all this previous information the volume integral in the cell at time t^{n+1} is expressed as:

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \sum_{k=1}^{NE}\sum_{m=1}^{3}(\widetilde{\lambda}^-\widetilde{\alpha} - \widetilde{\beta}^-)_k^m \widetilde{\mathbf{e}}_k^m l_k \frac{\Delta t}{A_i} \tag{14}$$

The superscript minus in (14) implies that only the incoming waves are considered for updating the flow variables of each cell, defining $\widetilde{\lambda}^- = \frac{1}{2}\left(\widetilde{\lambda} - \left|\widetilde{\lambda}\right|\right)$. Further, special care is considered when calculating wet/dry fronts. The strategy proposed is based on enforcing positive values of interface discrete water depths coming from a detailed study of the Riemann problem (Murillo & García-Navarro, 2010b; Murillo, García-Navarro, & Burguete, 2008). When they become negative, the numerical values of the friction and bed slope source terms is reduced instead of diminishing the time step.

Morphodynamic numerical scheme

Equation (5) is also integrated in a grid cell Ω_i. Using Gauss theorem:

$$\frac{\partial}{\partial t}\int_{\Omega_i} z \mathrm{d}\Omega + \oint_{\partial\Omega_i} q_{s\mathbf{n}} \mathrm{dl} = 0 \tag{15}$$

where $q_{s\mathbf{n}} = (q_{s,x}n_x + q_{s,y}n_y)$.

Assuming a piecewise representation of the variable z and that the second integral can be written as the sum of fluxes across the cell edges, the bed level is updated as:

$$z_i^{n+1} = z_i^n - \sum_{k=1}^{NE} \xi q^*_{s\mathbf{n},k} \frac{\Delta t\, l_k}{A_i} \tag{16}$$

where:

$$q^*_{s\mathbf{n},k} = \begin{cases} q_{s\mathbf{n},i} & \text{if } \widetilde{\lambda}_s > 0 \\ q_{s\mathbf{n},j} & \text{if } \widetilde{\lambda}_s < 0 \end{cases} \tag{17}$$

where $q_{s\mathbf{n},i}$ and $q_{s\mathbf{n},j}$ are the bed load discharge computed at the neighboring cells i, j, and $\widetilde{\lambda}_s$ is the numerical bed celerity estimated as:

$$\widetilde{\lambda}_s = \frac{\delta q_{s\mathbf{n},k}}{\delta z_k} \tag{18}$$

with $\delta q_{s\mathbf{n},k} = q_{s\mathbf{n},j} - q_{s\mathbf{n},i}$ and $\delta(z_k) = z_j - z_i$.

Stability criteria

As it was stated in Leveque (2002) the explicitly updated conserved variables are defined through the fluxes obtained within each cell, so, the computational time step has to be chosen small enough for ensuring a stability region. Traditionally, the numerical stability has been controlled through a dimensionless parameter, CFL,

$$\Delta t = \text{CFL}\,\frac{\min(\chi)}{\max|\widetilde{\lambda}^m|} \qquad \text{with} \qquad \text{CFL} \leq 0.5 \tag{19}$$

where χ is a relevant distance between neighboring cells (Murillo & García-Navarro, 2010b) and $\widetilde{\lambda}^m$ are the hydrodynamic celerities. The stability criterion is revisited for including a discrete estimation of the bed celerity, $\widetilde{\lambda}_s$, as in Juez et al. (2014),

$$\Delta t = \text{CFL}\,\frac{\min(\chi)}{\max|\widetilde{\lambda}^m, \widetilde{\lambda}_s|} \qquad \text{with} \qquad \text{CFL} \leq 0.5 \tag{20}$$

With this numerical strategy, the stability condition takes into consideration the most restrictive numerical wave speed coming from the hydrodynamical and morphodynamical solvers. The resulting global time step is used for updating the whole set of conserved hydrodynamic and morphological variables in the system of equations.

3 GPU implementation

Due to the large computational effort required to solve this kind of problems, a GPU based solution is presented. In particular, the proposed numerical scheme has been implemented using the NVIDIA CUDA Toolkit.

3.1 *NVIDIA CUDA & GPU Architecture*

The GPU devices were originally designed to perform operations related to computer graphics. Those operations are usually run on a mesh-based structure. With the improvement of the GPUs technology a more general approach to exploit their capabilities has been designed. This approach is commonly known as GPGPU (General Purpose computing on Graphics Processing Units) and it is the natural extension of the graphical oriented instruction set architecture (ISA) to a more generic range of applicability. It allows users to write code that can be run on the GPU hardware using high level language. On the other hand, as double-precision floating-point units are sometimes necessary in computing operations, this feature opens a new opportunity to increase the performance of numerical implementations that require that precision.
There are two main manufacturers in the field of graphical accelerators: AMD and NVIDIA. In the case of NVIDIA, their contribution to the improvement of the GPGPU paradigm has resulted in the creation of the Compute Unified Device Architecture (better known as CUDA) toolkit (NVIDIA Corporation, 2007, 2014). CUDA toolkit is a parallel framework for graphic processing which implements a set of instructions for their use in parallel codes in C. It has the disadvantage of being designed only for NVIDIA GPUs. Other more general implementations have been developed through open-source platforms such as OpenCL (Munshi et al., 2009). OpenCL has the main advantage of being hardware-independent. It is designed to enable the same implementation on a variety of computer architectures from CPU, to GPU or FPGA. Hence, the same code can be executed on both NVIDIA and AMD GPUs, which provides a high portability character to those elements developed under that framework. Nevertheless, some comparisons such as the one

proposed in Danalis et al. (2010) have demonstrated that CUDA is generally more efficient than OpenCL when using NVIDIA GPUs. Special mention requires the work described in Gandham, Medina, and Warburton (2014) where the implementation of a discontinuous Galerkin method to solve the Shallow Water equations is analyzed using CUDA and OpenCL, reaching the same conclusion as in Danalis et al. (2010). For this reason this work is based on the CUDA toolkit.

3.2 *Scheme of the implementation*

GPUs were originally oriented to perform arithmetical operations on vector-based information. Because of this design, the numerical scheme presented in this work is suitable for being implemented on GPU.

Unlike the conventional CPU implementations, the GPU solution must be designed taking into account the fact that the GPU is an independent device with its own RAM memory. This means that it is necessary to transfer those elements that may be used by the GPU from the CPU and vice versa. Although the last CUDA version makes these steps transparent to the developer by means of their *unified memory* (NVIDIA Corporation, 2014), the most common way of performing these operations is by means of explicit memory copy operations in the code. In any case, if the algorithm requires a large number of transfers, the performance of the GPU solution may be dramatically reduced due to this separate memory space.

In Fig. 1 the sequence diagram of the simulator is displayed. Except for the preprocess stage made on CPU and then its transfer to the GPU, the rest of the process is controlled by the CPU but computed on the GPU. In other words, the execution flow is controlled by the CPU and only the current time t is required by the CPU to know when the calculation has reached the target simulation time. To obtain that, it is necessary to transfer that information to the GPU at each time-step. The cost of this transfer is considerably smaller than the cost of each kernel, and it does not introduce important overheads. Moreover, in order to dump intermediate states of the simulation, the CPU may require the transfer of variables from the GPU. This transfer is heavier than the one related to the current time because of the number of elements to be copied. While the transfer of t is `sizeof(double)` bytes long, the whole domain has a total length of $N_{cells} \times$ `sizeof(double)` bytes. Memory transfer and disk writing may occupy less than 1% of the time consumed by the whole time step so it is negligible in practical situations that require a large number of time steps to complete the whole simulation.

The implementation of the numerical kernels has been made following Lacasta, Juez, Murillo, and García-Navarro (2015); Lacasta et al. (2014), where a deep analysis of these kind of solvers with unstructured meshes is provided. Briefly, the strategies proposed for obtaining an efficient implementation on GPU with unstructured meshes are the following:

- The variables as well as the rest of the information related to the wall and cell fluxes are mapped using *Structure of Arrays*. Hence, each variable is defined on a vector of size N_{cells} or N_{edges}. It provides a useful manner to access each element by each thread easily.
- The computational mesh is reordered during the preprocessing to provide an ordered pattern to access the cells as well as the edges. This is made by reordering the cell numbering, by using the RCM (Reverse Cuthill-McKee) technique and then ordering the edges. (Lacasta et al., 2014).

These two strategies contribute to increase the coalescence of memory accesses, which makes the GPU implementation between 15% and 30% more efficient (Lacasta et al., 2015).

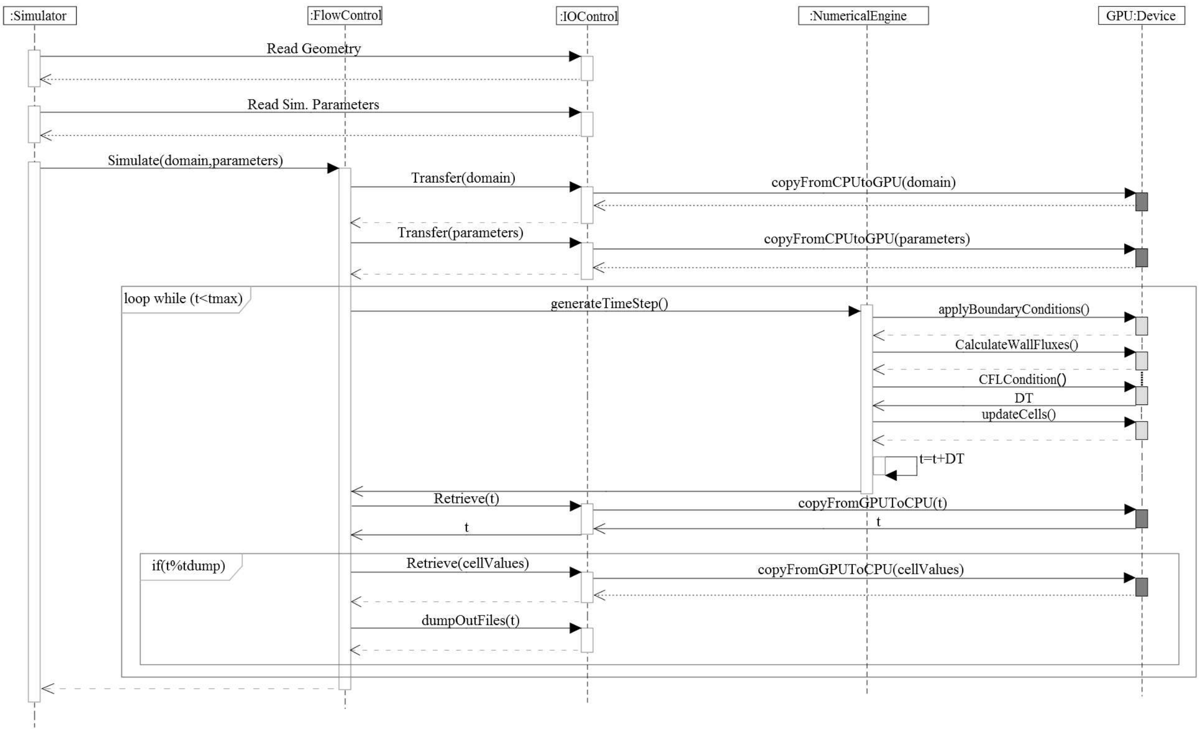

Figure 1 UML Sequence Diagram of the simulation process. Dark gray elements are memory interaction with the CPU and light gray elements are related to computing processes on the GPU.

3.3 *Details of the implementation*

As displayed in Fig. 1, the numerical scheme as in (14) may be decomposed in three main operations: the calculation of fluxes looping by cell edges, the election of the minimum time-step Δt, dynamically chosen to control the global stability, and the updating of the cells using the previous information. Using the CUDA toolkit, all the processed elements can be distributed by *threads* and *blocks* (of threads). Each thread uses its own thread index to identify the element to be processed. Then, the GPU launches several execution threads at the same time so that the calculations are performed in parallel.

As the GPU is well designed to work efficiently with ordered information, ordering techniques to reduce the distance in the memory address space of variables for cells i and j may produce a desirable effect. There are two main options to store the information: arrays of structures (AoS) or structures of arrays (SoA). The conserved variables $\{h, q_x, q_y\}$ can be stored sequentially by cells $(h^0, q_x^0, q_y^0, h^1, q_x^1, q_y^1, ..., h^{N_{cells}}, q_x^{N_{cells}}, q_y^{N_{cells}})$ generating an array of structures (AoS) or they can be stored grouped by variables as $(h^0, h^1, ..., h^{N_{cells}}, q_x^0, q_x^1, ..., q_x^{N_{cells}}, q_y^0, q_y^1, ..., q_y^{N_{cells}})$ forming three arrays with N_{cells} components each one (i.e. a structure of three arrays). Since all the threads within a block execute the same instruction at a certain moment, all of them may need to read the same variable. Therefore, a coalesced SoA improves spatial locality for these memory accesses (Lacasta et al., 2014).

In order to make feasible the calculations by edges, in the case of the fluxes computation, and by cells, in the update cells function, a strategy to access each element efficiently is required. In the case of the edge-based computations, each thread is devoted to calculate the numerical fluxes for each edge using differences across the edge of neighboring cells (i, j). Since each edge requires to know the value of the variables for each cell i and j, an auxiliary identifier vector is created. In Fig.

2 it is possible to see how this vector works. For instance, based on the thread index n, water depth for the cell i of the global index edge n is obtained by its correspondent index vIdxEdgeForCell$_i$(n) and analogously for cell j with another auxiliary vector using it as vIdxEdgeForCell$_j$(n). Here is where the optimized manner of distributing the information may improve these memory accesses.

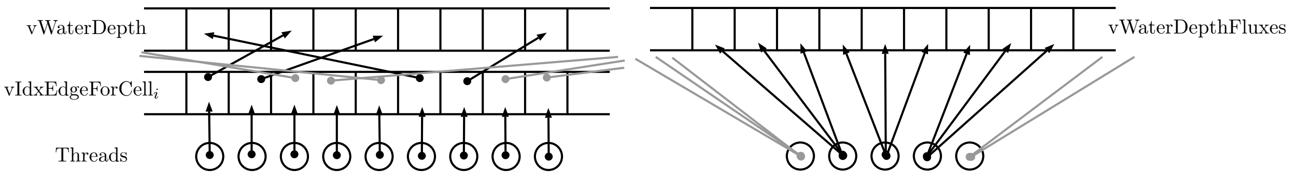

Figure 2 Sketch of the loading operations for one conserved variable (water depth h) in the fluxes calculation procedure (left) and loading operation of the fluxes calculated in the previous function for the update cells function (right)

Once the fluxes are calculated, they must be stored in another vector that will be read to update the cells. The way these elements are saved is using a vector (vIdxEdgeForLocalEdge(n,{0,1})) of size $2N_{edges}$ that relates the global edge indexing and the local index (i.e. $1, 2$ and 3 for each cell i or j). This vector contains the index that relates the local indexing for the cell i of the edge n in the position $2n$ and the equivalent for the cell j in the position $2n + 1$ (see Fig. 3).

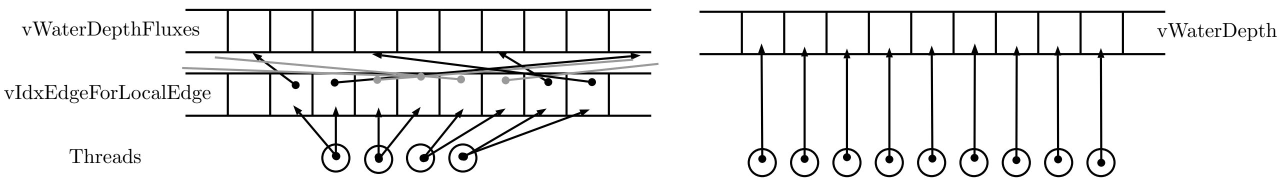

Figure 3 Sketch of the store operations for the fluxes related to the water depth variable in the fluxes calculation procedure (left) and storing operation for the update cells function (right)

When using the previous ideas, the updating procedure is simpler. Since it is necessary to integrate the inlet fluxes across the edges, it is required to add those correspondences to edge $1, 2$ and 3 by cells. As they have been stored sequentially, the access is performed consecutively given a cell identifier (i.e. given a thread, see Fig. 2). As the kernel is launched to perform the operations by cells (i.e. thread i corresponds to cell i), the storage is straightforward as the thread i will store data in the position i (see Fig. 3).

The last operation that is done in the GPU is the selection of the global time step. As the CFL restriction is governed by the celerities, $\widetilde{\lambda}^m, \widetilde{\lambda}_s$, at each edge in the global indexing n, the wall flux calculation step stores the local restriction for the time-step in the position n of a vector vDt. The global Δt is the minimum among them. To obtain that, a min-reduction primitive, as implemented in the CUBLAS library included in CUDA (NVIDIA Corporation, 2014), has been used. The operation `cublasidamin()` computes this reduction efficiently in the GPU and it returns the identifier of the minimum value within a vector (see Fig. 4).

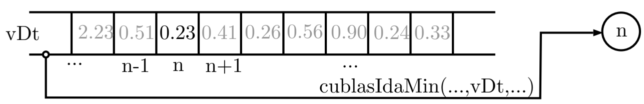

Figure 4 Min-reduction using CUBLAS to obtain the minimum Δt stored by edges

This last operation is included in the edge-loop of the CPU code and it is implemented using the common *reduction* OpenMP directive. It is important to take into account that it also represents a bottleneck in the CPU code.

4 Results

In this section, the solver implemented on the GPU is applied to two test cases in order to prove that the numerical prediction retains the accuracy of the original CPU solver, necessary to be reliable but also to measure the required computational speed in order to be efficient. Test 1 is based on a laboratory test case already considered by the authors for testing the numerical scheme in CPU (Juez et al., 2014). It allows to explore the accuracy and also the relative performance between a CPU and a GPU version. Test 2 shows the computational results for a real dam break event which took place in the past. Thanks to the GPU capabilities, it is affordable to design several possibilities in the dike breaching using desktop computing resources.

In both cases, unstructured meshes have been used with a dynamically computed time-step based on a CFL=0.5.

GPU implementation has been analyzed against single-core and multi-core CPU implementations. The computational time has been measured for the main loop of the numerical engine, that is, the $t < t_{max}$ loop displayed in Fig. 1. It includes not only the main computation but also those transfers between CPU and GPU required for dumping purposes as well as time-step accounting. Obviously, these operations only affect to the GPU implementation. The performance of the test cases has been measured through the speedup ratio.

Both the sequential and the parallel implementations have been tested on a Intel Core i7 3770K CPU while the GPU code has been run on a NVIDIA Titan Black GPU. It is important to remark that CPU implementation has not been fully optimized exploiting advanced capabilities such as vectorizations but multiprocessing has been included by means of OpenMP.

4.1 *2D laboratory dam break*

This experiment was carried out at the laboratory of the Civil and Environmental Engineering Department of the Université Catholique de Louvain (UCL) (Goutière, Soares-Frazao, & Zech, 2011; Palumbo, Soares-Frazao, Goutiere, Pianese, & Zech, 2008). It consists of a straight channel with a sudden enlargement. A sketch of the experimental set up is shown in Fig. 5. The bed material was uniform sand, gray area in Fig. 5, with the following properties: median diameter $d_{50} = 1.65$ mm, density $\rho_s = 2630$ kgm^{-3}, friction angle $\varphi = 15^o$, negligible cohesion, porosity $p = 0.42$ and a Manning roughness factor $n = 0.0185$ sm$^{-1/3}$.

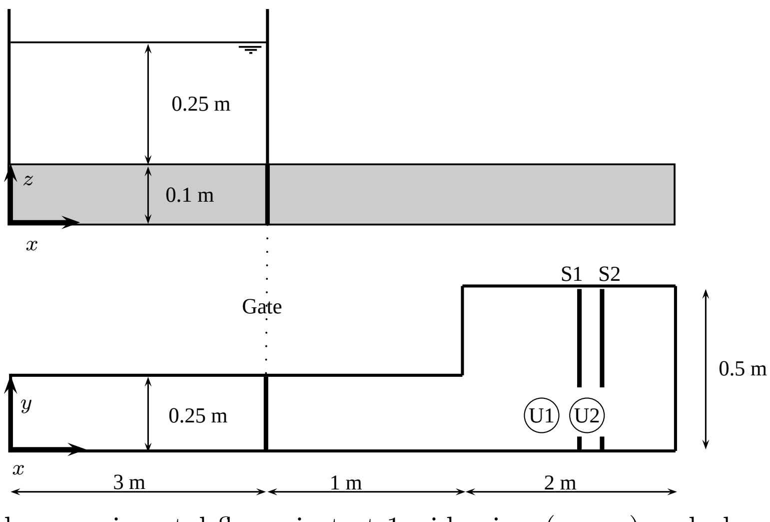

Figure 5 Sketch of the experimental flume in test 1: side view (upper) and plan view (lower)

This experiment was performed for simulating a dam break over erodible bed. For that purpose,

Table 1 Detail of execution time and speed-up for the compared implementations

1 Core	4 Cores		GPU	
t	t	s_{up}	t	S_{up}
6526.81 s	2331.52 s	2.95	115 s	56.75

in the middle of the straight channel there was a gate with an uniform water depth on the left. The gate was opened to release the water and due to the presence of the abrupt expansion a local erosion was generated and the material eroded by the flow was deposited in the vicinity of the wall area with the form of a bar. Later, the bar migrated and the erosion area increased its depth. This natural evolution is observed in Fig. 6, where the computational results for the erosion (-) and deposition (+) rates are plotted in time. The computational domain was discretized with 98000 cells. Despite the complexity of this test case, including wet/dry conditions, moving shocks and important erosion/deposition rates, no numerical instabilities are observed thanks to the augmented stability criterion.

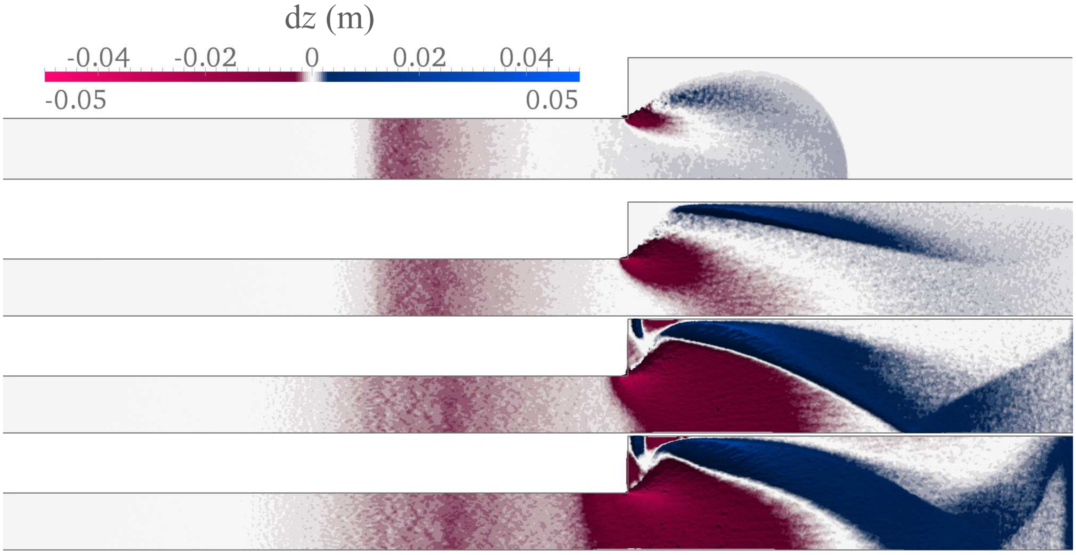

Figure 6 Bed surface variation at times t=1, 2, 4, 16 s

The numerical predictions are compared with the experimental data. Figure 7 displays the comparison between the water level measured and the numerical solution at two locations, U1 (x= 4.2 m, y= 0.125 m) and U2 (x= 4.45 m, y= 0.125 m). Additionally, the bed level is also compared at the end of the experiment in two sections, S1 (x= 4.4 m) and S2 (x= 4.5 m) in Fig. 8. Both, water and bed numerical estimations, are able to track the tendency of the experiment ensuring a correct comparison. Main differences in cross sections are due to the fact that the mathematical model considered in this work is depth averaged and consequently, the vertical flow accelerations are neglected. Therefore a mismatch in the results in the area close to the left wall is expected. It is worth noting here that the quality of the numerical results is the same as that offered by the CPU version of the method already published elsewhere (Juez et al., 2014). Discussion of the limitations of the underlying mathematical model or numerical method is out of the scope of the present study.

Table 1 collects the information concerning the computational effort using the CPU (1 and 4 Cores) and the GPU. As it can be observed the speedup with the GPU is roughly 57 meaning that this implementation is 57 times faster than the 1 core CPU model.

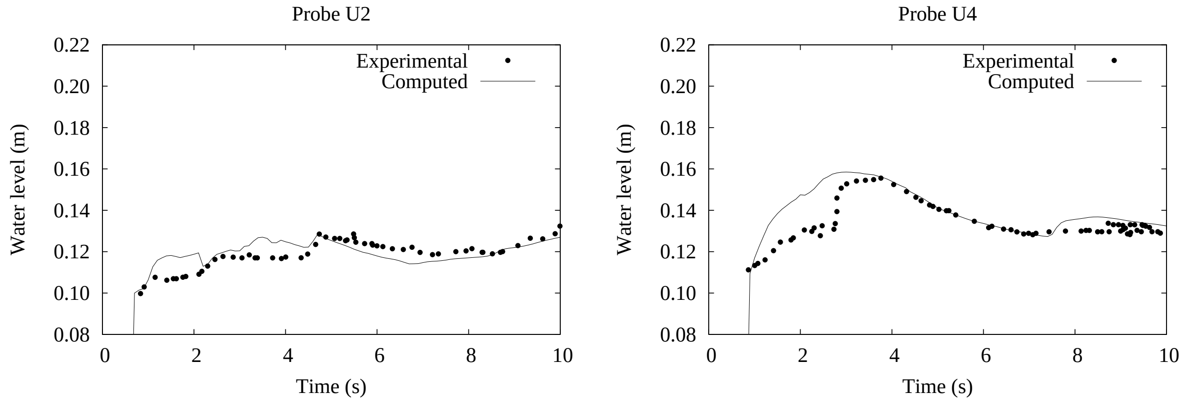

Figure 7 Temporal comparison between experimental and computed results for the water level at probes U1 and U2

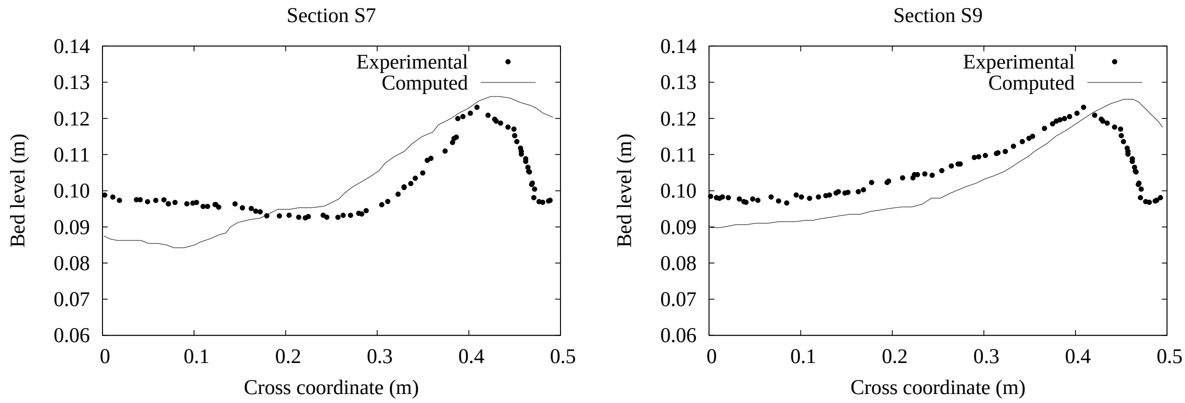

Figure 8 Comparison of the experimental and computed final bed surface at cross sections S1 and S2

4.2 *Tous dam break*

In this test the authors address the possibility of using large spatial domains, that require a high number of cells, for flood warning/hazard prediction. For this purpose the dam failure of Tous dam is proposed (Alcrudo & Mulet, 2007).

Tous dam is the last flood control structure of the Júcar River basin in the central part of the Mediterranean coast of Spain. During the 20th and the 21st October 1982 a particular meteorological condition led to extremely heavy rainfall. As a result the Júcar River basin suffered flooding all along and the Tous Dam failed with devastating effects downstream. The first affected town was Sumacárcel, about 5 km downstream of Tous Dam, lying at the toe of a hill on the right bank of Júcar river (Alcrudo & Mulet, 2007). The terrain is moderately mountainous and most of the buildings lie on a slope that partially protected them from the flood. The ancient part of the village, however, is located closer to the river course and was completely flooded, with high water marks reaching between 6 m and 7 m.

The DTM model used in this work was generated by CEDEX in 1998 Alcrudo and Mulet (2007). From this information a numerical mesh with $3 \cdot 10^5$ cells has been defined. This computational domain covers most of the original DTM, starting just after the dam location and finishing approximately 1 km downstream of Sumacárcel. The mesh has been refined in the dam area and in the village area (Fig. 9) for providing an adequate resolution for the hydraulic structures and the buildings. It is stressed that the decrease in the cell size leads to an increment in the simulation time since the stability criterion is more restrictive.

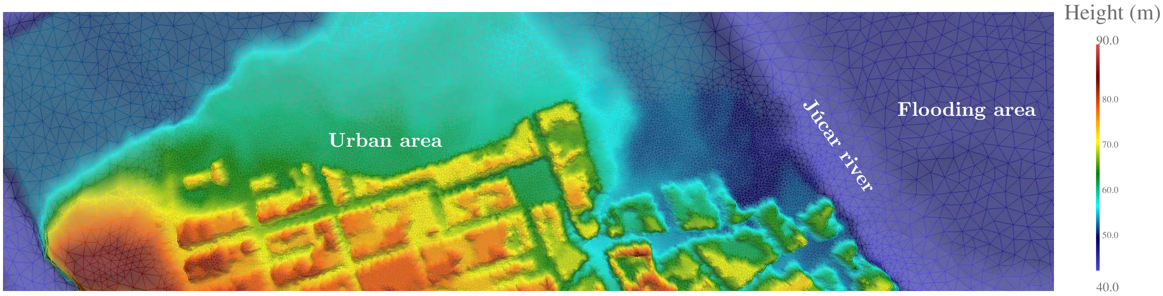

Figure 9 Detail of the simulation mesh at the village area nearby

The cause of the dam break was overtopping/dam-breaching, due to intense rainfall, and its later erosion and collapse. The height of the dam crest was 98.5 m and before reaching this level the discharge facilities of the dam were opened in order to evacuate the huge amount of incoming water. To reproduce this situation, the authors have considered the water elevation records together with the reservoir rating curves for simulating the spillway procedure, i.e. a water discharge of 3568 m^3s^{-1} is considered for obtaining the initial condition. Once the crest level is reached, a dam breach starts and it causes the erosion and collapse process. Hence, an outflow discharge emerging from the dam creates the traveling wave which is the responsible for the flooding event, i.e. it is the key information for the prediction of this event. In previous studies (Alcrudo & Mulet, 2007), since the morphodynamic change of the dam was not modeled, a tuning synthetic discharge, based on several assumptions, was estimated. Finally, at the outlet boundary, downstream of the domain, the flow was let to exit freely without imposing any conditions, as no information was provided.

On the other hand, following Alcrudo and Mulet (2007), a Manning coefficient of 0.030 $sm^{-1/3}$ has been set for the whole river bed reach and, additionally, an increased roughness coefficient of 0.1 $sm^{-1/3}$ has been defined in two zones close to the village with dense orange trees. The mean sediment diameter involved in the erosion process has been set to 0.02 m. As the ground in the town area was fully paved with concrete the flood did not erode it. The real time simulated has been 11.1 hours from the beginning of the dam overtopping.

In Fig. 10 the breach evolution of the dam is plotted at several times. The flow overtopping causes the inception of the erosion at the front edge of the dam crest. As the breach increases in size the flow is accelerated and a severe erosion occurs. Consequently, the water discharge in the breach also augments. The earthfill material is grabbed by the flow and it is settled downstream the dam creating a sediment tongue which migrates towards the riverbed. At the end of the event the morphology of the dam area has changed completely and an important fraction of the dam has been completely removed, which is in agreement with the photos taken after the event and provided in Alcrudo and Mulet (2007).

The evolution of the computed flooding can be seen in full plan view in Fig. 11 at times t =0, 1.3, 2.7 and 11.1 hours considering the time t=0 when the water surface level inside the reservoir has reached the dam crest and the overtopping is about to start. The flow advances towards the village filling the riverbed capacity and, consequently, inundating the floodplain areas nearby.

Thanks to the work described in Alcrudo and Mulet (2007), there are field data for the estimation of: (i) the maximum and minimum levels reached by the flood wave or (ii) a unique level for the water surface at different locations within the town, for evaluating the quality of the simulations. This estimation was performed considering a range of values within which it was completely ensured that the water reached that level. The location of the gauging points is shown in Fig. 12. Figure 13 displays the water depth recorded at several locations in Sumacárcel village together with the numerical predictions. There is a good agreement between the field data and the estimated depth,

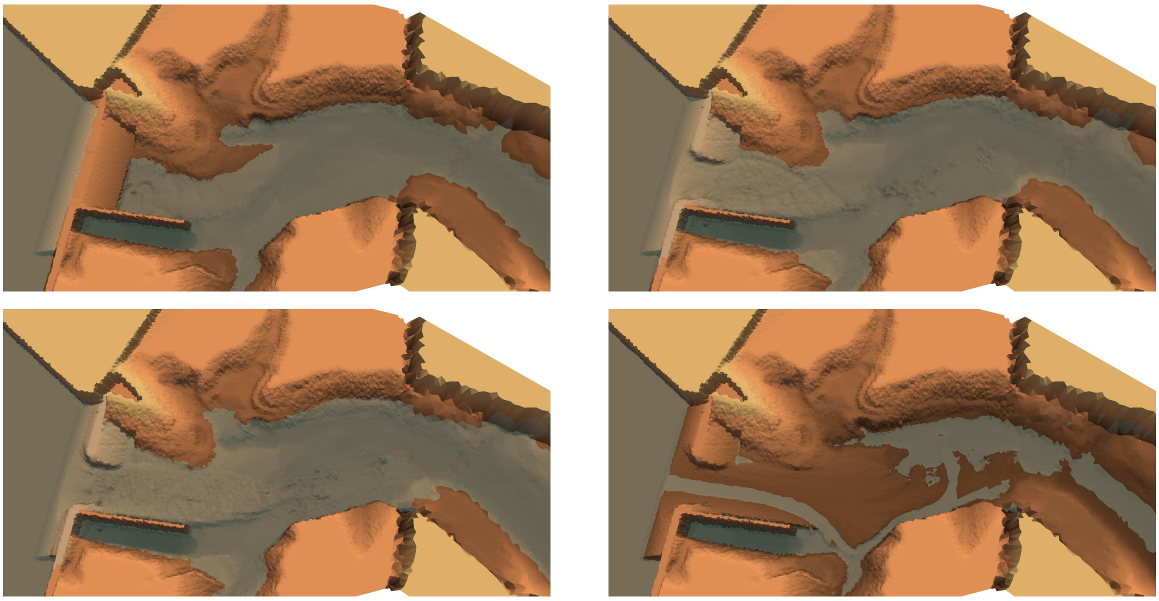

Figure 10 Initial condition (Top-Left) and evolution of the erosion process at t=1.3 hours (Top-Right), t=2.7 hours (Bottom-left) and at final stage (t=11.1 hours) (Bottom-right)

since most of the probes reach the range, between the maximum and minimum, estimated during the event. This agreement is attributed to the adequate simulation of the erosion process at the Tous dam.

It is also important to highlight that, by coupling the hydrodynamic and the breach erosion phenomena, less assumptions are required. This may be relevant in practical applications but is costly in computational terms. For instance, in Alcrudo and Mulet (2007) a synthetic hydrograph based on a detailed analysis of how the dam failed was proposed. However, thanks to the GPU capabilities it is possible to couple the hydrodynamics and the dam erosion for obtaining directly the hydrograph which is the responsible for the later flooding event. In Fig. 14 both hydrographs, the synthetic and the computed one in the dam-breach, are plotted. It is remarkable that the peak discharge observed by means of the simulation, Q_{peak}= 14568.09 m^{3}s^{-1}, is very close to the peak discharge estimated in Alcrudo and Mulet (2007), where Q_{peak}= 15000 m^{3}s^{-1}. Conversely, the computed discharge is less sustained in time. This difference is probably because the inlet tributaries of the reservoir have been neglected. Since this effect has not been taken into account, in Alcrudo and Mulet (2007) there is not a fair estimation of the magnitude of these inlet tributaries, only the water contained in the reservoir at the beginning of the event is allowed to outflow in the simulation.

The evolution of the dam-breach is also plotted in Fig. 14 using the same cross section used to evaluate the discharge. It can be observed that most of the process has occurred within the first 1500 s, i.e. during the peak discharge. After t=1500 s changes in bed morphology are less violent.

The execution time is summarized in Table 2. In this case, only the parallel CPU version has been benchmarked due to the huge execution time required for the single-core CPU version. The GPU reduces the simulation effort 25 times compared with the 4-Core version allowing an efficient simulation and accurate prediction. It is important to take into account that, in this case, the improvement has been increased compared against the previous cases where the GPU accelerates the computation of the OpenMP solution in a 20 factor. This effect has been previously reported in hydrodynamic simulation in Lacasta et al. (2014) and it is due to the large number of elements included in the calculation. Thanks to the GPU capabilities it has been affordable to locally refine the mesh in the breach area and provide an adequate design for the initial breach which provides

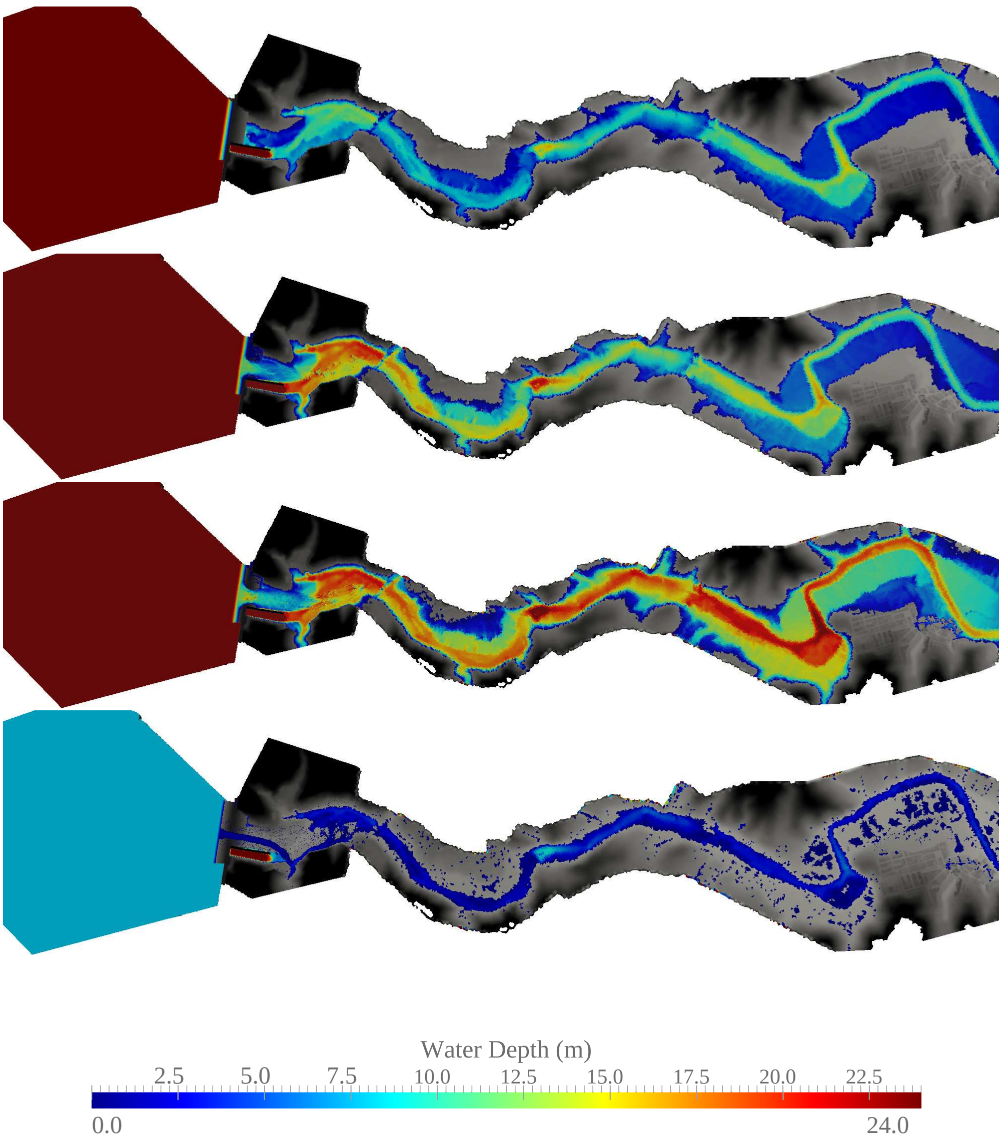

Figure 11 Water depth evolution along the valley at times t=0, 1.3, 2.7, 11.1 hours from top to bottom

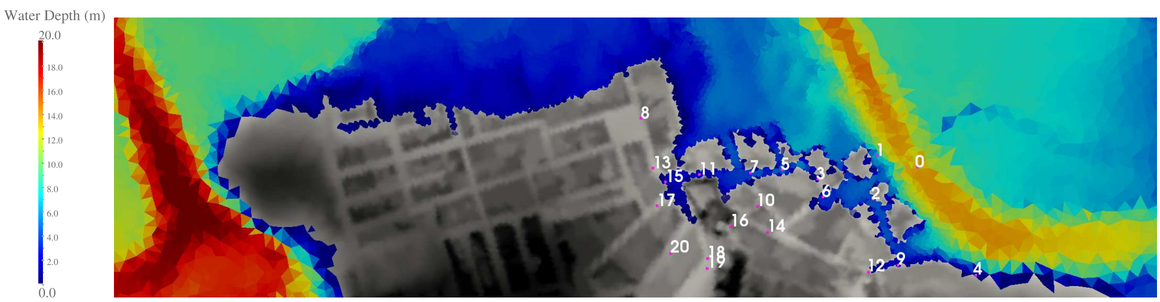

Figure 12 Detail of the location of the gauging points

the dam-breaching discharge. Therefore, several possibilities can be addressed in the same day which is a noticeable advance when comparing with the computational effort based on CPU.

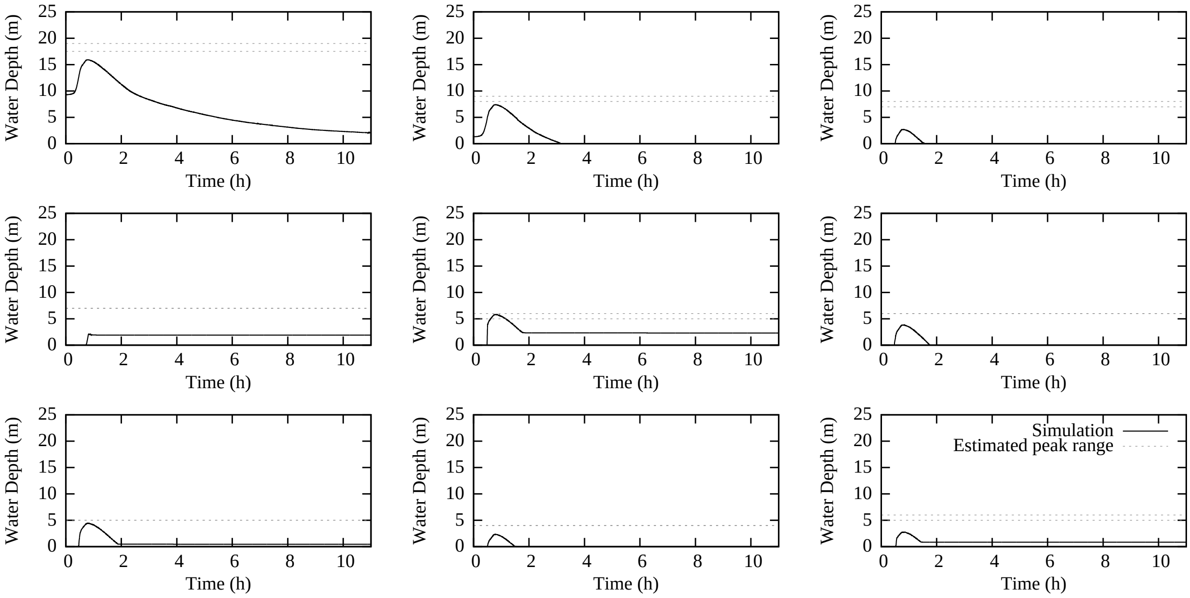

Figure 13 Water depth numerical predictions at several locations in Sumacárcel village and estimated range provided in Alcrudo and Mulet (2007) for gauges 1, 2, 3, 4, 6, 7, 11, 12 and 15 (from top to bottom and from left to right), see Fig. 12

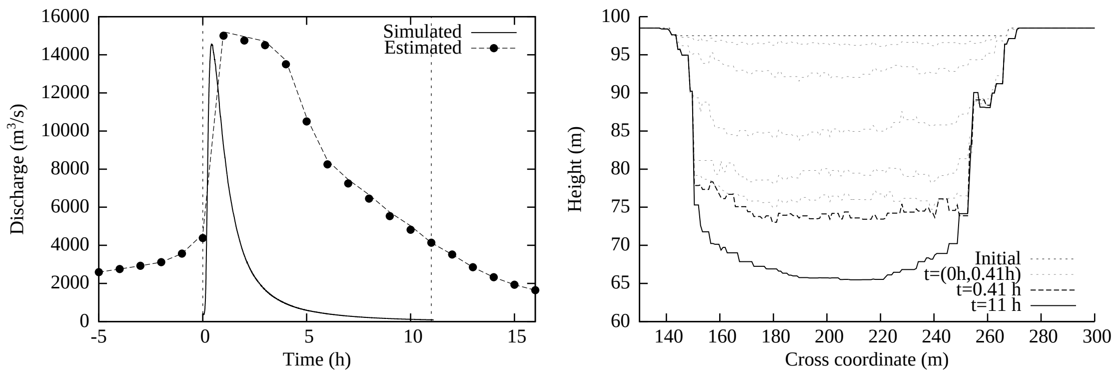

Figure 14 (Left) Comparison of the hydrograph generated due to the dam failure using the presented implementation against the hydrograph estimated in Alcrudo and Mulet (2007). Simulated window is highlighted considering the time interval between $t = 0$ and $t = 11$ hours. (Right) Evolution of the dam-breach from $t = 0$ to $t = 0.41$ hours (peak discharge) each 0.07 hour and $t = 11.0$ hours (final state)

5 Conclusions

The new opportunities given by the GPU implementation have been described in this work for the analysis of several situations where the morphodynamic effects are relevant. For this purpose, the shallow water equations in combination with the Exner equation have been discretized in Finite Volumes and the numerical schemes implemented to run on a GPU card. This model allows to properly represent the propagation of bed and surface waves over realistic bathymetries in affordable computation time even when considering large domains and retaining a high level of accuracy.

For maximizing the speedup performance, several strategies have been proposed in order to improve the implementation of the numerical scheme in these hardware devices: the use of Structure

Table 2 Detail of execution time and speed-up for the compared implementations

4 Cores	GPU	
t	t	S_{up}
207 h 7 min	8h 7 min	25.25

of Arrays (SoA) instead of Arrays of Structures (AoS), the cells reordering and the walls reordering. These optimization techniques allow a faster memory access reducing the execution time.

The speedups have been computed involving the performance of single-core and multi-core processors. The GPU implementation provides a peak speedup of 50. This saving of time allows to address large-number-of-cells, large-time and large-space scenarios, strengthening preventive measures and enhancing response capacities.

As future work, the authors will focus on the implementation of these methods on a cluster of GPUs. This kind of distributed computing will allow to compute morphodynamic problems in a larger scale. This opens the possibility of facing the sediment transport analysis in a particular location for several years or the geomorphological changes in domains of a regional-size.

Acknowledgments

The authors would like to thank the anonymous reviewers and the editor for their valuable comments and suggestions to improve the quality of the paper.

Funding

This work was partially supported and funded by the Spanish Ministry of Science and Technology under research projects CGL2011-28590, BIA2011-30192-C02-01 and by Diputación General de Aragón, DGA, through FEDER funds.

Notation

x	= spatial coordinate in the longitudinal direction (m)
y	= spatial coordinate in the traversal direction (m)
z	= bed level (m)
t	= time (s)
h	= water depth (m)
u	= depth averaged velocity in x coordinate (ms^{-1})
v	= depth averaged velocity in y coordinate (ms^{-1})
q_x	= unit water discharge in x coordinate (m^{2}s^{-1})
q_y	= unit water discharge in y coordinate (m^{2}s^{-1})
$q_{s,x}$	= unit sediment discharge in x coordinate (m^{2}s^{-1})
$q_{s,y}$	= unit sediment discharge in y coordinate (m^{2}s^{-1})
g	= gravity acceleration (ms^{-2})
n	= Manning coefficient (sm$^{-1/3}$)
p	= sediment porosity (–)
s	= ratio between sediment and water densities (–)
d_m	= grain median diameter (m)
d_{30}	= representative grain diameter for 30% of the weight of the sample (m)
d_{90}	= representative grain diameter for 90% of the weight of the sample (m)
S	= slope in the Smart formula (–)
A_i	= cell area (m^2)
n_x	= normal component in x coordinate
n_y	= normal component in y coordinate
F	= Froude number (–)
ρ_w	= water density (kgm^{-3})
ρ_s	= sediment density (kgm^{-3})
θ	= dimensionless shear stress (–)
θ_c^S	= dimensionless Shields parameter according Smart (–)
Δt	= timestep (s)

References

Alcrudo, F., & Mulet, J. (2007). Description of the Tous dam break case study (Spain). *Journal of Hydraulic Research*, *45(Extra Issue)*, 45–57.

Aricò, C., & Tucciarelli, T. (2008). Diffusive modeling of aggradation and degradation in artificial channels. *Journal of Hydraulic Engineering*, *134(8)*, 1079–1088.

Begnudelli, L., Valiani, A., & Sanders, B. F. (2010). A balanced treatment of secondary currents, turbulence and dispersion in a depth-integrated hydrodynamic and bed deformation model for channel bends. *Advances in Water Resources*, *33*, 17–33.

Bilaceri, M., Beux, F., Elmahi, L., Guillard, H., & Salvetti, M. (2012). Linearized implicit time advancing and defect correction applied to sediment transport simulations. *Computers and Fluids*, *63*, 82–104.

Burguete, J., & García-Navarro, P. (2001). Efficient construction of high-resolution TVD conservative schemes for equations with source terms: application to shallow water flows. *International Journal of Numerical Methods in Fluids*, *37*, 209–248.

Canelas, R., Murillo, J., & Ferreira, R. (2013). Two-dimensional depth-averaged modelling of dam-break flows over mobile beds. *Journal of Hydraulic Research*, *51(4)*, 392–407.

Caviedes-Voullieme, D., Morales-Hernandez, M., Lopez-Marijuan, I., & Garcia-Navarro, P. (2014). Reconstruction of 2D river beds by appropiate interpolation of 1D cross-sectional information for flood simulation. *Environmental Modelling and Software*, *61*, 206–228.

Chang, H. (1982). Mathematical model for erodible channels. *Journal of Hydraulic Engineering*, *108*, 678–689.

Danalis, A., Marin, G., McCurdy, C., Meredith, J. S., Roth, P. C., Spafford, K., et al. (2010). The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd workshop on general-purpose computation on graphics processing units* (pp. 63–74).

Gandham, R., Medina, D., & Warburton, T. (2014). GPU Accelerated discontinuous Galerkin methods for shallow water equations. *arXiv preprint arXiv:1403.1661*.

Garcia, R., Restrepo, P., DeWeese, M., Ziemer, M., Palmer, J., Thornburg, J., et al. (2015). Advanced GPU paralellization for two-dimensional operational river flood forecasting. *36th IAHR World Congress*.

Garegnani, G., Rosatti, G., & Bonaventura, L. (2013). On the range of validity of the Exner-based models for mobile-bed river flow simulations. *Journal of Hydraulic Research*, *51(4)*, 380–391.

Goutière, L., Soares-Frazao, S., & Zech, Y. (2011). Dam-break flow on mobile bed in abruptly widening channel: experimental data. *Journal of Hydraulic Research*, *49(3)*, 367–371.

Hou, J., Liang, Q., Zhang, H., & Hinkelmann, R. (2015). An efficient unstructured MUSCL scheme for solving the 2D shallow water equations. *Environmental Modelling and Software*, *66*, 131–152.

Juez, C., Murillo, J., & García-Navarro, P. (2013). Numerical assesment of bed load discharge formulations for transient flow in 1D and 2D situations. *Journal of Hydroinformatics*, *15(4)*, 1234–1257.

Juez, C., Murillo, J., & García-Navarro, P. (2014). A 2D weakly-coupled and efficient numerical model for transient shallow flow and movable bed. *Advances in Water Resources*, -, In press.

Kalyanapu, A., Siddharth, S., Pardyjak, E., Judi, D., & Burian, S. (2011). Assessment of GPU computational enhancement to a 2D flood model. *Environmental Modelling and Software*, *26*, 1009–1016.

Lacasta, A., García-Navarro, P., Burguete, J., & Murillo, J. (2013). Preprocess static subdomain decomposition in practical cases of 2D unsteady hydraulic simulation. *Computers & Fluids*, *80*, 225–232.

Lacasta, A., Juez, C., Murillo, J., & García-Navarro, P. (2015). An efficient solution for hazardous geophysical flows simulation using GPUs. *Computers & Geosciences*, *78*, 63–72.

Lacasta, A., Morales-Hernández, M., Murillo, J., & García-Navarro, P. (2014). An optimized GPU implementation of a 2D free surface simulation model on unstructured meshes. *Advances in Engineering Software*, *78*(1), 1-15.

Leveque, R. (2002). *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, New York.

Liu, Q., Quin, Y., Zhang, Y., & Li, Z. (2015). A coupled 1D-2D hydrodynamic model for flood simulation in flood detention basin. *Natural Hazards*, *75(2)*, 1303–1325.

Munshi, A., et al. (2009). The OPENCL specification. *Khronos OpenCL Working Group*, *1*, l1–15.

Murillo, J., & García-Navarro, P. (2010a). An Exner-based coupled model for two-dimensional transient flow over erodible bed. *Journal of Computational Physics*, *229*, 8704–8732.

Murillo, J., & García-Navarro, P. (2010b). Weak solutions for partial differential equations with source terms: Application to the shallow water equations. *Journal of Computational Physics*, *229*, 4327–4368.

Murillo, J., García-Navarro, P., Brufau, P., & Burguete, J. (2008). 2D modelling of erosion/deposition processes with suspended load using upwind finite volumes. *Journal of Hydraulic Research*, *46(1)*, 99–112.

Murillo, J., García-Navarro, P., & Burguete, J. (2008). Time step restrictions for well balanced shallow water solutions in non-zero velocity steady states. *International Journal of Numerical Methods in Fluids*, *56*, 661–686.

NVIDIA Corporation. (2007). NVIDIA CUDA Compute unified device architecture programming guide [Computer software manual]. NVIDIA Corporation.

NVIDIA Corporation. (2014). CUDA Toolkit 6.0 [Computer software manual].

Palumbo, A., Soares-Frazao, S., Goutiere, L., Pianese, D., & Zech, Y. (2008). *Proc., River Flow 2008 International Conference on Fluvial hydraulics, Cesme.*

Petaccia, G., Natale, L., Savi, F., Velickovic, M., Zech, Y., & Soares-Frazao, S. (2013). Flood wave propagation in steep mountain rivers. *Journal of Hydroinformatics*, *15(1)*, 120–137.

Serrano, A., Murillo, J., & García-Navarro, P. (2012). Finite volumes for 2D shallow-water flow with bed-load transport on unstructured grids. *Journal of Hydraulic Research*, *50(2)*, 154–163.

Siviglia, A., Stecca, G., Vanzo, D., Zolezzi, G., Toro, E., & Tubino, M. (2013). Numerical modelling of two-dimensional morphodynamics with applications to river bars and bifurcations. *Advances in Water Resources*, *52*, 243–260.

Smart, G. (1984). Sediment transport formula for steep channels. *Journal of Hydraulic Engineering*, *3*, 267–276.

Soares-Frazao, S., & Zech, Y. (2010). HLLC scheme with novel wave-speed estimators appropiate for two-dimensional shallow-water flow on erodible bed. *International Journal of Numerical Methods in Fluids*, *66(8)*, 1019–1036.

Vacondio, R., Dal Pal, A., & Mignosa, P. (2014). GPU-enhanced finite volume shallow water solver for fast flood simulations. *Envirnonmental Modelling and Software*, *57*, 60–75.

Villaret, C., Hervouet, J., Kopmann, R., Merkel, U., & Davies, A. (2013). Morphodynamic modeling using the Telemac finite-element system. *Computers and Geosciences*, *53*, 105–113.

Wu, W. (2004). Depth-averaged two-dimensional numerical modeling of unsteady flow and nonuniform sediment transport in open channels. *Journal of Hydraulic Engineering*, *130(10)*, 1013–1024.

Xia, J., Lin, B., Falconer, R., & Wang, G. (2010). Modelling dam-break flows over mobile beds using a 2D coupled approach. *Advances in Water Resources*, *33*, 171–183.