A local time stepping algorithm for GPU-accelerated 2D shallow water models

Contents lists available at ScienceDirect

Advances in Water Resources

journal homepage: www.elsevier.com

A local time stepping algorithm for GPU-accelerated 2D shallow water models

Susanna Dazzi[a,*], Renato Vacondio[a], Alessandro Dal Palù[b], Paolo Mignosa[a]

[a] Department of Engineering and Architecture, University of Parma, Parco Area delle Scienze 181/A, Parma 43124, Italy
[b] Department of Mathematical, Physical and Computer Sciences, University of Parma, Parco Area delle Scienze 53/A, Parma 43124, Italy

ARTICLE INFO

Keywords:
GPU
Local time stepping
Shallow water equations
Finite volume
Flood simulation
Parallel computing

ABSTRACT

In the simulation of flooding events, mesh refinement is often required to capture local bathymetric features and/or to detail areas of interest; however, if an explicit finite volume scheme is adopted, the presence of small cells in the domain can restrict the allowable time step due to the stability condition, thus reducing the computational efficiency. With the aim of overcoming this problem, the paper proposes the application of a Local Time Stepping (LTS) strategy to a GPU-accelerated 2D shallow water numerical model able to handle non-uniform structured meshes. The algorithm is specifically designed to exploit the computational capability of GPUs, minimizing the overheads associated with the LTS implementation. The results of theoretical and field-scale test cases show that the LTS model guarantees appreciable reductions in the execution time compared to the traditional Global Time Stepping strategy, without compromising the solution accuracy.

1. Introduction

The heavy economic and human losses caused by flood events in recent years (Barredo, 2009) demand the adoption of specific flood risk management strategies, which include not only structural defense systems (such as levees), but also resilient policies. From this point of view, the numerical modeling of flood scenarios can be very helpful both for the definition of the most effective technical interventions, and for the design of flood-hazard and flood-risk maps, necessary for flood risk management planning (as requested by the European Council, 2007).

Most free-surface flows, such as tsunamis, river floods, and dam-break events, can be simulated through the two-dimensional (2D) Shallow Water Equations (SWE), which can be discretized by different numerical methods (e.g. Alcrudo and Garcia–Navarro, 1993; Horritt and Bates, 2002; Vacondio et al., 2012; Costabile et al., 2012, 2015). Among these, Finite Volume (FV) schemes have the advantage of accurately describing transcritical flows and shock-type discontinuities (Toro, 1999), and can even be applied to flows over irregular bathymetries, provided that the scheme is associated with a specific treatment of wetting and drying fronts (e.g. Bradford and Sanders, 2002; Brufau et al., 2004; Liang and Borthwick, 2009), and of bed and friction slope source terms, which should guarantee that the C-property is satisfied (e.g. Bermudez and Vazquez, 1994; Greenberg and Leroux, 1996; Garcia-Navarro and Vazquez, 2000; Rogers et al., 2003; Audusse et al., 2004; Aureli et al., 2008; Liang and Marche, 2009; Vacondio et al., 2013).

The high computational cost entailed by the application of 2D SWE models to field-scale simulations with high-resolution meshes encouraged researchers to investigate parallelization techniques on supercomputers (e.g. Sanders et al., 2010). Moreover, the computational capability of Graphics Processing Units (GPUs) has been recently exploited to carry out High Performance Computing on personal workstations, and GPU-accelerated SWE models are now developed by a number of authors, both for structured (Lastra et al., 2009; de la Asuncion et al., 2013; Brodtkorb et al., 2012; Vacondio et al., 2014, 2017; Ferrari et al., 2017) and unstructured meshes (Castro et al., 2011; Lacasta et al., 2014; Petaccia et al., 2016). Results of these works show that significant reductions in computational time, up to two orders of magnitude compared with serial codes, can be achieved.

In many practical applications, local mesh refinement is often necessary, due to the presence of complex bathymetric features or specific areas of interest, and some GPU-enhanced models able to handle non-uniform resolution have been presented in literature (Sætra et al., 2015; Vacondio et al., 2017). Explicit numerical schemes are often preferred to implicit ones (as shown by Teng et al., 2017), due to their suitability for describing rapidly varying flows, but also to the possibility of designing efficient parallel models, exploiting modern heterogeneous computing architectures. However, the time step size used to update the solution with explicit schemes is computed based on the Courant-Friedrichs-Lewy (CFL) stability condition, and is generally dictated by the minimum grid size (neglecting the possible high variabil-

* Corresponding author.
Email addresses: susanna.dazzi@unipr.it (S. Dazzi); renato.vacondio@unipr.it (R. Vacondio); alessandro.dalpalu@unipr.it (A.D. Palù); paolo.mignosa@unipr.it (P. Mignosa)

https://doi.org/10.1016/j.advwatres.2017.11.023
Received 26 April 2017; Received in revised form 25 September 2017; Accepted 23 November 2017
Available online xxx
0309-1708/

ity in wave celerity values). All cells in the domain are advanced with the common (minimum) time step. This strategy is usually referred to as Global Time Stepping (GTS). If only a small region is refined, the other part of the domain, with a coarser grid size, is updated with a much smaller time step than its maximum admissible for stability, thereby increasing the required computational effort. In order to overcome this problem, Osher and Sanders (1983) introduced the Local Time Stepping (LTS) strategy, based on the idea of advancing each cell with its own time step, closer to its maximum allowable. LTS schemes were successfully applied to SWE models (Crossley and Wright, 2005; Sanders, 2008; Kesserwani and Liang, 2015; Dazzi et al., 2016), and appeared to be beneficial for enhancing their computational performances compared to standard GTS schemes, without degrading the solution accuracy. To the best of the authors' knowledge, the only attempt to apply a LTS scheme in a GPU-accelerated model can be attributed to Sætra et al. (2015), who focused on an Adaptive Mesh Refinement (AMR) technique.

This paper presents an efficient GPU implementation of a first- and second-order accurate FV scheme with a LTS strategy, which solves the 2D SWEs on a non-uniform structured grid. The present model is based on an existing GPU-model (Vacondio et al., 2014, 2017), which has been thoroughly tested and validated, even for the simulation of real flooding events (Vacondio et al., 2016). Compared to previous LTS techniques, the algorithm presented in this work has been derived and coded in such a way that it can exploit the computational capabilities of GPUs. The performances of the GTS and LTS schemes are assessed based on three theoretical and two real-field test cases.

The paper is structured as follows. Section 2 briefly describes the original GTS model. In Section 3, the new LTS scheme and its GPU implementation are detailed. Then, the model performances are evaluated through the simulation of five test cases in Section 4. Conclusions are drawn in Section 5.

2. Global time stepping scheme

The original GTS numerical model is described in detail by Vacondio et al. (2014, 2017). Here, only the most important features are briefly recalled. The governing equations are the 2D SWEs written in integral form (Toro, 2001):

$$\frac{\partial}{\partial t}\int_A \mathbf{U}\ \mathrm{d}A + \int_C \mathbf{H}\cdot\ \mathbf{n}\ \mathrm{d}C = \int_A \left(\mathbf{S}_0 + \mathbf{S}_f\right)\ \mathrm{d}A, \tag{1}$$

where t is the time, A is the area of the integration volume, C is the volume boundary, $\mathbf{U}$ is the vector of conserved variables, $\mathbf{H} = (\mathbf{F},\mathbf{G})$ is the tensor of fluxes in the x and y directions, $\mathbf{n}$ is the outward unit vector normal to C, $\mathbf{S}_0$ and $\mathbf{S}_f$ are the bed and friction slope source terms, respectively. The modified form of the SWEs, which guarantees that the scheme is well-balanced, is adopted (following Liang and Marche, 2009):

$$\begin{aligned}\mathbf{U} &= \begin{bmatrix}\eta\\ uh\\ vh\end{bmatrix},\ \mathbf{F}\\ &= \begin{bmatrix}uh\\ u^2h + \frac{1}{2}g\left(\eta^2 - 2\eta z\right)\\ uvh\end{bmatrix},\ \mathbf{G}\\ &= \begin{bmatrix}vh\\ uvh\\ v^2h + \frac{1}{2}g\left(\eta^2 - 2\eta z\right)\end{bmatrix},\end{aligned} \tag{2a}$$

$$\mathbf{S}_0 = \begin{bmatrix}0\\ -g\eta\frac{\partial z}{\partial x}\\ -g\eta\frac{\partial z}{\partial y}\end{bmatrix},\ \mathbf{S}_f = \begin{bmatrix}0\\ -gh\frac{n_f^2 u\sqrt{u^2+v^2}}{h^{4/3}}\\ -gh\frac{n_f^2 v\sqrt{u^2+v^2}}{h^{4/3}}\end{bmatrix}, \tag{2b}$$

In Eq. (2), h is the flow depth, z is the bed elevation above datum, and $\eta = h + z$ is the water surface elevation above datum; u and v are the velocity components along the x and y directions respectively, g is the acceleration due to gravity, and n_f is Manning's roughness coefficient.

An explicit FV scheme is used to discretize the equations; both first-order and second-order accurate approximations in space and time are implemented. The first-order approximation exploits the following equation to update the conserved variables in time:

$$\begin{aligned}\mathbf{U}_{i,j}^{n+1} = \mathbf{U}_{i,j}^{n} &- \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i+\frac{1}{2},j} - \mathbf{F}_{i-\frac{1}{2},j}\right)\\ &- \frac{\Delta t}{\Delta y}\left(\mathbf{G}_{i,j+\frac{1}{2}} - \mathbf{G}_{i,j-\frac{1}{2}}\right) + \Delta t\left(\mathbf{S}_0 + \mathbf{S}_f\right).\end{aligned} \tag{3}$$

Subscripts i,j represent the cell position, while superscript n refers to the time level; Δx and Δy are the cell dimensions in the x and y directions respectively, and Δt is the time step size, computed according to the CFL stability condition (Toro, 2001):

$$\Delta t_{i,j} = \frac{1}{2}Cr\min\left(\frac{\Delta x}{\left|u_{i,j}\right| + \sqrt{g h_{i,j}}}, \frac{\Delta y}{\left|v_{i,j}\right| + \sqrt{g h_{i,j}}}\right), \tag{4a}$$

$$\Delta t = \min_{i,j}\left(\Delta t_{i,j}\right), \tag{4b}$$

where Cr is the Courant number ($\leq$1). Notice that, according to the GTS scheme, each cell is updated with the same time step Δt, equal to the minimum value in the whole domain.

Second-order accuracy in space is achieved by means of a depth-positive MUSCL extrapolation at cell boundaries (Audusse et al., 2004), adopting the *minmod* slope limiter. Second-order accuracy in time is obtained by applying the second-order Runge–Kutta method:

$$\mathbf{U}_{i,j}^{n+1} = \mathbf{U}_{i,j}^{n} + 0.5\ \Delta t\left[\mathbf{D}_i\left(\mathbf{U}_{i,j}^{n}\right) + \mathbf{D}_i\left(\mathbf{U}_{i,j}^{n+1/2}\right)\right], \tag{5}$$

where $\mathbf{U}_{i,j}^{n+1/2}$ is obtained as:

$$\mathbf{U}_{i,j}^{n+1/2} = \mathbf{U}_{i,j}^{n} + \Delta t\ \mathbf{D}_i\left(\mathbf{U}_{i,j}^{n}\right), \tag{6}$$

and the operator $\mathbf{D}_i(\mathrm{U}_{i,j})$ is defined as:

$$\mathbf{D}_i\left(\mathrm{U}_{i,j}\right) = -\frac{1}{\Delta x}\left(\mathbf{F}_{i+\frac{1}{2},j} - \mathbf{F}_{i-\frac{1}{2},j}\right) - \frac{1}{\Delta y}\left(\mathbf{G}_{i,j+\frac{1}{2}} - \mathbf{G}_{i,j-\frac{1}{2}}\right) + \mathbf{S}_0 + \mathbf{S}_f. \quad (7)$$

Fluxes are computed using the HLLC approximate Riemann solver (Toro, 1999), together with the correction proposed by Kurganov and Petrova (2007), which avoids non-physical velocity values at wet/dry fronts. The slope source term is discretized using a centered approximation (Vacondio et al., 2014), while the friction source term is discretized using the implicit formulation proposed by Caleffi et al. (2003).

The CUDA/C + + implementation of the model exploits the intrinsic parallelization of computations on GPU devices, thus guaranteeing fast execution times compared to serial codes. The basic work unit in CUDA is the *thread*, and many threads are organized into a *block*. In the present model, each thread corresponds to one cell used to discretize the computational domain, and each block consists of $K \times K$ cells. In the first implementation of this model (Vacondio et al., 2014), the domain was discretized by means of a Cartesian grid; information about neighboring cells/blocks is easily retrieved thanks to the correspondence between the physical position of a cell/block and its position on the two-dimensional array where data concerning that cell/block are stored in the memory.

Recently, the model has been extended to the case of a new type of non-uniform structured grids, called Block-Uniform Quadtree (BUQ) grids (Vacondio et al., 2017). In this case, the domain is still partitioned into blocks, each containing $K \times K$ cells with uniform spatial resolution within the block, but cells belonging to different blocks can be characterized by a different grid size. In particular, starting from blocks with the maximum resolution level (level 0), which contain cells with size $\Delta x_0 = \Delta x_{min}$, each level l contains cells with size $\Delta x_l = 2^l \cdot \Delta x_{min}$, up to the minimum resolution level L, where cells have size $\Delta x_{max} = 2^L \cdot \Delta x_{min}$ (in this work, L is assumed equal to 3). Neighboring blocks can differ by one resolution level, at most. The idea is similar to standard quad-tree partitioning (Liang and Borthwick, 2009), but each node of the quad-tree mesh corresponds to a block instead of a single cell. Operatively, the user must specify seeding points with assigned spatial resolution level as input to the simulation, and the model automatically generates the multi-resolution grid according to the procedure described by Vacondio et al. (2017). Memory allocation of blocks is not pre-established, as in the case of Cartesian grid, hence some additional data must be stored in order to retrieve information about neighboring cells belonging to different blocks during the computations. In particular, when adjacent cells have a different spatial resolution level, the natural neighboring interpolation procedure (Liang, 2011) is employed to reconstruct the conserved variables in the neighboring cells.

3. Local time stepping scheme

The key idea of the LTS strategy is to advance each cell with a time step as close as possible to its maximum permissible for stability. Most papers concerning LTS (e.g. Kleb et al., 1992; Crossley and Wright, 2005; Krámer and Józsa, 2007; Sanders, 2008; Dazzi et al., 2016) assume the local time step size in each cell as an integer multiple (in particular, a power-of-two multiple) of a common reference time step Δt, equal to the minimum in the whole domain. The procedure usually starts with the application of Eq. (4a) for each cell, in order to compute its own allowable time step, and of Eq. (4b), in order to find the global minimum time step. A temporal resolution level m (0, 1, 2 …) is assigned to each cell based on the condition: $2^m\ \Delta t \le \Delta t_{i,j} < 2^{m+1}\ \Delta t$; the coarsest level in the domain M is also computed. Then, the update procedure begins: all cells belonging to level m will be updated via 2^{M-m} steps of size $2^m \Delta t$; for example, cells with $m = 0$ will be advanced for 2^M steps of size Δt, while cells with $m = M$ will only perform one step of size $\Delta t_{max} = 2^M \Delta t$. This sequence of intermediate updates for each cell leads to a synchronized solution in the whole domain, then the procedure starts again.

The LTS algorithm presented in this paper is generally based on the same idea, with a major difference: it is assumed that all cells belonging to a *block*, which already share the same mesh size in the BUQ grid, also share the same local time step, while different blocks may be updated with different time steps. Hence, similarly to the spatial resolution level l (0, 1…L) of each block, it is possible to define the temporal resolution level m (0, 1…M) of each block. Notice that, while the spatial resolution is assigned by the grid generation procedure in the pre-processing stage and remains unchanged during the simulation, the temporal resolution levels need to be re-computed before each sequence of partial updates, because flow conditions and admissible time step sizes can change throughout the domain during the simulation.

The choice of adopting the same time step for all cells belonging to a block may appear to be inefficient in the LTS framework, because in the worst case only one cell may actually need to be advanced at such a slow pace. However, this block-based design offers a simple and well-performing parallel code, which fully exploits the parallel architecture of GPUs. It is worth recalling that the best performances on GPUs are obtained when cells belonging to a block are characterized by uniform features, so that special treatments are required only for cells on the border of a block, and code branching is minimized. Ultimately, this LTS implementation represents a natural extension of the spatial multi-resolution model, and introduces limited coding efforts and overheads.

In the following sub-sections, the main aspects of the LTS scheme are outlined. Section 3.1 deals with the definition of active blocks, with the assignment of a temporal resolution level to each block, and with the m-level distribution regularization. In Section 3.2, the core updating procedure for different time resolution blocks is covered, while Section 3.3 describes the rollback procedure, needed in cases where the CFL condition is violated. Finally, Section 3.4 reports on the expected advantages and overheads for a LTS block-based simulation on GPUs.

3.1. Temporal resolution level assignment to blocks

This Section describes the procedures required before the asynchronous time advancement of blocks may start. Notice that, when these operations are performed, all cells are synchronized to the common time level t^n.

The procedure for the temporal resolution assignment to each block starts with the execution of a *kernel* (i.e. a CUDA piece of code executed on the GPU) which computes the allowable $\Delta t_{i,j}$ in each cell, and determines the minimum Δt_b of each block b. Then, the CPU calculates the minimum Δt in the whole domain:

$$\Delta t = \min_b \left[\Delta t_b\right], \quad (8)$$

and assigns an initial guess m-level to each block as follows:

$$m_b = \mathrm{int}\left[\log_2\left(\frac{\Delta t_b}{\Delta t}\right)\right]. \quad (9)$$

It is worth recalling that, in the original GTS model, a Block Deactivation Optimization (BDO) was introduced in order to increase model efficiency (Vacondio et al., 2014); thanks to this procedure, only active blocks are processed during each time step. A block is defined as active if it contains at least one wet cell, or one cell which may become wet at

the end of the current time step. The kernel that computes the allowable time step also determines whether a block is wet. Then, "wettable" blocks are added to the set of active blocks by CPU processing, based on the simple idea that a wet/dry front may propagate at most over one cell per time step: hence, a dry block may become wet if one of its neighboring blocks has at least one wet cell on the confining border. For LTS, this condition is not sufficient, since up to 2^M steps may be performed before re-computing the set of active blocks. Hence, all dry blocks that are neighbors to wet blocks are activated, as shown in Fig. 1, where wet blocks and "wettable" blocks are active, while dry blocks are not processed during the current iteration. Since the set of active blocks is not redefined for a few steps, it is strongly recommended to impose an upper bound to M, which should be limited to fulfil the condition $2^M = K$ (if the block consists of $K \times K$ cells). For example, if 8×8 cell-blocks are employed, M should be limited to 3, so that 8 steps may be performed at most, and that the wet/dry front may not propagate over more than the 8 cells of the neighbor block. Obviously, for "wettable" blocks it is impossible to compute Δt_b because depth and velocity are null in each cell. Therefore, each "wettable" block is temporarily assigned the following m-level:

$$m_{wettable\_block} = \min_{nwb} \left(m_{nwb} \right), \tag{10}$$

where *nwb* indicates a neighboring wet block. As an example, let us consider the case of block j in Fig. 1: its two wet neighbors are characterized by m-levels equal to 1 and 2; block j will then be given $m_j = 1$. Exception to this rule is the case of a "wettable" block with a finer spatial resolution than the one of its wet neighbors; in this case, its temporal resolution level is refined too. As an example, block k in Fig. 1, whose wet neighbors are coarser and are characterized by $m = 1$, is assigned a temporal resolution level equal to 0.

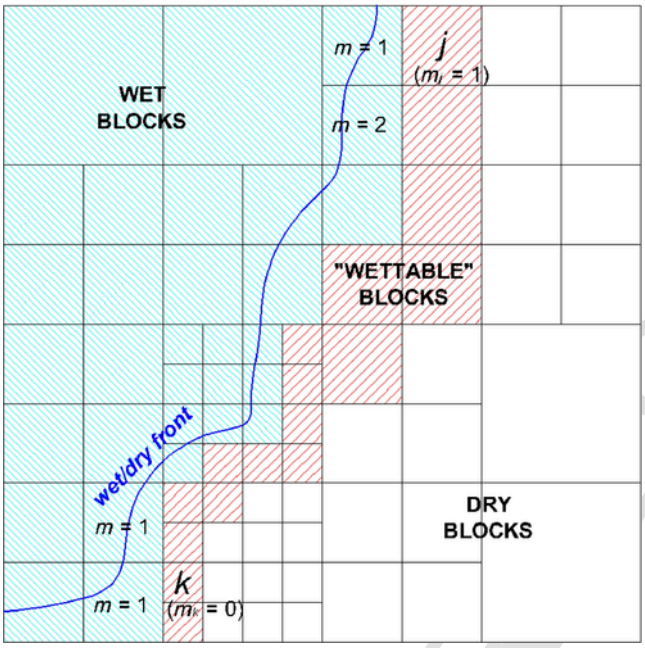

Figure 1. Sketch of the definition of active blocks: wet blocks are depicted in blue, "wettable" blocks in red, dry blocks in white. An example of first guess m-level assignment to "wettable" blocks j and k is also provided. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Many authors (e.g. Crossley and Wright, 2005; Krámer and Józsa, 2007) suggest that neighboring cells (or blocks, in this case) should differ by one temporal resolution level at most, in order to increase the robustness of the scheme. This also helps to develop an algorithm able to guarantee mass and momentum conservation (when no source terms are present) at interfaces between cells with different m-levels, while minimizing code branching and thus maintaining sufficient efficiency on GPUs. For this reason, after defining the set of active blocks, a regularization procedure to correct the initial guess m-levels is performed (see Algorithm 1). Iterations start from blocks with the maximum temporal resolution level ($m = 0$). For each block belonging to the current level, the temporal resolution of all neighboring blocks is checked, and, whenever the difference between the m-levels of the current and neighbor block exceeds one, the neighbor block is assigned a reduced m-level. An example of the application of this procedure is shown in Fig. 2a–b, which depict a few blocks of a BUQ grid with the corresponding temporal resolution level. After the initial guess m-level assignment (Fig. 2a), which obviously depends on the local grid size and flow variables, in this example two instances of blocks with one neighbor (indicated by a black arrow) whose m-level must be reduced (see Fig. 2b) can be identified.

Moreover, Crossley and Wright (2005) analyzed how waves propagate in neighboring cells characterized by different temporal resolution levels, and concluded that a "buffer" region needs to be defined in order to guarantee the propagation of waves travelling from maximum to minimum temporal resolution level regions. In the present model, the buffer region consists of a *buffer block*, whose m-level is artificially re-assigned according to Algorithm 2. In summary, each m-level region is broadened with one more line of buffer blocks. The allowable time step in these blocks is in theory double than the one actually used during the computations; however, the local artificial temporal resolution refinement ensures the correct propagation of waves and wet/dry fronts. An example of the m-level reassignment to buffer blocks is represented in Fig. 2c.

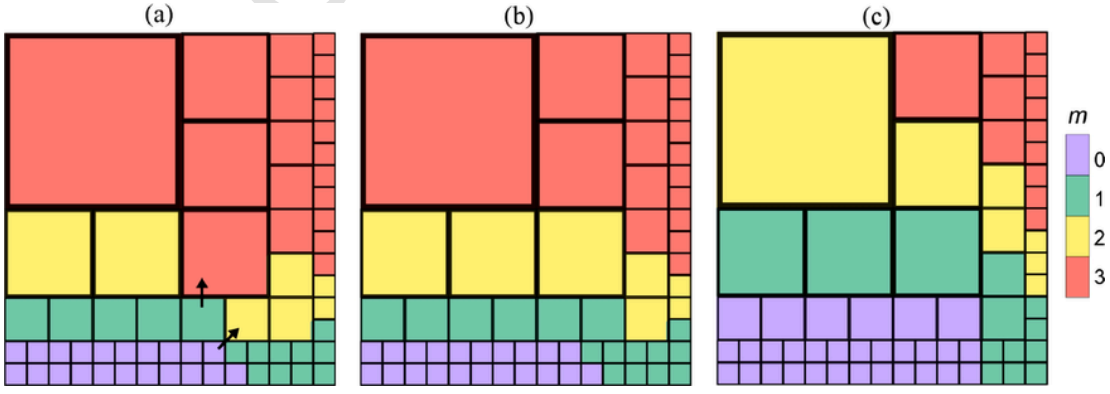

Figure 2. Example of temporal resolution level assignment to blocks: (a) initial guess, with arrows identifying neighboring blocks with a jump in the m-level larger than one; (b) after the regularization procedure of Algorithm 1; (c) after the m-level reassignment in buffer blocks according to Algorithm 2.

3.2. LTS update procedure

Algorithm 3 compares the GTS and LTS scheme pseudocodes (excluding input and output procedures). At the beginning of each iteration, the time step is computed, and active blocks are defined; in the LTS procedure, m-levels are also properly assigned, as already described in Section 3.1. Then, the update procedure may start. In the GTS version of the model, the solution is advanced by a single time step in the whole domain, and the iterations over time continue. In the LTS version of the scheme, instead, if M is the maximum temporal resolution level in the domain, a series of intermediate updates, which advances the solution from time level t^n to time level $t^n + \Delta t_{max}$ (where $\Delta t_{max} = 2^M \Delta t$) via 2^M steps, starts.

Let us consider the first-order version of the scheme. In this case, MUSCL extrapolation is not performed, and the left and right values required for computing fluxes at cell edges coincide with the conserved variables at cell centres; the solution in each cell is advanced via Eq. (3) with the local time step characterizing the current block. At each step s (0, 1 … $2^M - 1$), only blocks that fulfil a specific criterion are processed: all cells belonging to block b with level m are updated with a time step equal to $\Delta t_b = 2^m \Delta t$, if s is an integer multiple of 2^m. The procedure is sketched in Fig. 3 for a simple one-dimensional (1D) case with six blocks and $M = 2$.

A special treatment is necessary for cells on the border of a buffer block, whose neighboring cells are characterized by a different temporal resolution level. Although some authors (e.g. Kesserwani and Liang, 2015; Saetra et al., 2015) adopt temporal interpolation procedures at these interfaces, which in turn require corrections to guarantee mass and momentum conservation, the present scheme adopts an intrinsically conservative strategy to handle these interfaces. Figure 4 shows two cells belonging to a buffer block with $m = 0$, and two cells on its neighboring block with $m = 1$. Without loss of generality, a 1D representation with cells characterized by the same mesh size and only two temporal resolution levels is chosen for simplicity. During the first step ($s = 0$), all cells are updated, and fluxes are computed based on the conserved variables values at time t^n, as usual. During the second step ($s = 1$), however, only cells belonging to the block with $m = 0$ must be advanced in time. While inner cell $i - 1$ has both west and east neighbor values available for flux computation at time t^{n+1}, border cell i lacks a synchronized value to the east. For this reason, on its eastern edge, which coincides with the block border, values at time t^n are reused for the flux computation. In this way, the sum of the eastern fluxes of cell i computed at times t^n and t^{n+1} (which are then multiplied by Δt), and the western flux of cell $i + 1$ at t^n (which is multiplied by $2\Delta t$), are the same, and the scheme is conservative. Notice that on the western edge of cell i, flux computation is still performed based on values at t^{n+1}. This strategy is possible because arrays for fluxes are not allocated in the present model, and the computation of the same flux at the edge between two cells is repeated twice (once for each thread) for efficiency reasons (as discussed by Vacondio et al., 2014). Hence, the only drawback of this procedure is the necessity of allocating an additional array in memory, where conserved variables from the previous time step in cells on the block boundaries are stored and retrieved when necessary. The detailed implementation (neglecting GPU parallelization) is reported in Algorithm 4, which corresponds to lines 7–13 of Algorithm 3b.

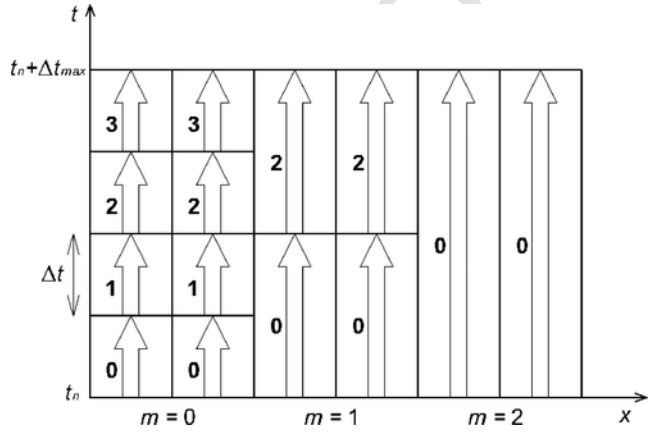

Figure 3. First-order LTS scheme: sketch of the update procedure for six blocks with different temporal resolution levels ($M = 2$). White arrows represent the flux computation and time integration operations, performed at the step specified in bold type.

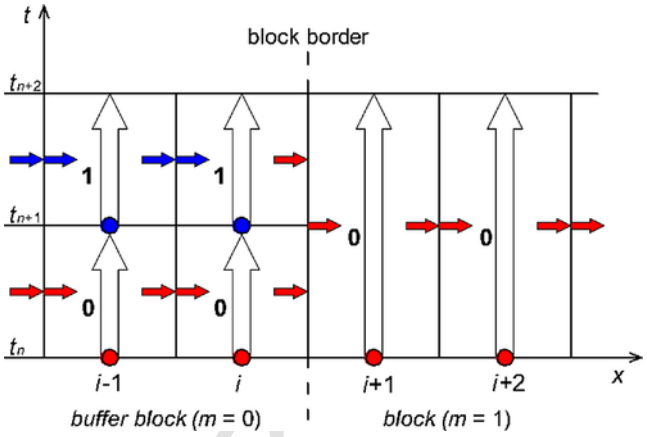

Figure 4. First-order LTS scheme: sketch of the update procedure for cells near the border of a buffer block ($M = 1$). Red and blue dots represent variables at cell centers at t_n and t_{n+1}. Red and blue arrows represent flux computation based on the variables of the corresponding color. White arrows represent time integration, performed at the step specified in bold type. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The second-order version of the scheme is presented next. In this case, two sub-steps are performed during each time advancement, and two kernels are launched during each of these two sub-steps: one, which executes the MUSCL extrapolation, and one, which performs flux computation and time integration. Finally, an additional kernel sums the contributions retrieved from the two half time steps, according to Eq. (5).

In the LTS implementation of the scheme, each pair of sub-steps of the second-order scheme can be compared to two consecutive steps of the first order scheme, provided that appropriate criteria are defined in order to distinguish whether blocks belonging to each m-level are to be processed during the current sub-step. Time advancement from t^n to $t^n + \Delta t_{max}$ is schematized in Algorithm 5 (which substitutes lines 7–13 of Algorithm 3b). Fig. 5 depicts three blocks with different m-levels ($M = 2$), specifying the step (0, 1, 2, 3) and sub-step (A, B) at which each block is processed, according to Algorithm 5. Fig. 5a represents MUSCL extrapolation, while Figure 5b shows flux computation and time integration.

Cells lying on the border of buffer blocks are updated asymmetrically, similarly to the first-order scheme. At sub-steps for which the synchronized neighboring values are missing, MUSCL extrapolation is skipped in these cells: on the border block edge, extrapolated values at the previous sub-step are reused for flux computation; on the inner edge, accuracy is locally reduced to first-order in space (e.g. values at the cell center are used for flux computation), in order to avoid interpolations. Results presented in the following section show that this simplification does not significantly impair the accuracy of the solution.

Finally, a short remark about open boundary conditions (BCs) needs to be made. For each open BC (discharge, rating curve and/or water

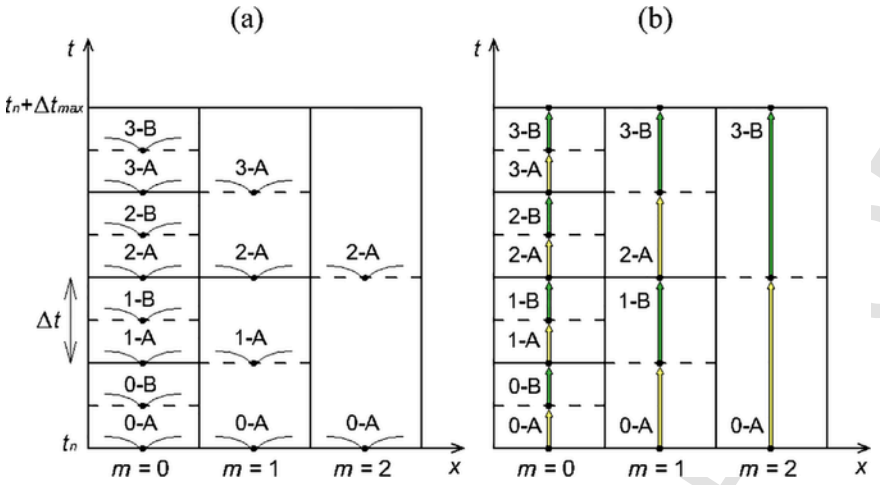

Figure 5. Second-order LTS scheme: sketch of the update procedure for three blocks with different temporal resolution levels ($M = 2$). (a) MUSCL extrapolation procedure, and (b) flux computation and time integration, each performed at the specified step (0…3) and sub-step (A, B). Dashed lines indicate the half time steps.

level) the minimum allowable temporal resolution level m_{bc} is assigned to all blocks affected by the given open BC. The specific CUDA kernel which assigns the prescribed quantities (discharge, water elevation, rating curve) at open BCs cells is executed only at steps for which blocks with $m = m_{bc}$ are processed. For details about BCs handling in the model, the reader is referred to Vacondio et al. (2014). From a practical point of view, this is not a relevant feature in the LTS model, because the number of the cells to which an open boundary conditions is assigned is usually negligible compared to the total number of cells used to discretize the domain.

3.3. Roll back procedure

According to Algorithm 3b, the time step computation and the m-level assignment (lines 2–6) are performed before the inner LTS loop (lines 7–13) which advances the solution from t^n to $t^n + \Delta t_{max}$. This means that the temporal resolution level (hence, the time step size) is maintained fixed for a few steps. However, during the computations, flow field conditions vary; in some cases, velocity and/or celerity might increase, and the time step can no longer fulfil the CFL condition, generating instabilities. For this reason, a procedure to roll back simulation time is implemented in the present model. An additional array needs to be allocated, where conserved variables are stored at time t^n. After each partial time integration, the CFL condition is re-evaluated in each cell based on the updated values of conserved variables, assuming $Cr = 1$. If this check highlights potential instabilities somewhere in the domain, the LTS loop is stopped, and partial computations already performed are discarded. Then, the loop begins again starting from the values stored in the rollback array and assuming a halved time step size everywhere. The prevention of potential simulation crashing compensates for the slight overload induced by this procedure.

3.4. Efficiency of the LTS algorithm on the GPU

A comparison between the computational efficiency of the GTS and LTS schemes can be performed by evaluating the execution time ratio S_U, defined as the ratio of the GTS execution time to the LTS execution time for each simulation.

This factor strongly depends on the test case under consideration and on the grid refinement, and the maximum expected time ratio for a given test case can be theoretically estimated by analyzing the amount of computational operations associated with the two versions of the code. For this analysis, let us distinguish between the "dt" piece of code (lines 2–3 in Algorithm 3a for GTS, or 2–6 in Algorithm 3b for LTS), and the "update" piece of code (lines 4–5 in Algorithm 3a for GTS, or 7–14 in Algorithm 3b for LTS, which are substituted by Algorithm 5 for the second-order scheme). Preliminary investigations show that, in the GTS version of the code, the "dt" stage takes roughly 30% of the total simulation time for the first-order scheme, and 15% for the second-order scheme; accordingly, the "update" stage takes roughly 70% and 85% of the total execution time for the first- and second-order schemes, respectively.

As regards the "dt" stage, it can be observed that, in the LTS version of the scheme, the time step computation and the BDO procedure are performed only once every 2^M steps, while the GTS scheme repeats these operations at every time step. Therefore, the LTS version is in theory 2^M times faster than the GTS scheme in performing the "dt" stage, even if the additional processing for the m-level assignment (see Section 3.1) reduces the actual time ratio for this part of the code.

On the other hand, the most important computational saving of the LTS scheme is associated with the "update" stage, since a subset of blocks is processed fewer times in the LTS code than they would be in the GTS scheme. An estimate of the achievable theoretical time ratio for the "update" piece of code can be quantified by dividing the total number of cells (blocks) processed in the GTS simulation by the same value in the LTS simulation, assuming that the update operations performed on each block require roughly the same computational time (Kleb et al., 1992). In the example depicted in Fig. 3, fourteen update operations are performed instead of twenty-four, which results in a theoretical time ratio (for the "update" stage only) equal to 1.7. However, this simple computation does not take into account the overheads associated with the LTS scheme. First of all, a few operations must be performed inside each kernel of time advancement, in order to distinguish whether a block must be processed during the current time step. In addition to this, managing interfaces between blocks with different temporal resolution levels requires code branching, which may slow down computations on the GPU. The whole roll back procedure, described in Section 3.3, is also added to the code. All these overheads lead to an actual time ratio smaller than the theoretical one.

In summary, the theoretically achievable time ratio can be evaluated by combining the contributions from the "dt" and "update" stages as follows:

$$S_U^{theor} = \left[\frac{1}{2^M} \cdot T_{dt}^* + \left(\frac{1}{N} \sum_{m=0}^{M} \frac{N_m}{2^m} \right) \cdot T_{update}^* \right]^{-1}, \tag{11}$$

where N_m is the number of cells (or blocks) belonging to each m-level, N is the total number of cells (or blocks) in the domain, and T^*_{dt} and T^*_{update} are the execution times for the "dt" and "update" stages, normalized to the total execution time. Considering the example depicted in Fig. 3 ($M = 2$), the theoretical time ratio is equal to 2.05 for the first-order scheme (assuming $T^*_{dt} = 0.3$, and $T^*_{update} = 0.7$), and to 1.86 for the second-order scheme ($T^*_{dt} = 0.15$, and $T^*_{update} = 0.85$).

Finally, an additional aspect should be considered in the assessment of the computational efficiency of the model. On GPUs, tasks on different blocks are executed in parallel by the different processors the hardware architecture consists of. If the number of active blocks is larger than the available cores, blocks need to be queued by the GPU's scheduler, while, when simulations with a smaller number of active blocks are performed, the computational capabilities of the GPU may not be fully exploited. In this latter case, the increased model efficiency due to parallel computations on the GPU is somehow impaired by the CPU-GPU communication overheads (see Vacondio et al., 2017). Hence, the typical scalability of GPUs (i.e. the computational time, normalized to the total number of cells, decreases with increasing the number of processed cells, and remains almost constant after reaching a threshold value, which depends on the GPU type) should also be taken into account when evaluating the computational efficiency of both GTS and LTS schemes.

In any case, a quantitative analysis of the overheads and savings of the LTS scheme is discussed for the first two test cases in Section 4.

4. Numerical tests

In this Section, a comparison between the performances of the original GTS and the novel LTS model implementation is performed based on three theoretical test cases and on two field-scale practical applications. All simulations were run using a K40 Tesla® GPU. For all test cases, the Courant number was assumed equal to 0.8. Moreover, the BUQ grid was formed by 8×8-cell blocks; hence, the temporal resolution level was limited to $M = 3$ for all simulations.

4.1. Vortex test case

The first numerical experiment considers the steady-state test case of a vortex flow with analytical solution (Sanders and Bradford, 2006). The vortex circulates clockwise on a flat frictionless bottom, and its motion can be described by means of the following relations (assuming that the coordinate system origin coincides with the vortex center):

$$h = h_0 + \frac{U_0^2}{4g}\left[1 - \frac{2r}{r_0} exp\left(\frac{-2r}{r_0}\right) - exp\left(\frac{-2r}{r_0}\right)\right], \tag{12a}$$

$$u = \frac{U_0 y}{r_0} exp\left(\frac{-r}{r_0}\right), \quad v = \frac{-U_0 x}{r_0} exp\left(\frac{-r}{r_0}\right), \tag{12b}$$

where $r = \sqrt{x^2 + y^2}$ is the distance from the vortex center, and the values attributed to the other parameters are as follows: $h_0 = 10$ m, $U_0 = 1.5$ m/s, $r_0 = 100$ m. The domain is extended up to $r = 3000$ m.

The maximum resolution (Δx_{min}) is imposed in the area close to the vortex center (approximatively up to $r = 500$ m), surrounded by a smooth transition up to the minimum resolution Δx_{max} in the outer region of the domain. A detail of the resulting multi-resolution grid is shown in Fig. 6, which represents each 8×8-cell block as a square element. The temporal resolution level for the LTS simulation is dictated by the grid size for this test case. Thus, the area characterized by the maximum temporal resolution level ($m = 0$) coincides with the finest grid size region, plus the circle of neighboring blocks (with halved grid size) required by the numerical scheme as a buffer region to ensure correct wave propagation. The same holds for the areas with coarser grid size. The m-level distribution is also represented in Fig. 6, where different colors identify blocks with homogeneous temporal resolution level.

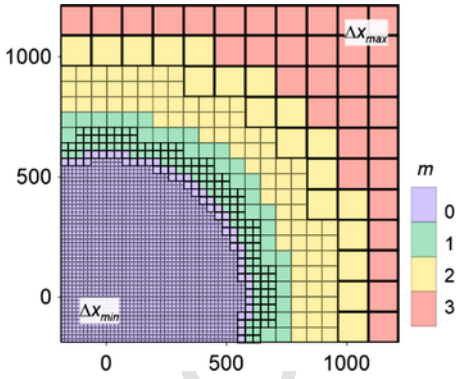

Figure 6. Vortex test case: detail of the non-uniform grid (each square element is a 8×8-cell block), and temporal resolution level distribution.

Sixteen simulations were run, assuming four different test configurations (see Table 1), each performed with both models (LTS and GTS), and with both the first- and the second-order accurate version of the scheme. Minimum and maximum grid sizes were in the range 1–8 m, and 8–64 m respectively. The analytical solution was imposed as initial condition, and the simulation was run for a physical time in the range 500–4000 s, in order to ensure that the same number of update operations was performed for all test configurations.

As an example, Figs. 7 and 8 compare the results of the GTS and LTS simulations performed with the second-order accurate scheme for test configuration B. Fig. 7 reports the contour maps of water depth and velocity magnitude $|u| = \sqrt{u^2 + v^2}$, while Fig. 8 shows the profiles of the same variables along the $y = x$ line. A quantitative comparison regarding the agreement of numerical results with the exact solution can be performed by computing the non-dimensional L_2 error norms of water depth and velocity components:

Table 1
Vortex test case: L_2 norms and time ratio S_U for four configurations characterized by different minimum and maximum resolution, number of cells, and simulation time, obtained from GTS and LTS first-order (FO) and second-order (SO) simulations.

Test		A	B	C	D
Δx_{min} (m)		1	2	4	8
Δx_{max} (m)		8	16	32	64
# cells (10^6)		1.577	0.411	0.102	0.029
T (s)		500	1000	2000	4000
L_2 (h)	GTS-FO	2.81E-05	6.94E-05	1.17E-04	1.40E
	LTS-FO	2.81E-05	6.93E-05	1.16E-04	1.40E
	GTS-SO	3.86E-06	4.19E-06	1.16E-05	4.95E
	LTS-SO	3.87E-06	4.19E-06	1.16E-05	4.95E
L_2 (u) = L_2 (v)	GTS-FO	1.98E-03	5.49E-03	1.13E-02	1.69E
	LTS-FO	1.98E-03	5.50E-03	1.14E-02	1.69E
	GTS-SO	3.20E-05	1.66E-04	8.91E-04	3.84E
	LTS-SO	4.25E-05	1.76E-04	9.00E-04	3.85E
S_U (-)	FO	1.53	1.51	1.50	1.38
	SO	1.44	1.42	1.28	1.23

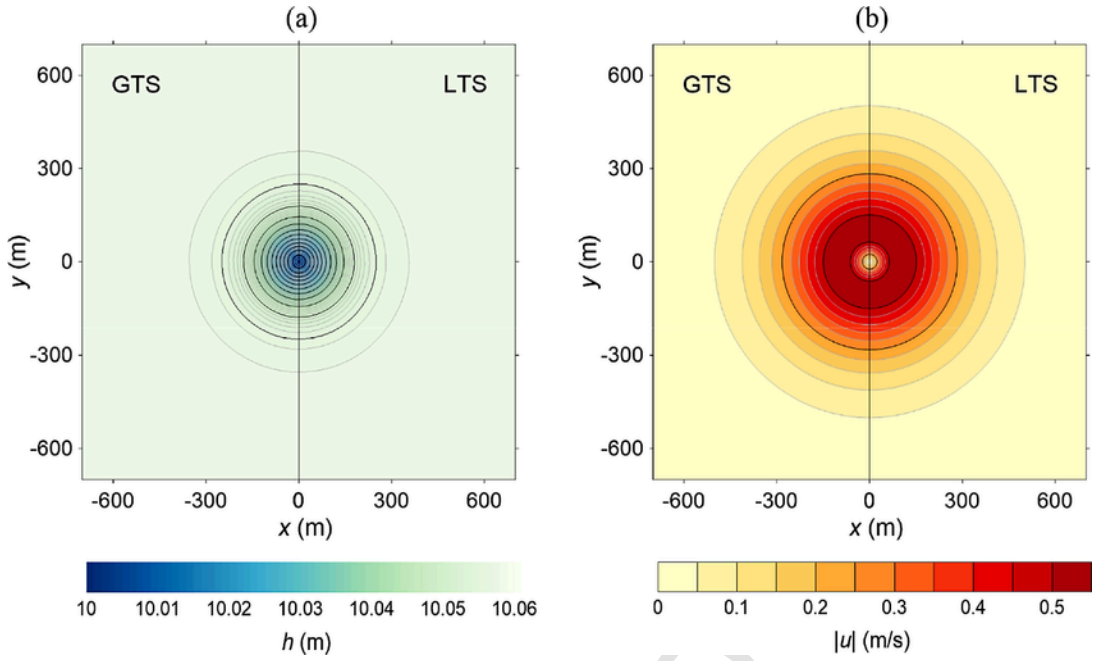

Figure 7. Vortex test case (configuration B, second-order scheme): depth (a) and velocity magnitude (b) contour maps at the end of the GTS and LTS simulations (shown side by side).

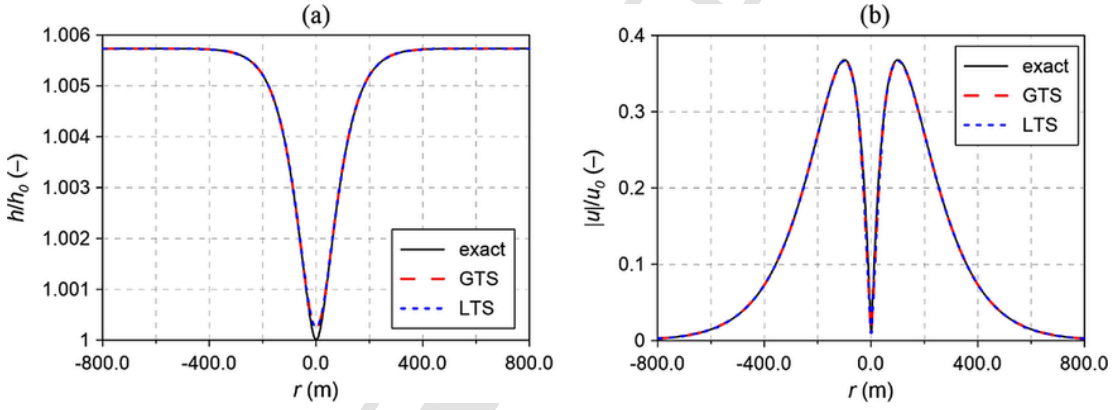

Figure 8. Vortex test case (configuration B, second-order scheme): comparison between exact and numerical depth (a) and velocity magnitude (b) profiles along the $y = x$ line.

$$L_2(h) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{h_{i,num} - h_{i,exact}}{h_0}\right)^2}, \tag{13}$$

where N is the number of cells in the domain, and subscripts *num* and *exact* refer to the numerical and analytical solutions. The same expression can be used to calculate $L_2(u)$ and $L_2(v)$ by adopting U_0 as reference value for normalization.

The L_2 error norms computed for the sixteen simulations are reported in Table 1. As expected, errors increase with the grid size for both first- and second-order simulations, and second-order error norms are at least one order of magnitude smaller than first-order ones for all configurations. No differences can be appreciated between LTS and GTS water depth error norms, which remain in the range 10^{-5}–10^{-6}, close to machine precision, for second-order simulations, while velocity error norms reach the order of magnitude 10^{-4}–10^{-3} for the coarsest grid sizes, and show small differences between LTS and GTS values. For first-order simulations, LTS and GTS error norms practically coincide.

Finally, Table 1 reports the execution time ratios for all test configurations. Time ratios achieved by the first-order scheme, reaching 1.53, are always slightly larger than the ones obtained from the second-order scheme, which are limited to 1.44. The best performances can be observed for tests with the finest grid size, even if the time ratio is not expected to depend much on the grid size and number of cells, due to the steadiness of the test case. In fact, the achievable theoretical time ratio is approximately the same for all configurations, and is equal to 1.86 and to 1.6 for the first- and second-order schemes, respectively.

A detailed analysis regarding the computational times associated with different parts of the code is reported for configuration A ($\Delta x_{min} = 1$ m). The theoretical time ratios for the "dt" and "update" stages are equal to 8 and 1.4, respectively. Fig. 9 shows the execution times of the two parts of the code for first- and second-order schemes (both GTS and LTS); values are normalized to the total execution time of the GTS second-order simulation, which is the longest one. In the first-order scheme, the time ratio associated with the "update" piece of code is 1.26; this means that the overhead associated with LTS branching in the update kernels is only 10% for this simulation. On the other hand, the time ratio for the "dt" stage is equal to 3.31, and the overhead associated with the additional operations performed for the *m*-levels assignment is widely compensated by the fact that this piece of code is executed only once every eight steps. The combination of the computational savings associated with the two parts of the code results in an actual time ratio equal to 1.53. Therefore, the LTS overheads in-

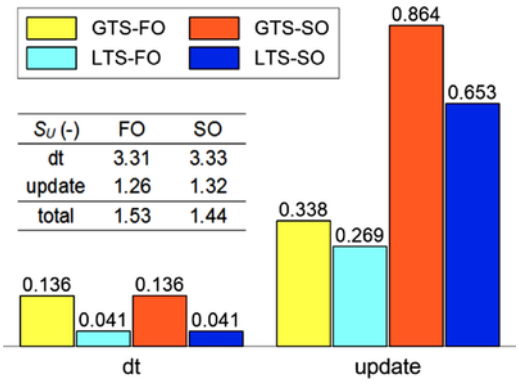

Figure 9. Vortex test case (configuration A): execution times of "dt" and "update" pieces of code for GTS and LTS simulations in the first (FO) and second-order (SO) version of the scheme. Values are normalized to the total run time of the GTS-SO simulation. The table in the insert reports the total and partial time ratios.

crease the total simulation time by roughly 18% compared to the theoretically achievable value for this test. For the second-order scheme, due to the heavier "update" operations, the "dt" piece of code is computationally less relevant than for the first-order one. In fact, while both first- and second-order schemes require approximately the same execution time for the "dt" operations, the "update" stage for the second-order scheme takes more than twice the time required for the first-order scheme (this is true for both GTS and LTS), due to the larger number of executed kernels. For this reason, despite similar partial time ratios, the total execution time ratio for the second-order scheme, equal to 1.44, is slightly smaller than the value obtained from first-order simulations, and the LTS overheads can be estimated to be only 10% of the total execution time.

4.2. Circular dam-break

The model was then tested by simulating the classical wet circular dam-break problem (Liska and Wendroff, 1997). The domain $[-25, 25]\,\text{m} \times [-25, 25]\,\text{m}$ is characterized by a horizontal frictionless bottom. A 10 m-high cylindrical water column, with radius $R = 10\,\text{m}$, is centered in the domain, and is surrounded by 1 m-deep still water. Due to the cylindrical symmetry of the problem, a reference solution can be obtained by deriving the inhomogeneous 1D system of the SWE in radial geometry (Toro, 2001) and solving it with a very fine mesh ($\Delta x = 0.005\,\text{m}$). The grid for the 2D LTS and GTS simulations was generated by forcing the maximum resolution $\Delta x_{min} = 0.025\,\text{m}$ alongside the initial discontinuity ($9\,\text{m} < r < 11.5\,\text{m}$), and by imposing the halved

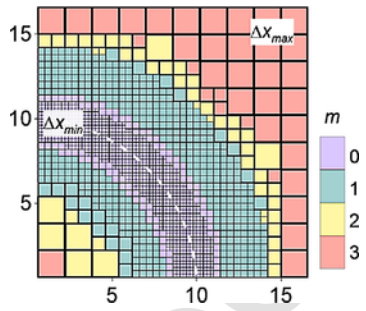

Figure 10. Circular dam-break test case: multiresolution grid (each square element is a 8×8-cell block), and temporal resolution level map at $t = 0.4\,\text{s}$. The dashed white line identifies the initial discontinuity position.

mesh size 0.05 m in the regions $6.5\,\text{m} < r < 9\,\text{m}$ and $11.5\,\text{m} < r < 14.5\,\text{m}$. The automatic grid generation procedure created the remaining transitions up to the minimum resolution $\Delta x_{max} = 0.2\,\text{m}$. Fig. 10 represents a detail of the resulting mesh, which consists of $0.465 \cdot 10^6$ cells. Simulations were run until the physical time $t = 1\,\text{s}$.

Fig. 11 compares the water depth profiles along the radial direction at $t = 0.4\,\text{s}$ obtained by the LTS and GTS simulations with the reference solution. Results from both first- and second-order schemes agree well with the pseudo-exact solution, the latter showing a closer agreement, as expected. Notice that, even if the shock wave is propagating over a region with $m = 1$ (see the temporal resolution level distribution at $t = 0.4\,\text{s}$ in Fig. 10), the LTS solution accuracy is not degraded compared to GTS results. On the contrary, dimensional L_2 error norms for water depth and velocity magnitude (reported in Table 2) show that the LTS model provides even slightly more accurate results than GTS, in particular as regards the first-order scheme.

Also for this test case, the execution time ratio, reported in Table 2, is slightly larger for the first-order scheme ($S_U = 1.49$) than for the second-order scheme ($S_U = 1.39$). The achievable theoretical values are equal to 1.81 and 1.56 for the first- and second-order schemes, respectively (in particular, 8 for the "dt" stage, and 1.36 for the "update" stage). As can be inferred from Fig. 12, the "update" time ratio is equal to 1.22 for the first-order simulation, and to 1.28 for the second-order one, confirming the 6–10% overhead induced by the LTS implementation within this piece of code, which reduces the achieved time ratio compared to the theoretical one. Similarly to the previous test case, the execution times for the "dt" operations show that the overhead induced by the procedure for m-level assignment in the LTS scheme is largely compensated by the computational savings due to the fewer "dt" executions. Globally, the LTS overhead is 10–18% for this test case too.

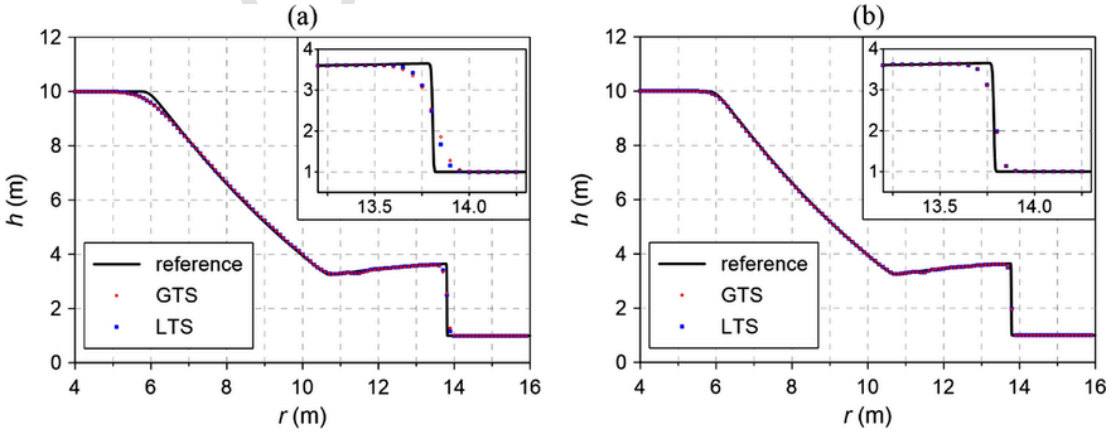

Figure 11. Circular dam-break test case: reference and numerical (LTS and GTS) water depth profiles for first- (a) and second-order (b) schemes; inserts zoom on the shock wave.

Table 2
Circular dam-break test case: dimensional L_2 error norms for water depth and velocity magnitude at $t = 0.4$s for GTS and LTS in the first-order (FO) and second-order (SO) version of the scheme, and time ratio S_U.

	FO		SO	
	GTS	LTS	GTS	LTS
L_2 (h) (m)	8.61E-02	7.96E-02	5.86E-02	5.86E-02
L_2 ($\|u\|$) (m/s)	2.15E-01	1.94E-01	1.43E-01	1.43E-01
S_U (-)	1.49		1.39	

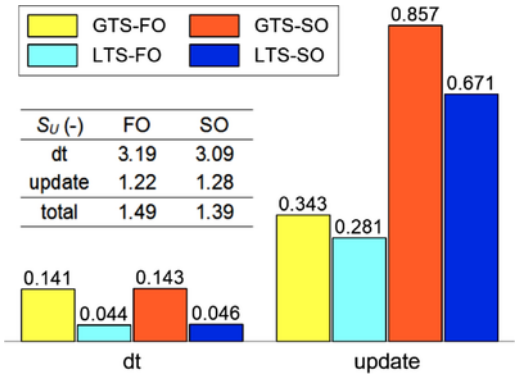

Figure 12. Circular dam-break test case: execution times of "dt" and "update" pieces of codes for GTS and LTS simulations in the first- (FO) and second-order (SO) version of the scheme. Values are normalized to the total runtime of the GTS-SO simulation. The table in the insert reports the total and partial time ratios.

4.3. Thacker test case

The LTS scheme was further validated by simulating the periodic oscillation of a water volume in a frictionless paraboloidic basin (Thacker, 1981), which involves wetting/drying and non-flat topographies. The bottom can be described by means of the following equation:

$$z = z_0 \left(\frac{x^2 + y^2}{L^2} - 1 \right), \tag{14}$$

where z_0 is the depth of the vertex, and L is the radius at $z = 0$ (see Fig. 13a). The water volume, initially at rest and paraboloidic, expands and contracts periodically due to gravity. The exact solution for this test case is:

$$\begin{cases} \eta\ (x,y,t) = z_0 \left\{ \frac{\sqrt{1-A^2}}{1-A\cos\omega t} - 1 - \frac{x^2+y^2}{L^2} \left[\frac{1-A^2}{(1-A\cos\omega t)^2} - 1 \right] \right\}, \\ u\ (x,y,t) = \frac{1}{1-A\cos\omega t} \left(\frac{1}{2}\omega x A \sin\omega t \right), \\ v\ (x,y,t) = \frac{1}{1-A\cos\omega t} \left(\frac{1}{2}\omega y A \sin\omega t \right), \end{cases}$$

where ω and A are defined as:

$$\omega = \frac{\sqrt{8gz_0}}{L}, \quad A = \frac{(\eta_0 + z_0)^2 - (z_0)^2}{(\eta_0 + z_0)^2 + (z_0)^2}. \tag{16}$$

The test parameters are set as follows: $z_0 = 50$m, $\eta_0 = 10$m, $L = 1000$m, which correspond to a period of oscillation equal to 50s. The domain is extended up to $r = 1500$m. The maximum resolution $\Delta x_{min} = 1$m is assigned to the area subject to wetting/drying (850m $< r <$ 1100m), and near the paraboloid center ($r < 30$m), while the grid generation procedure automatically creates transitions up to $\Delta x_{max} = 8$m. The final mesh (Fig. 14) consists of $2{\cdot}10^6$ cells, and the simulation time includes four complete oscillations ($t_{final} = 400$s).

Fig. 13b presents a comparison between analytical and numerical results (for the second-order scheme); in particular, slices of water surface elevation along the x-axis at selected times are depicted. GTS and LTS profiles practically coincide, confirming the accuracy of the scheme even in the presence of wetting/drying and bottom slope source terms. Dimensional L_2 error norms, reported in Table 3 for $t = 250$s, show that LTS results are even slightly better than GTS. The obtained time ratios reported in Table 3 (equal to 1.47 and 1.32 for first- and second-order simulations, respectively) are slightly smaller than the corresponding theoretical values (estimated to be equal to 1.70 and to 1.43), due to the overheads introduced in the LTS scheme, as already discussed for the previous test cases.

The same case was also simulated with friction, in order to assess the LTS scheme accuracy when both source terms are present. For this case, no analytical solution is available, therefore LTS results are only compared with GTS. Manning's roughness coefficient is set equal to 0.15 $\mathrm{m}^{-1/3}$s. Fig. 15 reports the water surface elevation trend in time at point (0.0; 0.0) for both frictionless and non-frictionless simulations. While oscillation amplitude and frequency remain constant in the former case, in the latter case water level oscillations are dampened, as expected. Again, LTS and GTS results are almost overlapping. In this case, time ratios are equal to 1.74 and 1.65 for the first- and second-order simulations, respectively (in contrast with theoretical values of 2.11 and 1.86).

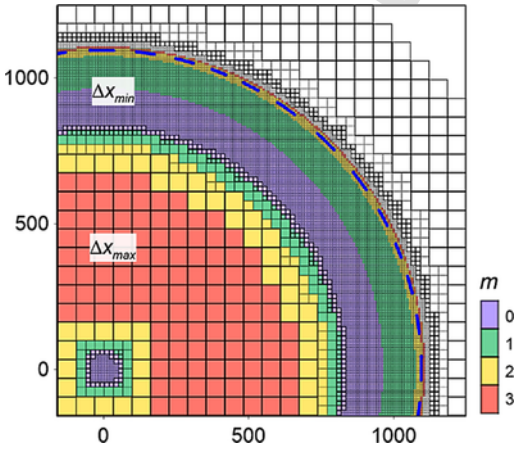

Figure 13. Thacker test case: detail of the multiresolution grid (each square element is a 8×8-cell block), and temporal resolution level map at $t = 250$s. The dashed blue line identifies the wet-dry front.

4.4. Parma–Baganza test case

For field-scale test cases, computational savings are expected to be considerable when the LTS scheme is used to simulate a large domain where only a small region is discretized with a very fine mesh. This might be the case of the presence of bridge piers in a riverbed, which require a high level of detail. The Baganza River (Northern Italy), which is crossed by a bridge 500m upstream of its confluence in the Parma River, was considered for this test case. The bridge is characterized by four 3×10m round nosed piers. The bathymetry is shown in Fig. 16: the 3.4km-long final branch of the Baganza River was modelled, together with the 2km- and 1km-long branches of the Parma River upstream and downstream of the confluence, respectively. The non-uniform BUQ grid is characterized by $\Delta x_{min} = 0.25$m, imposed only at the bridge site, which gradually transitions to $\Delta x_{max} = 2$m in

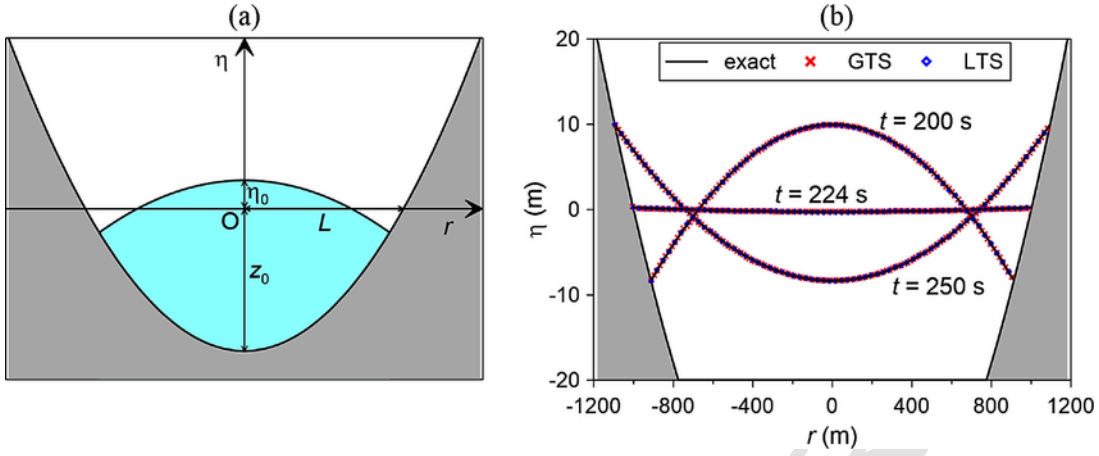

Figure 14. Thacker test case: (a) sketch of the test geometry, (b) analytical and numerical (second-order LTS and GTS schemes) water surface elevation slices along the *x*-axis at selected times.

Table 3
Thacker test case: dimensional L_2 error norms for water surface elevation and velocity magnitude at $t = 250$ s for GTS and LTS in the first-order (FO) and second-order (SO) version of the scheme, and time ratio S_U.

	FO		SO	
	GTS	LTS	GTS	LTS
L_2 (η) (m)	5.38E-01	4.35E-01	3.76E-02	3.39E-02
L_2 ($\|u\|$) (m/s)	3.73E-01	3.70E-01	6.69E-02	6.05E-02
S_U (-)	1.47		1.32	

the remainder of the domain, as shown in the insert in Fig. 16. The total number of cells is equal to $0.536 \cdot 10^6$.

As upstream boundary condition, a discharge hydrograph is assigned at the inflow of the Baganza River: the initial value is 100 m^3/s, which is gradually increased to 300 m^3/s within one hour; the simulation is then prolonged for three more hours. Similarly, a discharge hydrograph with initial and final values equal to 200 and 400 m^3/s is imposed as upstream boundary condition for the Parma River. Downstream, a constant water depth is assigned. Initial conditions are obtained from a preliminary steady-state simulation with the initial discharge values. Manning's roughness coefficient is assumed equal to 0.04 $m^{-1/3}s$.

Table 4 reports the execution time ratios for both first- and second-order simulations. Values of approximately 2.7 confirm the achievable reduction in execution time due to the adoption of the LTS version of the model. In this case, the theoretical values are equal to 3.83 for the first-order scheme, and to 3.45 for the second-order scheme. For real test cases with boundary conditions and wet/dry fronts, code branching probably enhances the overheads associated with the LTS scheme, and the total overhead (22–27%) is larger than the one obtained from the simulation of the theoretical test cases previously analyzed (10–18%).

In addition to the greater computational efficiency, the choice of the LTS scheme does not degrade the quality of results compared to GTS. The root mean square error (RMSE) between GTS and LTS water surface elevations, when steady-state conditions were achieved, was also computed:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\eta_{i,GTS} - \eta_{i,LTS}\right)^2}, \tag{17}$$

and was observed to be almost negligible (see Table 4).

Finally, as an example of results, Fig. 17a–b show the water surface elevation and velocity maps around one of the bridge piers (obtained from the second-order LTS simulation). The adoption of a mesh with local high resolution allows predicting the local 2D flow field in detail; in particular, the stagnation points upstream of the pier and the recirculation region downstream can be resolved. For comparison, Fig. 17c depicts the velocity field for a simulation performed without imposing a local grid refinement: in this case, the pier geometry is only roughly described, and the velocity field is not accurately captured. This justi-

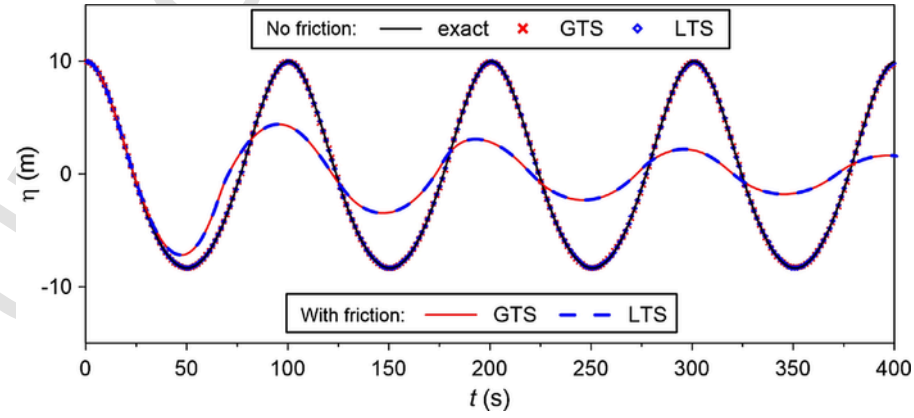

Figure 15. Thacker test case: water surface elevation trend in time at point (0; 0) for the frictionless (with exact solution) and non-frictionless simulations. Only second-order LTS and GTS results are reported.

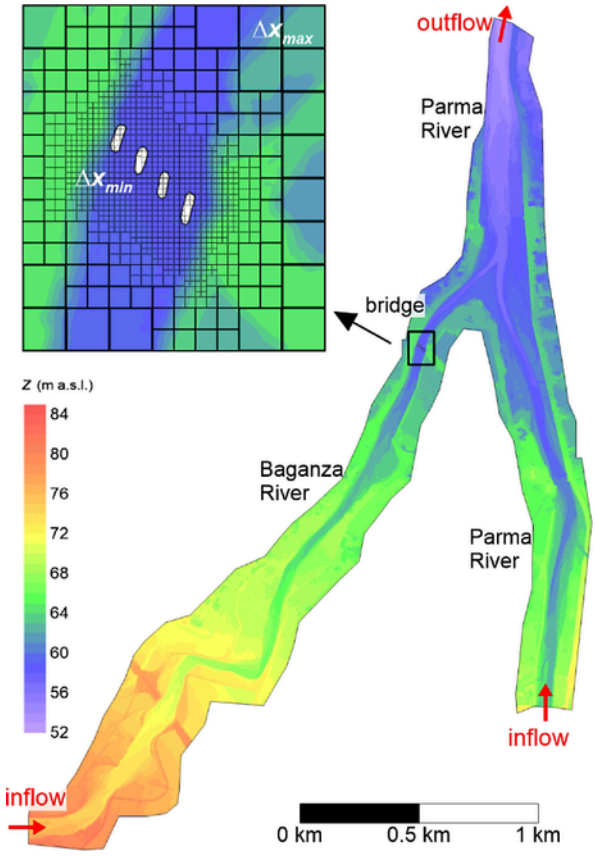

Figure 16. Bathymetry for the Parma-Baganza test case. The insert zooms on the bridge site (bridge piers are shown in white), and represents the multi-resolution grid (each square is a 8×8-cell block).

Table 4
Parma–Baganza test case: time ratio S_U and RMSE between GTS and LTS results, at the end of the simulation, for first-order (FO) and second-order (SO) schemes.

	FO	SO
S_U (-)	2.77	2.66
RMSE (m)	5.1E-03	6.3E-03

fies the choice of a locally refined mesh when the flow field near the bridge is of interest, which makes the adoption of a LTS scheme particularly convenient.

4.5. Flooding scenario due to levee breaching

Finally, the achievable performance improvement of the LTS scheme over the GTS scheme was assessed based on the simulation of the flooding scenario induced by a hypothetical levee breach on the Secchia River (Italy). The 750 km^2-domain, reported in Fig. 18a, includes a 43 km-long branch of the river (from Modena to Concordia), and the floodable area on its right bank, stretching to the Panaro River.

The BUQ grid used for the simulations was characterized by $\Delta x_{min} = 5$ m and $\Delta x_{max} = 40$ m. The maximum resolution was forced near the inflow and outflow boundary conditions, and around the breach location. A grid size $\Delta x = 10$ m was imposed along the remainder of the river. The resulting grid size distribution along the river is shown in Fig. 18b. Besides, the whole floodable area was discretized at the minimum resolution (hence its representation is avoided in Fig. 18b). The domain includes roughly $1{\cdot}10^6$ cells.

The 20 years-return period discharge hydrograph (with peak inflow discharge equal to 620 m^3/s after 24 h) was set as upstream boundary condition, while a rating curve was assigned downstream, far from the breach location. The breach opening was generated on the right levee when the peak discharge reached that location (indicated in Fig. 18a), 28 h after the beginning of the simulation. The event was then prolonged for 72 h, so that 100 h of physical time were simulated. Manning's roughness coefficient was assumed equal to 0.05 m$^{-1/3}$s, except for the residential and industrial areas, where it was increased to 0.143 m$^{-1/3}$s. Simulations were run using both first- and second-order accurate schemes, and results and execution times of the LTS and GTS models were compared.

Fig. 19a shows an example of the temporal resolution level distribution in the blocks near the breach location 5 h after the breach opening, while Fig. 19b represents the water depth map obtained from the second-order LTS simulation at the same moment. It can be noticed that the maximum temporal resolution ($m = 0$) is observed along the riverbed and at the breach location, where the fine grid size and the high water depth and velocity values dictate the time step for the whole domain. The leveed floodplain on the left is only partially inundated, with small water depths, thus a smaller temporal resolution level is observed here despite the fine grid size. The whole flooded area at the minimum spatial resolution is updated with the minimum temporal resolution ($M = 3$). Notice that a temporal resolution level is also assigned to a few blocks outside the wet area due to the BDO procedure, which activates dry blocks bounding wet blocks.

Fig. 20 describes the flooding evolution in time, by depicting the water surface elevation maps at three selected times, obtained from the second-order LTS simulation. A synthetic index for comparing flood extent of different scenarios is suggested by Horritt and Bates (2002):

$$I_F = \frac{n_{GTS} \cap n_{LTS}}{n_{GTS} \cup n_{LTS}}, \tag{18}$$

where n_{GTS} and n_{LTS} indicate the number of flooded cells at a fixed time in the GTS and LTS simulations, respectively. The index coincides with the unity when the flooded areas are the same. Table 5 reports I_F for first- and second-order simulations at selected times. All values are very close to one, in particular for the first-order scheme, confirming the close agreement between the inundated areas for GTS and LTS simulations. The RMSE of the water surface elevations in the flooded region obtained from GTS and LTS simulations is also reported in Table 5. For the first-order scheme, negligible differences can be noticed, while RMSE values up to a few centimeters are observed for the second-order scheme. After 72 h from the breach opening, approximately 140 km^2 are inundated. A comparison between the total flooded areas for all tests is reported in Table 6, showing almost equivalent flood extents for all simulations.

Table 6 also reports information about the computational performance of the models. The achieved time ratio is slightly larger for the first-order scheme (equal to 1.77) than for the second-order scheme (1.61); the same holds for the corresponding theoretical values, equal to 2.14 and to 1.74, respectively. During the second-order LTS simulation of this test case, the roll back procedure is activated only once, thus not affecting the computational time significantly. Ratios of physical to computational time R_T, reported in Table 6, confirm the efficiency of GPU-accelerated numerical models for practical applications, particularly if the LTS version of the scheme is adopted.

5. Conclusions

In this paper, a local time stepping strategy was implemented in an explicit FV numerical scheme, which solves the 2D SWEs on structured

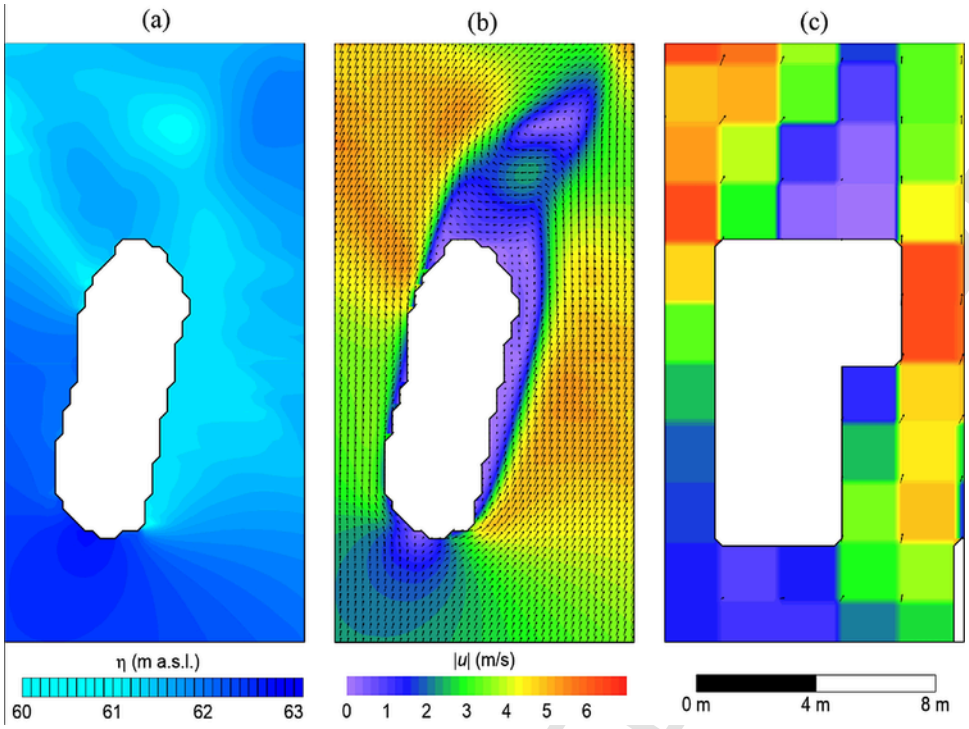

Figure 17. Details of the results for the Parma–Baganza test case at the bridge site: water surface elevation (a) and velocity (b) maps from the second-order LTS simulation; (c) velocity map from a second-order simulation where the bridge site is discretized at the minimum resolution ($\Delta x_{max} = 2$m).

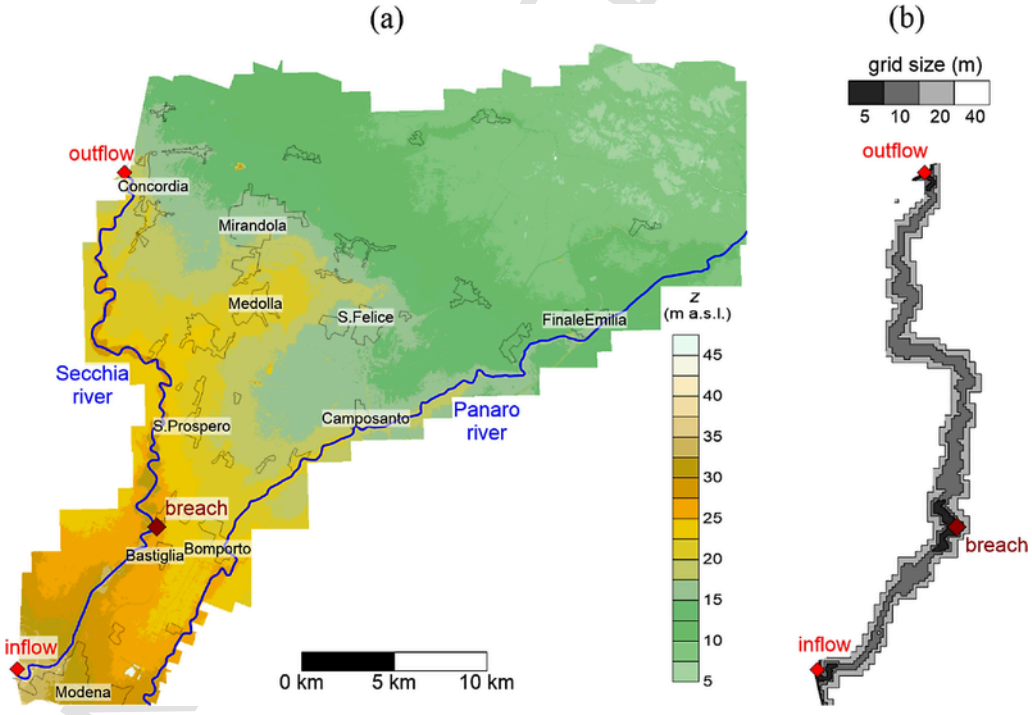

Figure 18. (a) Levee breach test case: bathymetry. The main residential areas are also indicated. (b) Grid size distribution along the river (the floodable area is discretized by elements of size equal to 40 m everywhere, and is not represented).

non-uniform grids. Both first- and second-order accurate schemes were considered. The method was developed in a CUDA/C + + code, so that the computational capability of GPUs could be fully exploited.

The model assessment was performed based on the simulation of three theoretical test cases and two field-scale problems. Results of the numerical tests show that, compared to the traditional global time stepping strategy, LTS reduces the total simulation times without impairing the solution accuracy. Execution time ratios between 1.2 and 2.8 were obtained. The achievable performance improvement mainly depends on the spatial resolution level distribution, and on the local flow field conditions. The adoption of the LTS method was observed to be especially convenient in the simulation of real field test cases characterized by a large domain discretized with a coarse mesh, where a high level of

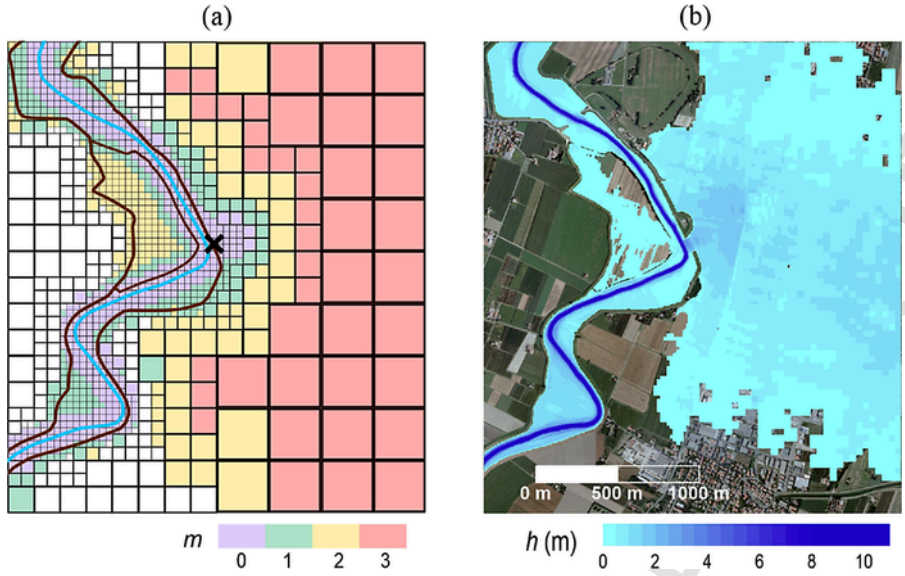

Figure 19. (a) Detail of the multiresolution grid, and temporal resolution levels, 5 h after the breach opening; the breach position (black cross), riverbed (blue line) and levees (brown lines) are also represented. (b) Water depth contour map near the breach position (5 h after the breach opening). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

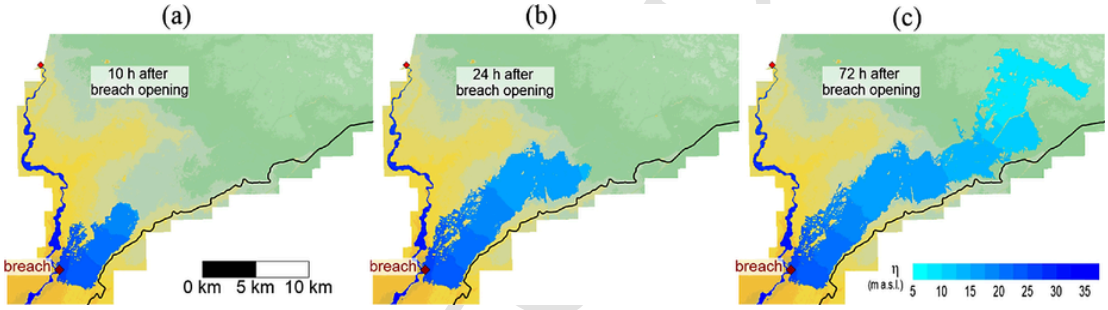

Figure 20. Levee breach test case: water surface elevation contour maps obtained from the second-order LTS simulation 10 h (a), 24 h (b), and 72 h (c) after the breach opening.

Table 5
Levee breach test case: I_F and RMSE between GTS and LTS water surface elevations in the flooded area at selected times t' after the breach opening, for first-order (FO) and second-order (SO) simulations.

t' (h)	I_F (-)		RMSE (m)	
	FO	SO	FO	SO
2	0.961	0.966	4.6E-03	1.6E-02
12	0.994	0.987	7.1E-03	2.6E-02
22	0.996	0.991	8.3E-03	2.6E-02
32	0.997	0.984	3.9E-03	2.7E-02
42	0.998	0.989	2.5E-03	2.8E-02
52	0.996	0.991	4.8E-03	2.9E-02
62	0.997	0.988	2.1E-03	3.4E-02
72	0.998	0.991	1.6E-03	4.4E-02

Table 6
Levee breach test case: total inundated area A_{flood}, ratio of physical to computational time R_T, and time ratio S_U for first-order (FO) and second-order (SO) simulations.

	FO		SO	
	GTS	LTS	GTS	LTS
A_{flood} (km^2)	140.74	141.03	141.25	140.08
R_T (-)	52.9	93.6	29.7	47.7
S_U (-)	1.77		1.61	

Algorithm 1
Pseudocode for the m-levels regularization procedure.

```
1: for level = 0 … M − 1
2:   for each block with m[block] = = level
3:     for neighbor = 1 … n_neighbors
4:       if m[neighbor] > level + 1
5:         m[neighbor] = level + 1
6:       end if
7:     end for
8:   end for
9: end for
```

Algorithm 2
Pseudocode for the buffer block definition procedure.

```
1: for block = 1 … n_active_blocks
2:   for neighbor = 1 … n_neighbors
3:     if m[block] – m[neighbor] > 0
4:       m_correct[block] = m[neighbor]
5:     end if
6:   end for
7: end for
```

refinement was required only in small selected regions (e.g. bridge piers, breach location).

The high values of ratios of physical to computational time achieved for the levee-breach scenario support the idea of applying GPU-accelerated models, coupled with an efficient LTS strategy, to

Algorithm 3
Comparison between GTS (a) and LTS (b) pseudocodes. The GPU parallel implementation is neglected here for the sake of clarity.

(a) GTS	(b) LTS
1: **WHILE** time ≤ time_end	1: **WHILE** time ≤ time_end
2: compute Δt and define wet blocks	2: compute Δt and define wet blocks
	3: assign first guess m-level to blocks
3: define active blocks (BDO procedure)	4: define active blocks (BDO procedure)
	5: m-levels regularization procedure & buffer region
	6: find M = max(m-level) of all active blocks
	7: **for** step = 0: $2^M - 1$
	8: **for** block = 1: n_active_blocks
	9: **if** step is an integer multiple of $2^{m[\text{block}]}$
4: update all active blocks (by Δt)	10: update block (by $2^m \cdot \Delta t$)
	11: **end if**
	12: **end for**
	13: **end for**
5: time = time + Δt	14: time = time + $2^M \cdot \Delta t$
6: **END WHILE**	15: **END WHILE**

Algorithm 4
Pseudocode for flux computation in the LTS integration procedure (first-order scheme). The GPU parallel implementation is neglected here for the sake of clarity.

```
1: for step s = 0 … 2^M − 1
2:   for block b = 1: n_active_blocks
3:     if s is an integer multiple of 2^m[b]
4:       for cell = 1: n_cells_per_block
5:         if (inner cell OR border cell with m[neigh] ≤ m[b])
6:           compute all fluxes with values at t + s·Δt
7:         else  // border cell with m[neigh] > m[b], as cell i in Figure 4
8:           if (s·2^−m[b] is even)
9:             compute all fluxes with values at t + s·Δt
10:          else
11:            compute inner flux with values at t + s·Δt
12:            compute border flux with values at t + (s − 2^m[b])·Δt
13:          end if
14:        end if
15:        update cell (by 2^m[b]·Δt) (Eq. 3)
16:      end for
17:    end if
18:  end for
19: end for
```

Algorithm 5
Pseudocode for second-order LTS integration procedure. The GPU parallel implementation is neglected here for the sake of clarity.

```
1: for step s = 0 … 2^M − 1
2:   for block = 1: n_active_blocks
       // sub-step A
3:     if 2·s is an integer multiple of 2^m [block] then do MUSCL extrapolation on block
       end if
4:     if s is an integer multiple of 2^m [block] then do flux comput. (Eq. (7)) + time
       integr. (Eq. (6)) on block end if
       // sub-step B
5:     if (2·s + 1) is an integer multiple of 2^m [block] then do MUSCL extrapolation on
       block end if
6:     if (s + 1) is an integer multiple of 2^m [block] then do flux comput. (Eq. (7)) on
       block end if
7:     if (s + 1) is an integer multiple of 2^m [block] then do half time step sum (Eq.
       (5)) on block end if
8:   end for
9: end for
```

quasi real-time 2D simulations of flooding events over domains of considerable extent, preserving the required level of detail where necessary. This prospect can become even more feasible if the rapid development of GPU capabilities and the possibility of extending the model to multi-GPUs are taken into account.

Acknowledgements

This work was partially supported by the Ministry of Education, Universities and Research (Italy) under the Scientific Independence of young Researchers programme, grant number RBSI14R1GP, CUP code D92I15000190001. The authors gratefully acknowledge the support of CINECA under Project PANCIA id. HP10CIGMEB, of NVIDIA under the CUDA Research Center Program, and of GNCS-INDAM. The authors wish to thank the anonymous Reviewers, whose valuable suggestions greatly contributed to improving the paper.

References

Alcrudo, F., Garcia-Navarro, P., 1993. A high-resolution Godunov-type scheme in finite volumes for the 2D shallow-water equations. Int. J. Numer. Methods Fluids 16 (6), 489–505.
Audusse, E., Bouchut, F., Bristeau, M.O., Klein, R., Perthame, B.T., 2004. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. SIAM J. Sci. Comput. 25 (6), 2050–2065.
Aureli, F., Maranzoni, A., Mignosa, P., Ziveri, C., 2008. A weighted surface-depth gradient method for the numerical integration of the 2D shallow water equations with topography. Adv. Water Resour. 31 (7), 962–974.
Barredo, J.I., 2009. Normalised flood losses in Europe: 1970–2006. Nat. Hazards Earth Syst. Sci. 9 (1), 97–104.
Bermudez, A., Vazquez, M.E., 1994. Upwind methods for hyperbolic conservation laws with source terms. Comput. Fluids 23 (8), 1049–1071.
Bradford, S.F., Sanders, B.F., 2002. Finite-volume model for shallow-water flooding of arbitrary topography. J. Hydraul. Eng. 128 (3), 289–298.
Brodtkorb, A.R., Sætra, M.L., Altinakar, M., 2012. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. Comput. Fluids 55, 1–12.
Brufau, P., García-Navarro, P., Vázquez-Cendón, M.E., 2004. Zero mass error using unsteady wetting–drying conditions in shallow flows over dry irregular topography. Int. J. Numer. Methods Fluids 45 (10), 1047–1082.
Caleffi, V., Valiani, A., Zanni, A., 2003. Finite volume method for simulating extreme flood events in natural channels. J. Hydraul. Res. 41 (2), 167–177.
Castro, M.J., Ortega, S., De la Asuncion, M., Mantas, J.M., Gallardo, J.M., 2011. GPU computing for shallow water flow simulation based on finite volume schemes. Comptes Rendus Mécanique 339 (2-3), 165–184.
Costabile, P., Costanzo, C., Macchione, F., 2012. Comparative analysis of overland flow models using finite volume schemes. J. Hydroinf. 14 (1), 122–135.
Costabile, P., Costanzo, C., Macchione, F., 2015. Performances and limitations of the diffusive approximation of the 2-d shallow water equations for flood simulation in urban and rural areas. Appl. Numer. Math. 116 (1), 141–156.
Crossley, A.J., Wright, N.G., 2005. Time accurate local time stepping for the unsteady shallow water equations. Int.J. Numer. Methods Fluids 48 (7), 775–799.
Dazzi, S., Maranzoni, A., Mignosa, P., 2016. Local time stepping applied to mixed flow modelling. J. Hydraul. Res. 54 (2), 145–157.
de la Asunción, M., Castro, M.J., Fernández-Nieto, E.D., Mantas, J.M., Acosta, S.O., González-Vida, J.M., 2013. Efficient GPU implementation of a two waves TVD-WAF method for the two-dimensional one layer shallow water system on structured meshes. Comput. Fluids 80, 441–452.
European Council (2007). Directive 2007/60/EC of the European Parliament and of the Council of 23 October 2007 on the assessment and management of flood risks.
Ferrari, A., Vacondio, R., Dazzi, S., Mignosa, P., 2017. A 1D–2D shallow water equations solver for discontinuous porosity field based on a generalized Riemann problem. Adv. Water Resour. 107, 233–249.
Garcia-Navarro, P., Vazquez-Cendon, M.E., 2000. On numerical treatment of the source terms in the shallow water equations. Comput. Fluids 29 (8), 951–979.
Greenberg, J.M., LeRoux, A.Y., 1996. A well-balanced scheme for the numerical processing of source terms in hyperbolic equations. SIAM J. Numer. Anal. 33 (1), 1–16.
Horritt, M.S., Bates, P.D., 2002. Evaluation of 1D and 2D numerical models for predicting river flood inundation. J. Hydrol. 268 (1), 87–99.
Kesserwani, G., Liang, Q., 2015. RKDG2 shallow-water solver on non-uniform grids with local time steps: application to 1D and 2D hydrodynamics. Appl. Math. Model. 39 (3), 1317–1340.
Kleb, W.L., Batina, J.T., Williams, M.H., 1992. Temporal adaptive Euler/Navier-Stokes algorithm involving unstructured dynamic meshes. AIAA J. 30 (8), 1980–1985.
Krámer, T., Józsa, J., 2007. Solution-adaptivity in modelling complex shallow flows. Comput. Fluids 36 (3), 562–577.
Kurganov, A., Petrova, G., 2007. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. Commun. Math. Sci. 5 (1), 133–160.
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. Adv. Eng. Softw. 78, 1–15.

Lastra, M., Mantas, J.M., Ureña, C., Castro, M.J., García-Rodríguez, J.A., 2009. Simulation of shallow-water systems using graphics processing units. Math. Comput. Simul. 80 (3), 598–618.
Liang, Q., 2011. A structured but non-uniform Cartesian grid-based model for the shallow water equations. Int. J. Numer. Methods Fluids 66 (5), 537–554.
Liang, Q., Borthwick, A.G., 2009. Adaptive quadtree simulation of shallow flows with wet–dry fronts over complex topography. Comput. Fluids 38 (2), 221–234.
Liang, Q., Marche, F., 2009. Numerical resolution of well-balanced shallow water equations with complex source terms. Adv. Water Resour. 32 (6), 873–884.
Liska, R., Wendroff, B., 1997. 2d shallow water equations by composite schemes. Int. J. Numer. Methods Fluids 30 (4), 461–479.
Osher, S., Sanders, R., 1983. Numerical approximations to nonlinear conservation laws with locally varying time and space grids. Math. Comput. 41 (164), 321–336.
Petaccia, G., Leporati, F., Torti, E., 2016. OpenMP and CUDA simulations of Sella Zerbino Dam break on unstructured grids. Comput. Geosci. 20 (5), 1123–1132.
Rogers, B.D., Borthwick, A.G., Taylor, P.H., 2003. Mathematical balancing of flux gradient and source terms prior to using Roe's approximate Riemann solver. J. Comput. Phys. 192 (2), 422–451.
Sætra, M.L., Brodtkorb, A.R., Lie, K.A., 2015. Efficient GPU-implementation of adaptive mesh refinement for the shallow-water equations. J. Sci. Comput. 63 (1), 23–48.
Sanders, B.F., 2008. Integration of a shallow water model with a local time step. J. Hydraul. Res. 46 (4), 466–475.
Sanders, B.F., Bradford, S.F., 2006. Impact of limiters on accuracy of high-resolution flow and transport models. J. Eng. Mech. 132 (1), 87–98.
Sanders, B.F., Schubert, J.E., Detwiler, R.L., 2010. ParBreZo: a parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale. Adv. Water Resour. 33 (12), 1456–1467.
Teng, J., Jakeman, A.J., Vaze, J., Croke, B.F.W., Dutta, D., Kim, S., 2017. Flood inundation modelling: a review of methods, recent advances and uncertainty analysis. Environ. Model. Softw. 90, 201–216.
Thacker, W.C., 1981. Some exact solutions to the nonlinear shallow-water wave equations. J. Fluid Mech. 107, 499–508.
Toro, E.F., 1999. Riemann Solvers and Numerical Methods For Fluid dynamics: a Practical Introduction. Springer.
Toro, E.F., 2001. Shock-capturing Methods For Free-Surface Shallow Flows. John Wiley.
Vacondio, R., Rogers, B.D., Stansby, P.K., 2012. Accurate particle splitting for smoothed particle hydrodynamics in shallow water with shock capturing. Int. J. Numer. Methods Fluids 69 (8), 1377–1410.
Vacondio, R., Rogers, B.D., Stansby, P.K., Mignosa, P., 2013. A correction for balancing discontinuous bed slopes in two-dimensional smoothed particle hydrodynamics shallow water modeling. Int. J. Numer. Methods Fluids 71 (7), 850–872.
Vacondio, R., Dal Palù, A., Mignosa, P., 2014. GPU-enhanced finite volume shallow water solver for fast flood simulations. Environ. Model. Softw. 57, 60–75.
Vacondio, R., Aureli, F., Ferrari, A., Mignosa, P., Dal Palù, A., 2016. Simulation of the January 2014 flood on the Secchia River using a fast and high-resolution 2D parallel shallow-water numerical scheme. Nat. Hazards 80 (1), 103–125.
Vacondio, R., Dal Palù, A., Ferrari, A., Mignosa, P., Aureli, F., Dazzi, S., 2017. A non-uniform efficient grid type for GPU-parallel shallow water equations models. Environ. Model. Softw. 88, 119–137.