2014

High Performance GPU Speed-Up Strategies For The Computation Of 2D Inundation Models

Asier Lacasta

Mario Morales-Hernández

Javier Murillo

Pilar García-Navarro

Reinaldo Garcia

HIGH PERFORMANCE GPU SPEED-UP STRATEGIES FOR THE COMPUTATION OF 2D INUNDATION MODELS

ASIER LACASTA (1), MARIO MORALES-HERNÁNDEZ(1), JAVIER MURILLO(1), PILAR GARCÍA-NAVARRO (1), REINALDO GARCIA (2)
(1): LIFTEC-CSIC, University Zaragoza, C/ Maria de Luna 3, Zaragoza, Spain
(2):Hydronia, LLC. USA

Two-dimensional (2D) models are increasingly used for inundation assessment in situations involving large domains of millions of computational elements and long-time scales of several months. Practical applications often involve a compromise between spatial accuracy and computational efficiency and to achieve the necessary spatial resolution, rather fine meshes become necessary requiring more data storage and very long computer times that may become comparable to the real simulated process. The use of conventional 2D non-parallelized models (CPU based) makes simulations impractical in real project applications and improving the performance of such complex models constitutes an important challenge not yet resolved. We present the newest developments of the RiverFLO-2D Plus model based on a fourth-generation finite volume numerical scheme on flexible triangular meshes that can run on highly efficient Graphical Processing Units (GPU's). In order to reduce the computational load, we have implemented two strategies: OpenMP parallelization and GPU techniques. Since dealing with transient inundation flows the number of wet elements changes during the simulation, a dynamic task assignment to the processors that ensures a balanced work load has been included in the Open MP implementation. Our strict method to control volume conservation (errors of Order 10^{-14} %) in the numerical modeling of the wetting/drying fronts involves a correction step that is not fully local, which requires special handling to avoid degrading the model. The efficiency of the model is demonstrated by means of results that show that the proposed method reduces the computational time by more than 30 times in comparison to equivalent CPU implementations. We present performance tests using the latest GPU hardware technology, that shows that the parallelization techniques implemented in RiverFLO-2D Plus can significantly reduce the Computational-Load/Hardware-Investment ratio by a factor of 200-300 allowing 2D model end-users to obtain the performance of a super computation infrastructure at a much lower cost.

INTRODUCTION

Physically based simulations of complex systems usually require large computational facilities to be completed in a reasonable time. Moreover when the simulated phenomenon is unsteady and based on a dynamical estimation of the updating time step, the computational performance

is an important topic to be taken into account. One of the most widespread strategies to reduce the computational cost is the use of parallel techniques, involving a suitable number of processors. Since CPU frequencies seem to be reaching their maximum capacity [1], nowadays Many-Core parallel techniques appear to be an interesting option. In recent years, Graphic Processing Unit (GPU) has been used to accelerate the calculations because of its inherent vector-oriented designing. In the present work, special attention is paid to the application of this GPUs to unsteady flows of interest in hydraulics.

The use of multiple CPU's was recently reported in [2],[3] or [4] and that of using GPU can be found in [5][6][7][8]. The GPU technology offers the performance of smaller clusters at a much lower cost [9].

When dealing with topographic representation some recent works [10] have shown the benefit of using unstructured meshes in unsteady hydraulic simulations over irregular topography. The quality of the numerical results is sensitive to the grid resolution. Hence grid refinement in general and adaptive grid refinement in particular are clearly an option. The latter is easy to implement on unstructured triangular meshes [11]. The present work shows the implementation in GPU of a code able to perform unsteady hydraulic simulations on variable density triangular unstructured meshes. This numerical engine is included in the last version of RiverFLO-2D Plus model [16].

FORMULATION

Governing equations

The water flow under shallow conditions can be formulated by means of the depth averaged set of equations expressing water volume conservation and water momentum conservation. That system of partial differential equations will be formulated here in a conservative form as follows:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \boldsymbol{E} = \boldsymbol{S} \tag{1}$$

Where $\boldsymbol{E}$=($\mathbf{F}$,$\mathbf{G}$) are the fluxes and

$$\mathbf{U} = \left(h,\, q_x,\, q_y\right)^T \tag{1}$$

is the vector of conserved variables with h representing the water depth, $q_x = hu$ and $q_y = hv$, and (u, v) the depth averaged components of the velocity vector $\mathbf{u}$ along the (x, y) coordinates respectively. The fluxes of these variables are given by:

$$\mathbf{F} = \left(q_x,\, \frac{q_x^2}{h} + \frac{1}{2} g h^2,\, \frac{q_x q_y}{h} \right)^T, \qquad \mathbf{G} = \left(q_y,\, \frac{q_x q_y}{h},\, \frac{q_y^2}{h} + \frac{1}{2} g h^2 \right)^T \tag{2}$$

where g is the acceleration of the gravity. The terms $0.5gh^2$ in the fluxes have been obtained after assuming a hydrostatic pressure distribution in every water column, as usually accepted in shallow water models. The bed slope and friction are source terms of the momentum equations:

$$\mathbf{S} = \left(0,\, gh(S_{ox} - S_{fx}),\, gh(S_{oy} - S_{fy})\right)^T \tag{3}$$

where the bed slopes of the bottom level z are

$$S_{ox} = -\frac{\partial z}{\partial x}, \qquad S_{oy} = -\frac{\partial z}{\partial y} \tag{4}$$

and the friction losses are written in terms of the Manning's roughness coefficient n :

$$S_{fx} = \frac{n^2 u\sqrt{u^2+v^2}}{h^{4/3}}, \qquad S_{fy} = \frac{n^2 v\sqrt{u^2+v^2}}{h^{4/3}} \tag{5}$$

Numerical method

In order to formulate the upwind cell-centered finite volume method, (1) is integrated for every cell *i*, Gauss theorem is applied and Riemann solvers are oriented perpendicularly to the edges of the grid cells. Assuming a piecewise representation of the conserved variables [12]

$$\frac{\partial}{\partial t}\int_{\Omega_i} \mathbf{U} d\Omega + \sum_{k=1}^{N_E} (\mathbf{E}\mathbf{n} - \mathbf{S})_k l_k = 0 \tag{9}$$

where N_E indicates the number of edges in cell i, $\mathbf{n} = (n_x, n_y)$ is the outward unit normal vector and l_k is the length of each wall edge k. The Jacobian matrix of the normal flux is evaluated

$$\mathbf{J}_{n,k} = \frac{\partial(\mathbf{E}\cdot\mathbf{n})}{\partial \mathbf{U}} = \frac{\partial(\mathbf{F}\,\mathrm{n_x})}{\partial \mathbf{U}} + \frac{\partial(\mathbf{G}\,\mathrm{n_y})}{\partial \mathbf{U}} \tag{10}$$

and can be diagonalized as $\mathbf{P}_k^{-1}\mathbf{J}_{\mathbf{n},k}\mathbf{P}_k = \mathbf{\Lambda}_k$ where $\mathbf{P}_k = (\tilde{\mathbf{e}}^1, \tilde{\mathbf{e}}^2, \tilde{\mathbf{e}}^3)$ is built using the eigenvectors of the Jacobian and $\mathbf{\Lambda}_k$ is a diagonal matrix with eigenvalues $\tilde{\lambda}_k^m$ in the main diagonal. Applying Roe's linearization, it is possible to express the difference in vector $\mathbf{U}$ as well as the source term vector $\mathbf{S}$ across the grid edges, projected onto the matrix eigenvectors basis $\tilde{e}_k^m$:

$$\delta\mathbf{U}_k = \mathbf{P}_k \mathbf{A}_k \qquad \mathbf{S}_k = \mathbf{P}_k \mathbf{B}_k \tag{11}$$

where $\mathbf{A}_k = (\alpha^1, \alpha^2, \alpha^3)_k^T$ contains the set of wave strengths and $\mathbf{B}_k = (\beta^1, \beta^2, \beta^3)_k^T$ contains the source strengths. More details about the formulation of the numerical scheme as well as the entropy fix are given in [12].The 2D numerical scheme is formulated according to the upwind approach for the updating of a single cell, dealing with the contributions that arrive to the cell:

$$\Delta\mathbf{U}_i^n = -\frac{\Delta t}{\Omega_i}\sum_{j=1}^{N_E}\sum_{m=1}^{3}\left(\tilde{\gamma}_m^- \tilde{\mathbf{e}}_m l\right)_k^n \tag{12}$$

In this expression, $\tilde{\gamma}_k^- = \frac{1}{2}\left[1 - \mathrm{sign}\left(\tilde{\lambda}_k\right)\right]\left(\tilde{\lambda}\tilde{\alpha} - \tilde{\beta}\right)_k$. The time step is dynamically chosen to fulfill the CFL condition:

$$\Delta t = \mathrm{CFL}\,\frac{\min(\chi_i, \chi_j)}{\max_m |\tilde{\lambda}^m|} \qquad \mathrm{CFL} \le 1 \tag{13}$$

where i, j are the indexes of the cells sharing the edge k and $\chi_i = \frac{A_i}{\max_{k=1,N_E} l_k}$. A correct formulation of the source terms as proposed in [12] avoids the appearance of negative values of water depth. However, with the solute volume as a new conserved quantity, nonphysical solutions for the solute concentration may appear [13].This problem is fixed in [14], avoiding unbounded solute concentrations. This scheme has been proved to be robust, conservative, well-balanced and positivity preserving even in presence of wet/dry fronts over irregular bed [12].

GPU ACCELERATION

The acceleration of the calculation has been developed by using Shared Memory programming Model and Many-Core Model by means of OpenMP for Intel Processors and CUDA for the NVIDIA GPU programming. Some works such [15] explain how OpenMP can be applied to

achieve a 4x or 8x speed-up factor depending on the processor used in the test while other works such [15] deal with the implementation of the shallow water equations using the same method in distributed memory machines. The last option is very useful when very big domains are required and then, an enormous number of cells (>100M Cells) is used in the simulation. In these cases the main drawback is the amount of memory required to perform the simulation and then, the distribution of the complete domain in different memory systems appears to be a solution.

The GPU contains a large number of processors working all together applying the same operation over different elements. In order to program using this paradigm, NVIDIA has developed CUDA (Compute Unified Device Architecture) that abstracts some aspects of the hardware, allowing programmers to develop general purpose programs efficiently. There are two main points to understand the performance of GPUs by means of CUDA. The first is based on the way CUDA applications are developed. The basic element to be processed is called Thread. Threads are identified by labels ranging between 0 and `BlockDim`. The group of Threads is called Block, and it contains a (recommended) 32 multiple number of Threads. Finally any group of Blocks is called Grid. The second aspect of interest is the hardware architecture. The minimum unit is the Streaming Processor (SP), where a single Thread is executed. A group of SP's form the Streaming Multiprocessor (SM), typically with 32 SP's. Finally, a GPU is generally composed by between 2 and 16 SM's. The GPU distributes the Blocks among the SMs. The SMs in turn assigns the Threads to the SP's. All SP's inside the multiprocessor perform the same operations at the same time, but each of them applies it to a different element inside a vector. The designing of the GPU is the reason of the recommendation of configure `blockDim` multiple of 32. The set of 32 threads processed in a SM is called `warp`.

Implementation

GPU programming requires being careful with the next four aspects:

- *Number of elements to be processed*: It is required that the number of blocks and threads per block is greater than or equal to the number of elements to be processed.
- *Bottlenecks*: In order to process all the operations following the GPU paradigm, special attention must be paid to the shared information between the processing elements.
- *Floating Point data precision*: The GPU arithmetic performance is halved when using double precision data. Many applications require double precision because of numerical aspects but there exist many others for which simple precision is enough to develop the calculations. When single precision is acceptable, performance can be almost doubled on GPU
- *Data transfer reduction*: The communication between CPU and GPU is very slow. In general, all the operations must take place inside the GPU, otherwise the overhead caused by data transfers may generate such a cost that the global performance of the implementation can be lower than on CPU.

In this case, double precision is required to ensure the most accurate results.

APPLICATION TO A REAL CASE

Description

In order to analyze the accuracy of the model results as well as the performance of the implementation, a real event in the Ebro River is used. The event occurred in January 2013

when the river discharge increased around 4 times. This produced the inundation of an urban region near the river that was captured using aerial photographs. The photographs provide a qualitative extension of the flooded area, as displayed in Fig.1 (left). The zone where the flooding had more impact was urban area at the inner part of the Meander of Ranillas. It is worth stressing that infiltration and exfiltration effects are not included in the model hence cannot be captured with this model and will not be analyzed. Moreover, the length of the hydrograph is 8 days. The measurement was made in Zaragoza by the Ebro Water Authority (Figure 1 right).

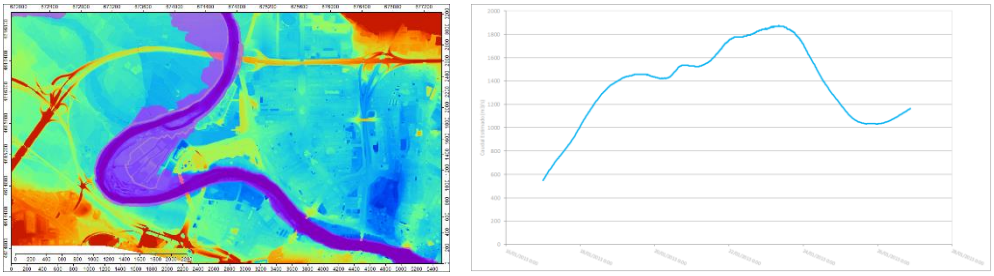

Figure 1. Left: Flooding extent for the 25/01/2013 as captured by the Ebro Water Authority (Spain). Right: Measured discharge at the nearest gauge station.

This case requires precise definition of the levees as well as those areas where buildings block the water flow. This justifies the necessity of refined meshes that capture those features allowing the method to have enough information to model the actual behavior of the flow. In order to demonstrate the necessity of this kind of meshes, the domain will be discretized by means of two grids GA and GB (Figure 2).

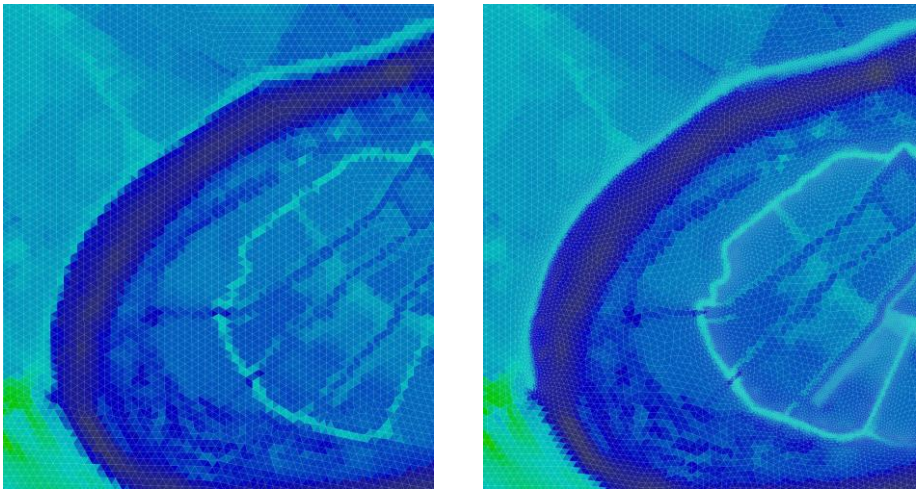

Figure 2. Detail of the meshes used in the simulation: GA is a coarse and not refined mesh (left) and mesh GB which is locally refined (right)

Mesh GA is made of triangles of uniform size and contains 134886 elements. The size of the triangles is such that the main river bed is well represented. However, the details of the nearby topography are poorly represented. Mesh GB with 201208 cells offers the possibility of a local refinement along the levees and in the zone where the inundation was produced.

The cases have been run with several configurations: two for the CPU version using 1 core and 4 cores and two using GPUs NVIDIA GTX780 and NVIDIA Tesla c2075.

Results

Results may be analyzed from two points of view. The first and most important is the quality of the simulation and therefore its deterministic character. Figure 3 shows the flooding extension for both meshes GA (left) and GB (right) when the highest point in the hydrograph was reached. In the case of mesh GA, where all the mesh has the same cell size density, the estimation of the flooding is excessive. Moreover, the levees cannot be represented with precision and there are zones where the water jumps over the wall providing the advance of the water path (Figure 3 left). On the other hand, mesh GB includes a detailed representation of the levees shape at all the relevant locations that helps in the prediction of the actual flood extension (Figure 3 right). For a quantitative comparison, Figure 4 shows the measured water depth at a river gauging station versus the computed value at the same location using mesh GB.

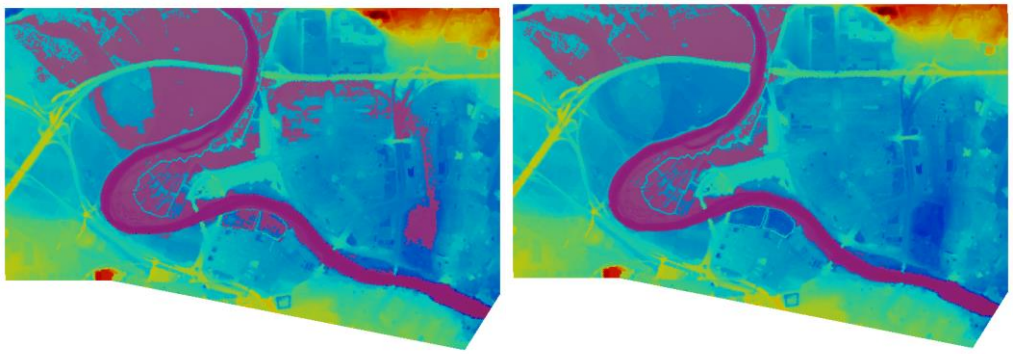

Figure 3. Flooding area for t=190 h for mesh GA (left) and mesh GB (right).

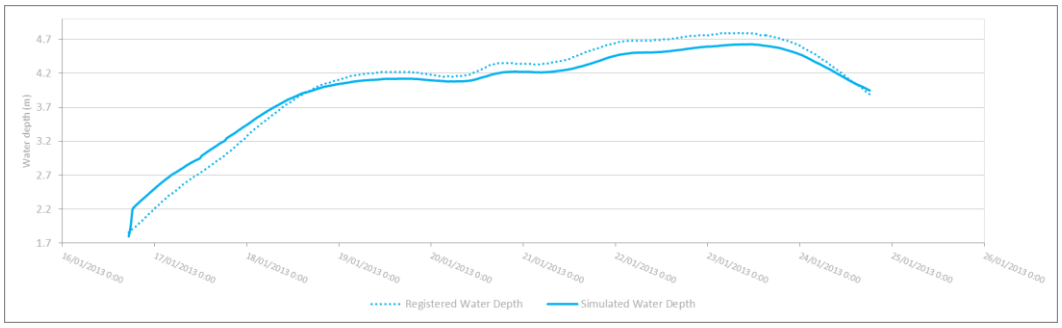

Figure 4. Comparison of water depth at a gauge located within the domain and the depth as computed with mesh GB.

The second feature to analyze is the computational efficiency. Table 1 and Figure 5 display the speed-up achieved by the CPU (sequential and parallel) and GPU computations on both meshes for this event. The speed-up is defined with reference to the sequential CPU computation. The local refinement is useful to improve the accuracy of the flooding extension but it increases the number of wet cells in the calculation. This has a double impact on the computational time as it also influences the dynamical time step choice. That is the reason why the CPU time is larger for mesh GB than for mesh GA. Nevertheless, our results show that the GPU implementation provides the best speed-up when using mesh GB that, at the same time, offers more accurate

predictions. If the case involved more wet cells, the speed-up figures would increase reaching a value up to 65.

			CPU (Intel Core i7-3820 @ 3.60 GHz)			GPU			
			1 Core	4 Core		NVIDIA GTX 780		NVIDIA TESLA c2075	
	# Cells	Wet Cells	Time (h:m:s)	Time (h:m:s)	**Speed-Up**	Time (h:m:s)	**Speed-Up**	Time (hh:mm:ss)	**Speed-Up**
Mesh GA	134886	33601	13:17:57	3:39:06	**3.64**	0:49:29	**16.12**	0:49:48	**16.02**
Mesh GB	201208	48464	60:50:38	15:38:28	**3.89**	2:22:35	**25.60**	2:33:11	**23.75**

Table 1. Computational cost and performance of different optimizations for the test case.

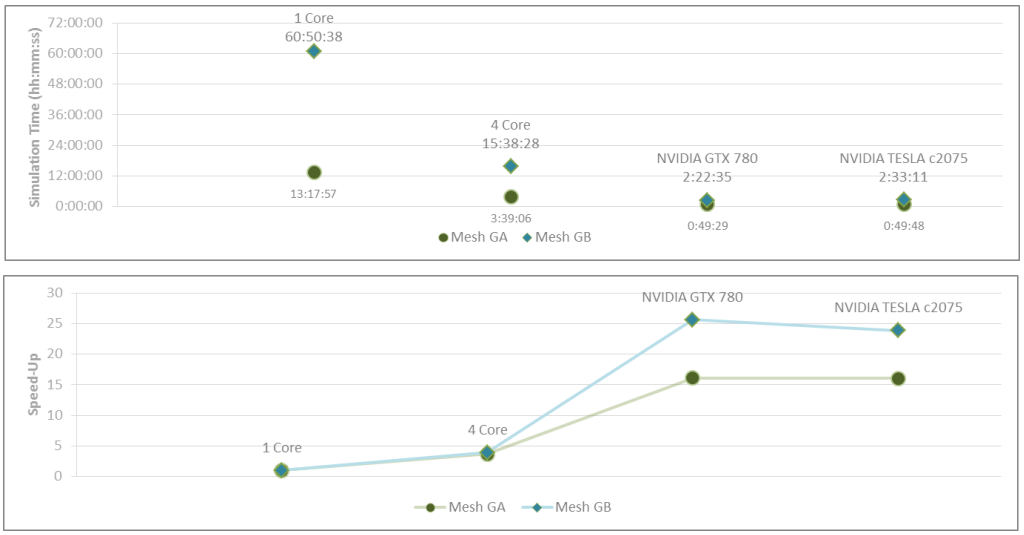

Figure 5. Computational cost and performance of different optimizations for the test case.

CONCLUSIONS

Practical hydraulic applications require a compromise between spatial accuracy and computational efficiency in order to achieve both the necessary spatial resolution and cover long events. Rather fine grids are necessary in many cases reducing the allowable time step size for explicit calculations. When, at the same time, a reasonable computational time is desired, the use of GPU codes such as the one implemented in RiverFLO-2D Plus, is one of the options for computing large space and temporal domain problems.

ACKNOWLEDGMENTS

The authors acknowledge the Ebro Water Authority for the useful data provided

REFERENCES

[1] Danowitz, A., Kelley, K., Mao, J., Stevenson, J.P., Horowitz, M..Cpu db: Recording microprocessor history. *Queue* 2012

[2] Kalyanapu, A.J., Shankar, S., Pardyjak, E.R., Judi, D.R., Burian, S.J. Assessment of GPU computational enhancement to a 2d flood model. *Environmental Modelling & Software* 2011;26(8):1009 – 1016.

[3] Brodtkorb, A.R., Sætra, M.L., Altinakar, M.. Efficient shallow water simulations on gpus: Implementation, visualization, verification, and validation. *Computers & Fluids* 012;55(0):1–12

[4] Rostrup, S., Sterck, H.D.. Parallel hyperbolic pde simulation on clusters: Cell versus gpu. *Computer Physics Communications* 2010;181(12):2164

[5] Castro, M.J., Ortega, S., de la Asunción, M., Mantas, J.M., Gallardo, J.M.. Gpu computing for shallow water flow simulation based on finite volume schemes. *Comptes Rendus Mécanique* 2011;339(2–3):165 – 184.

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

[7] Brett F. Sanders, Jochen E. Schubert, Russell L. Detwiler, ParBreZo: A parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale, *Advances in Water Resources*, Volume 33, Issue 12, December 2010, Pages 1456-1467

[8] Castro, M., García-Rodríguez, J., González-Vida, J., Parés, C.. Aparallel 2d finite volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows. *Computer Methods in Applied Mechanics and Engineering* 2006;195(19–22):2788 – 2815.

[9] Jacobsen, D., Thibault, J.C., Senocak, I. An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters. *American Institute of Aeronautics and Astronautics (AIAA) 48th Aerospace Science Meeting Proceedings*. 2010

[10] Caviedes-Voullième, D., García-Navarro, P., Murillo, J.. Influence of mesh structure on 2d full shallow water equations and scs curve number simulation of rainfall/runoff events. *Journal of Hydrology* 2012;448–449(0):39

[11] Schubert, J.E., Sanders, B.F., Smith, M.J., Wright, N.G.. Unstructured mesh generation and landcover-based resistance for hydrodynamic modeling of urban flooding. *Advances in Water Resources* 2008;31(12):1603 – 1621.

[12] Murillo, J. and García-Navarro, P., 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. *Journal of Computational Physics*, 229, 4327-4368.

[13] Murillo J., Burguete J., Brufau P. and García-Navarro P., 2005. Coupling between shallow water and solute flow equations: analysis and management of source terms in 2D, *Int. J. Numer. Methods Fluids*, 49, 267-299.

[14] Murillo, J. and García-Navarro, P., 2011. Improved Riemann solvers for complex transport in two-dimensional unsteady shallow flow. *Journal of Computational Physics*, 230, 7202-7239.

[15] A. Lacasta, M. Gonzalez-Sanchis, J. Murillo, P. Garcia-Navarro, A compromise between computational load and refinement criteria in two-dimensional hydraulic simulation: a real case, *Hydroinformatics 2012*, Hamburg

[16] Hydronia, 2014. RiverFLO-2D Plus User's Guide. www.hydronia.com , Pembroke Pines, FL. USA.