Universidad Zaragoza
1542

Modeling Hydrodynamics for Water Resources
University of Zaragoza, 17-20 June 2019

High Performance Computing in hydraulics: the new era of flood forecasting

M. Morales-Hernández[a,*], M. B. Sharif [b], S. Gangrade [c,d], T.T. Dullo [b], S.-C. Kao [c], A. Kalyanapu [b], S.K. Ghafoor [b], K.J. Evans [a]
[a]Computational Science and Engineering Division
Oak Ridge National Laboratory, Oak Ridge, TN (USA)
[b] Tennessee Technological University
Cookeville, TN (USA)
[c]Environmental Sciences Division
Oak Ridge National Laboratory, Oak Ridge, TN (USA)
[d]The Bredesen Center, University of Tennessee, Knoxville, TN (USA)
*Corresponding author: moraleshernm@ornl.gov

This work was supported by the U.S. Air Force Numerical Weather Modeling Program. The research used resources of the Oak Ridge Leadership Computing Facility. Some of the co-authors are employees of Oak Ridge National Laboratory, managed by UT Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

1 Introduction

Computational hydraulics is experiencing a breakthrough in the recent decade. The conventional single central processing unit (CPU) environment is no longer sufficient nowadays for computing, having made way to CPU clusters, graphic processing units (GPUs) or clusters of GPUs. The cost of access to these cutting-edge technologies, unaffordable in the past, has been made possible due to the increasing popularity of GPUs and also to the possibility of renting low-cost cloud services such as those offered by Amazon or Google.

Flood forecasting and real-time predictions were associated with simplified models. Although their accuracy was sometimes in question, they were the only models able to provide an approximation for operational purposes. Muskingum-Cunge models, diffusive-wave models or even the complete 1D shallow water model are examples of that in the context of hydrology and hydraulics. However, some of the assumptions underlying these models are not valid when trying to reproduce major flood events, with a marked bi-dimensional character. On the other hand, 2D shallow water models provide more accurate solutions of complex flows over floodplains. However, their computational cost was extremely high, confining them for small temporal and spatial scale applications.

With the aim of accelerating the computations leaving aside the advances related to computer technologies, many enhancements have been done in the recent decade to increase the computational efficiency without sacrificing the accuracy of the results. In that sense, it is worth mentioning the development of 1D-2D coupled models [7], Large Time Step (LTS) schemes [8] and the correct estimation of source terms [9]. 1D-2D coupled models are able to achieve a double gain: first, the river channel is discretized using the 1D model hence many 2D cells are removed from the domain. But second, most importantly, these cells usually governed the time step

size considering that they are usually the smallest cells with largest velocities. On the other hand, LTS schemes are able to relax the stability condition when using explicit schemes, allowing bigger time step sizes and consequently reducing the computational time. Finally, source terms frequently dominate over fluxes when dealing with flooding events and a wrong estimation at each computational edge may lead to non-physical solutions or dramatic time step reductions. Therefore, a careful estimation of the source strength can be done from the analysis of the Riemann Problem at each edge.

Computational times are extraordinarily reduced with the aid of HPC and GPUs. In fact, parallel programming has become the essential way of accelerating the computations. As an example, both OPENMP and MPI strategies (shared and distributed memory parallelization) have been widely used in the past for CPU computations. However, GPU computing has emerged in the last few years as one of the most promising and affordable way of acceleration due to its massively parallel architecture. Although structured meshes are more convenient for this paradigm, some reordering algorithms have been successfully applied to improve the speed-up when dealing with unstructured meshes [6]. On top of that, this technology can be also combined with OPENMP and MPI (the so-called multi-GPU) to achieve even faster computations, allowing large temporal and spatial scales.

In this work, a physically-based model based on the resolution of the 2D shallow water equations [5] is explored. It is implemented in CUDA and able to run on multiple GPUs. In particular, different implementations are analyzed: the convenience of using float/double precision, the computation on wet cells versus all the domain and the way of writing the information (ASCII or binary). The results are compared in terms of accuracy and performance.

2 Equations and numerical method

In this work, the focus is put on the resolution of 2D shallow water equations. They can be written in conservative differential form as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S_b} + \mathbf{S_f} \tag{1}$$

$$\begin{gathered} \mathbf{U} = \begin{pmatrix} h \\ q_x \\ q_y \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} q_x \\ \frac{q_x^2}{h} + \frac{1}{2}gh^2 \\ \frac{q_x q_y}{h} \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} q_y \\ \frac{q_x q_y}{h} \\ \frac{q_y^2}{h} + \frac{1}{2}gh^2 \end{pmatrix} \\ \mathbf{S_b} = \begin{pmatrix} 0 \\ -gh\frac{\partial z}{\partial x} \\ -gh\frac{\partial z}{\partial y} \end{pmatrix} \quad \mathbf{S_f} = \begin{pmatrix} 0 \\ -\frac{gn^2}{h^{7/3}} q_x \sqrt{q_x^2 + q_y^2} \\ -\frac{gn^2}{h^{7/3}} q_y \sqrt{q_x^2 + q_y^2} \end{pmatrix} \end{gathered} \tag{2}$$

where $\mathbf{U}$ is the vector of conserved variables including the water depth, h, and the unit discharges in x and y directions, named q_x and q_y respectively. Moreover, $\mathbf{F}$ and $\mathbf{G}$ are the vector of fluxes and $\mathbf{S_b}$ and $\mathbf{S_f}$ contains the bed and roughness source terms respectively. In particular, roughness is modeled by means of Manning-Gauckler's law. In eq. (2), g accounts for the gravity acceleration, z is the elevation and n denotes the Manning's coefficient.

A finite volume upwind explicit scheme is implemented in this work, based on Roe's linearization. A squared mesh is used, denoting Δx as the grid spacing. The

derivation of the numerical scheme follows [9, 7] for the fluxes and bed slope source terms, including the correct estimation of bed slope source terms at each edge. Nevertheless, a different discretization for the roughness terms is proposed here, following [11], in which an implicit formulation is chosen. Therefore, a two-step algorithm is proposed for the update of a cell i from time t^n to time t^{n+1}:

$$\mathbf{U}_i^\star = \mathbf{U}_i^n - \underbrace{\frac{\Delta t}{\Delta x}\sum_{k=1}^{4}\sum_{m=1}^{3}\left[(\tilde{\lambda}\tilde{\alpha} - \tilde{\beta}_b)\tilde{\mathbf{e}}\right]_{m,k}^n}_{(\#)} \tag{3}$$

$$\mathbf{U}_i^{n+1} = \mathcal{F}(\mathbf{U}_i^n, \mathbf{U}_i^\star) \tag{4}$$

where $\tilde{\alpha}$ and $\tilde{\beta}_b$ are the fluxes and slope source term linearizations and $\tilde{\lambda}$ and $\tilde{\mathbf{e}}$ are the eigenvalues and eigenvectors of the system of equations respectively. On the other hand, $\mathcal{F}$ is defined as follows:

$$\begin{aligned}
\mathcal{F}^1 &= h^\star \\
\mathcal{F}^2 &= -(q_x{}^\star)\left(\frac{1-\sqrt{1+4S_f}}{2S_f}\right) \\
\mathcal{F}^3 &= -(q_y{}^\star)\left(\frac{1-\sqrt{1+4S_f}}{2S_f}\right)
\end{aligned} \tag{5}$$

where

$$S_f = \frac{\Delta t g n^2 \sqrt{(q_x{}^\star)^2 + (q_y{}^\star)^2}}{(h^n)^{7/3}} \tag{6}$$

Time step is restricted by the CFL condition:

$$\Delta t = \mathrm{CFL}\frac{\Delta x}{\max_i\left\{\left|\frac{q_x}{h}\right| + \sqrt{gh}, \left|\frac{q_y}{h}\right| + \sqrt{gh}\right\}} \qquad \mathrm{CFL} \leq 0.5 \tag{7}$$

More details about the numerical scheme can be found in [9, 7, 11].

3 High Performance Computing

The term High Performance Computing refers to the use of supercomputers and parallel processing to solve advanced problems. Although it involves the use of CPUs and cluster of CPUs to perform these large and complex computational problems, the focus in this work is put on GPU computing and the use of multiple graphic cards to carry out the simulations.

3.1 GPU implementation

As in CPU, GPU computing has its own languages and platforms to implement the code. Although CUDA (the architecture created by Nvidia) has gained acceptance in the last years due to its analogy with C and C++ and also as a result of the growth of Nvidia market, other directives and platforms such as OpenACC, OpenMP (with GPU offload from version 4.5) and Kokkos are increasing their popularity. Although it is very unlikely to outperform CUDA when dealing when Nvidia graphic cards, they offer less programming effort and portability to other machines and compilers, which are extraordinarily valuable characteristics.

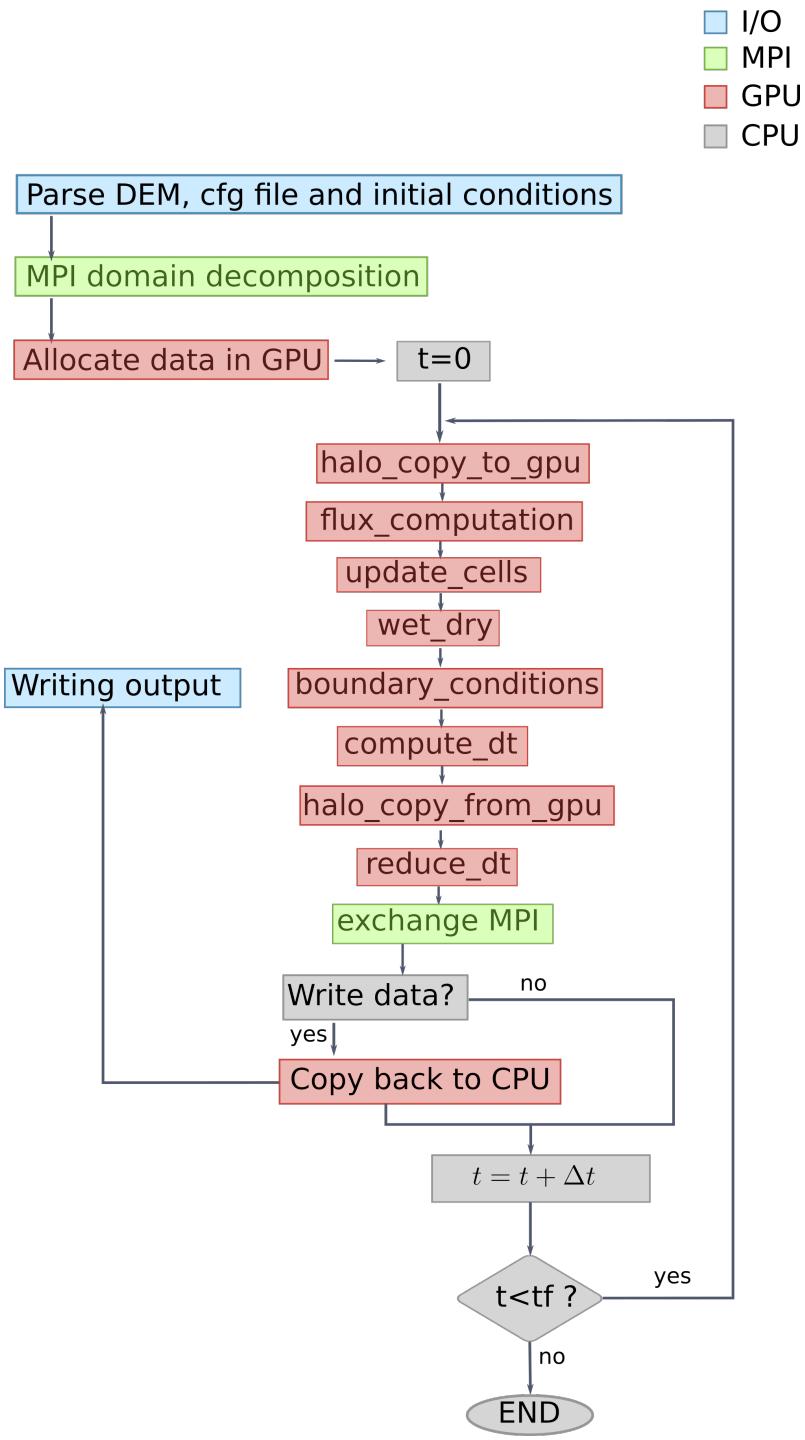

Figure 1: Flowchart of the GPU implementation

In this work, the focus is on CUDA. A flowchart of the current implementation of the flooding code is displayed in Figure 1 where the CPU, GPU, I/O and MPI processes are colored.

After the preprocessing and the MPI decomposition, it is necessary to allocate the memory and copy the information to the GPU (*Allocate data in GPU*). Once this is done, the temporal loop is almost computed in GPU except from the output writing. Going into details of the GPU functions, *flux_computation* refers to the computation of (#) in eq. (3). After (#) has been computed, *update_cells* evolves the conserved variables from time t^n to time t^{n+1} according to (4) while the subroutine *wet_dry* remove normal velocities when facing up wet/dry interfaces with discontinuous elevations. After boundary conditions are imposed (*update_boundaries*), the time step size

in the sub-domain is computed for the next iteration (*compute_dt*). Afterwards, the minimum of all sub-domains (*reduce_dt*) will be imposed as the value that guarantees the stability condition. It is worth mentioning the subroutines *halo_copy_to_gpu* and *halo_copy_from_gpu* (beginning and end of the loop respectively) that copy the value of the sub-domain boundaries to an auxiliary variable that communicates with the neighbor sub-domains. This fact is explained in detail in the following paragraph.

3.2 MPI domain decomposition: multi GPU

Message Passing Interface (MPI) is a message-passing library interface specification that allows a portable and scalable communication for large-scale parallel applications. Its main feature is that it does not need shared memory. Consequently, it becomes crucial in the programming of distributed systems. When dealing with large domains, domain decomposition is the key for parallelization. The main goal is to subdivide the computational domain into sub-domains of equal or variable size trying to guarantee the following requirements:

- Each sub-domain contains the same amount of computational effort.
- The communication between the sub-domain and its neighbor sub-domain is minimal.

In this work, a 1D row-wise domain decomposition is applied. Although for the 2D framework there exists other ways of partitioning (2D blocks involves less communication with asynchronous data transfer), the row-wise 1D partitioning is chosen because of its simplicity. The procedure is detailed in Figure 2 for the decomposition of a 16x10 partitioned in 4 sub-domains.

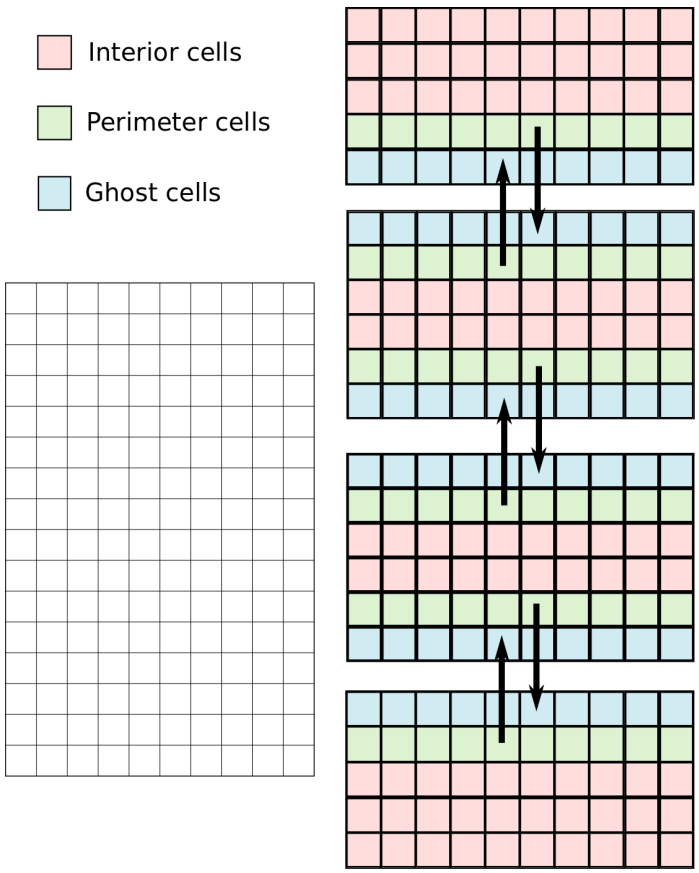

Figure 2: Row-wise MPI partitioning. A 16x10 domain is divided into 4 sub-domains

As observed, each sub-domain will be composed by interior, perimeter and ghost cells. Interior cells do not need information from other sub-domain and can be computed independently inside their sub-domain. On the contrary, perimeter cells require some data from the neighboring sub-domains at the same time as they influence also the result for other perimeter cells in the neighboring sub-domains. Therefore, their values must be sent to the 'ghost cells' as indicated by the arrows. This process is referred to *exchange MPI* in Figure 1 and it is closely connected with the *halo_copy_to_gpu* and *halo_copy_from_gpu*. As a summary, each sub-domain has to send the information to their neighbors sub-domains and to receive the information from them. As the MPI calls to exchange the information are performed in the CPU, these perimeter values have to be copied for each sub-domain to auxiliary variables in the CPU. Once the exchange has been done, these values have to be copied back to the GPU for the next iteration.

4 Other features

Apart from a good GPU and MPI implementations, some other considerations have to be taken into account when trying to design an optimized tool able to run as fast and as accurate as possible. Here, the focus is put on single/double precision, wet/all cells and input/output files.

4.1 Single vs. double precision

The precision for the real numbers involved in the computations has been already discussed in [4]. Almost every compiler has the option to define either a single or a double precision computation hence the choice is completely open to the programmer. The difference between both approaches is the number of bites that are used to represent the real number: single precision floating point arithmetic deals with 32 bit floating point numbers whereas double precision deals with 64 bit.

The trend in CFD applications and, in particular, in the computational hydraulics framework, is to implement the code using a double precision floating point. The main advantage for this choice is the accuracy of the solution since machine epsilon (relative error due to rounding) is 10^{-15}. In fact, double precision might be necessary if tiny relative differences are significant or if large variations in depth or velocities are expected. Moreover, on modern CPUs, the arithmetic performance difference with single precision is almost negligible. On the other hand, single precision is able to save memory since exactly the half of the information is used to store the value of each real number. However, the machine epsilon is 10^{-6}, which can compromise the accuracy of the numerical scheme or the convergence in an iterative method.

When dealing with GPU computations and specially with multi GPU computing, additional aspects have to be considered. First, GPUs have been historically developed and optimized to perform single precision computations faster than double precision. Depending on the graphic card, the difference between single and double computation can be up to eight times faster. Besides, GPU memory can be a limiting factor for large scale problems. In this case, switching to single-precision would make possible the storage of a mesh of a double size. And last but not least, data transfers and communications not only between nodes in multi GPU computing but also between device and host could represent a bottleneck in the case of double precision variables. For these reasons, among others, single/double precision dichotomy is analyzed in this work from the point of view of accuracy and performance.

4.2 Wet/all cells computation

The majority of flooding problems (not including rainfall/runoff components) only involves a low percentage of wet cells. It is very well-known that excluding the dry cells from the computations could remarkably and positively impact the computational performance. An easy solution could be to introduce an "if statement" at the beginning of the flux computation, in order to include just the pair of cells which at least one of them is wet. It is not a complete exclusion of dry cells since the loop is still done over all cells, but the improvement should be noticeable. However, when dealing with multiple GPUs, an uneven distribution of work across the nodes can result in a load imbalance and impact the scalability [10].

On the contrary, it is possible to absolutely remove the dry cells from the computation just building a set of wet cells from the beginning of the computation and enlarging its size as long as the simulation runs. Altough the implementation is not an ardous task in CPU, its extension to GPU is not straightforward [1]. Even though, the potential enhancement of eliminating these cells from the computation can be balanced out with the cost of building the set of wet cells each iteration and to sort them.

In this work, just the effect of introducing an "if statement" is evaluated, and the results are compared with the solution including all cells.

4.3 I/O data

Large temporal and spatial scales imply considerable amount of data to read and write. The read is done in the preprocessing therefore it usually represents a low amount of time compared to the simulation time. Conversely, the write of data is carried out during the simulation each output time. For instance, when working with a flood model that is in charge of prediction and forecasting, a typical output time goes from 15 minutes to one hour. Therefore, writing millions of data each half an hour sometimes turns out to be a bottleneck. Binary files has been proven to save computational time rather than ASCII files. In this work, we analyze the impact of writing the information in a binary format in contrast to the conventional ASCII format. It is important to remark the difference in the implementation when using multi-GPU computing: while the binary format gathers the information for each MPI task rejoining it in a single domain and writing it sequentially, in the ASCII output each sub-domain is in charge of writing its own information, without joining them in a single file.

5 Results and discussion

5.1 Description

Three test cases are considered in this work to evaluate the accuracy and the performance of the scheme. The first test case (C1) is a classical dambreak problem over dry initial condition in a frictionless domain with flat topography. The numerical solution is computed with a mesh resolution of $\Delta x = 1$. The details of the analytical solution can be found in [2]. The second test case (C2) consists in a domain whose topography is a paraboloid surface and the initial condition is a planar surface of water with initial velocity. The details as well as the analytical periodic solution are described in [2]. After one period, the solution should recover the initial condition. However, three periods are considered in this work, and the numerical solution is computed with a mesh of $\Delta x = 0.01m$. The last test case (C3) is the massive flood produced by Hurricane Harvey in the summer 2017 along the Gulf Cost of United States, which is by far the heaviest large-scale rainfall event in the US history. It

is considered as a challenge test case. The domain is about 7000 km^2 and the ten day event (including six days of model spin-up, followed by the heaviest four days Hurricane Harvey rainfall) representing the heaviest rainfall event in the US history has been simulated with a coarse (30m $\times$ 30m) and fine (10m $\times$ 10m) square grid. Output time is set to 1800s. More information about the details can be found in [3].

The simulations are carried out in the Oak Ridge National Laboratory's 200 petaflop supercomputer called Summit. To the date (June 2019) is the fastest supercomputer of the world, made of 4,608 nodes, each one containing 6 GPUs Tesla V100. Four different implementations are considered in this work. They are summarized in Table 1.

Implementation	Single/Double	Wet/All	I/O
I1	Double	Wet	Binary
I2	Single	Wet	Binary
I3	Double	All	Binary
I4	Double	Wet	ASCII

Table 1: Implementations

5.2 Accuracy and mass conservation

The accuracy of the model is analyzed as follows: for test cases C1 and C2, the results achieved by the single and double precision implementation are compared between them and also against the analytical solution. Obviously, either writing the information in binary or ASCII, or carrying out the computation in all the domain or only in wet cells has no effect on the accuracy of the results.

Regarding C1, Figure 3 includes the computational profile of water depth (left) and velocity magnitude (right) at time t=6s. The solution with respect to the exact solution is acceptable for both float and double precision approaches and their solutions are almost identical, having negligible differences between them.

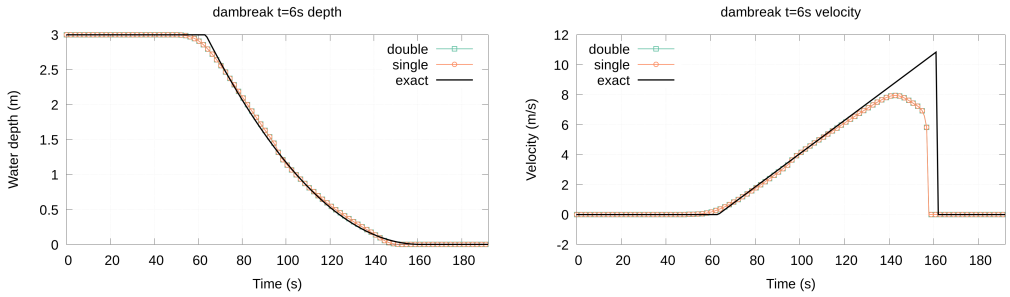

Figure 3: Exact and numerical solutions of the water depth (left) and velocity magnitude (right) for test case C1

Results for C2 are included in Figures 4 and 5. Figure 4 includes the plot of y=2 (centerline) of the exact solution and the single and double precision implementations at t=3T for the water surface elevation. On the other hand, Figure 5 contains the 2D plot (at t=3T as well) of the error with respect to the analytical solution of the single (left) and double (right) precision approaches. Although there are some difference patterns on the left part of the solution, which should be dry on the analytical solution, they are almost insignificant since the water depth values are in the order of machine epsilon for the float implementation.

The mass conservation is shown for test cases C1 and C2 in Figure 6. The mass percentage understood as the relative mass error with respect to the mass at the beginning of the simulation is displayed in the y-axis while the x-axis contains the

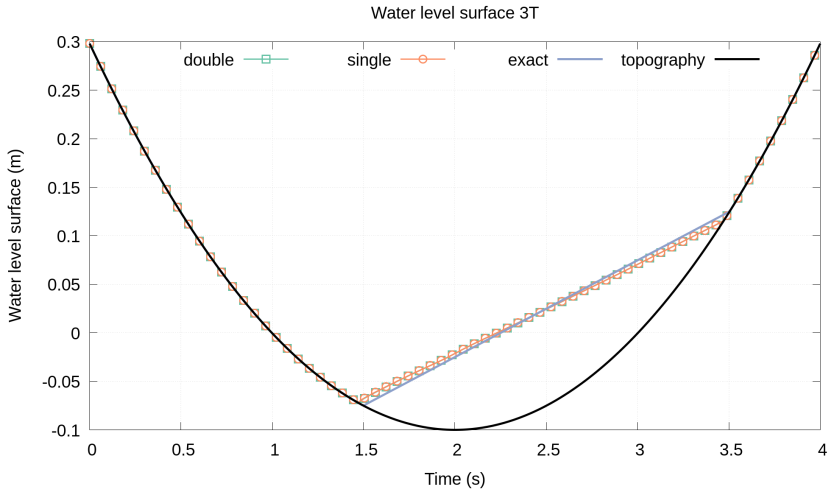

Figure 4: Exact and numerical solutions for the centerline (y=2) at t=3T for the water surface elevation, computed with single and double precision for test case C2

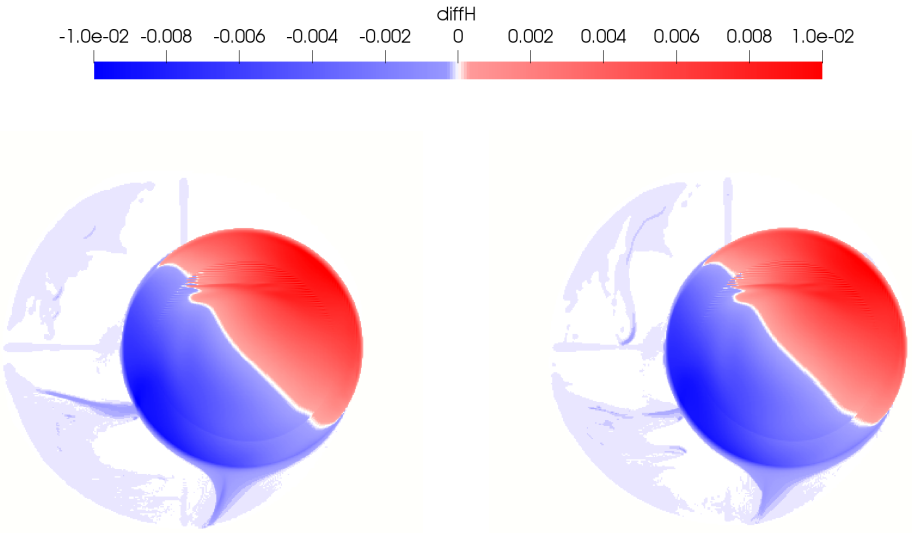

Figure 5: 2D plot of the error with respect to the analytical solution made by the single (left) and double (right) implementations

normalized time for both simulations, where 0 is the beginning of the simulation and 1 is the end. As expected, each implementation achieves an error in the order of magnitude of its machine epsilon value.

5.3 Performance

The performance of the model with the four implementations (I1-I4) shown in Table 1 is analyzed in this section. Each implementation is used to run both cases C3-30m and C3-10m with different number of GPUs: 1, 2, 4, 8 and 16 respectively. First of all, the information in terms of computational time is displayed in Figure 7 (in logscale) for test case C3-30m (left) and test case C3-10m (right).

As observed, single precision (I2) implementation is the fastest approach, being around 20%-40% faster than the I1 implementation. Moreover, as long as the number of GPUs increases, the computational time decreases in most of the cases. However there is some point for the C3-30m test case from which the performance collapses,

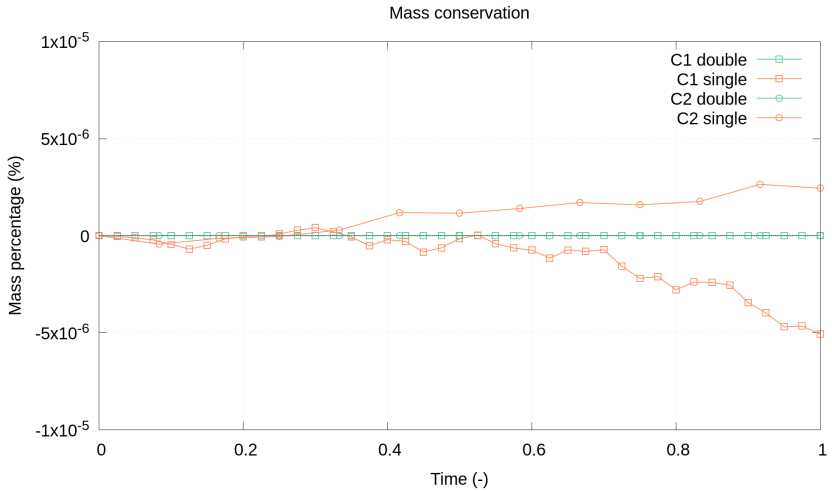

Figure 6: Mass conservation for test cases C1 and C2 and float and double implementations

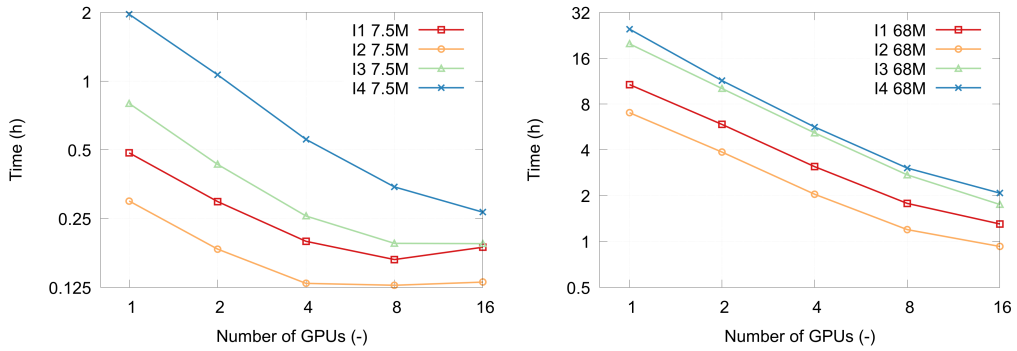

Figure 7: Time consumed by each implementation (I1-I4) for test case C3-30m (left) and test case C3-10m (right)

even getting worse results than with less resources. This fact can be also figured out for the C3-10m test case although a higher number of GPUs should be used (more than 16). Apart from this, I3 implementation, which includes the computation over all the cells is always more expensive than the I1 approach, which only includes the computation over wet cells. However, as long as the number of GPUs increases, the differences are diminished as well. Finally, the output writing is a key factor. The I4 implementation that writes the information in ASCII format takes between 40% and 300% of extra time to run the solution, in comparison with the I1 implementation.

The speed-up with respect to 1 GPU is plotted in Figure 8 (in logscale as well) for test case C3-30m (left) and test case C3-10m (right). The theoretical speed-up is also included in these graphs.

The main conclusion is that, given a certain implementation, the higher number of cells is, the better the scalability of the model is (in the sense that it is closer to the theoretical speed-up). Moreover, the inclusion of all cells for the computation (in contrast to only wet cells) improves the scalability. An explanation for this fact is given afterwards. Last, it is worth mentioning that the I4 implementation is overperforming the theoretical speed-up for 2, 4 and 8 GPUs in the C3-10m test case. The reason for this is simple: as long as we increase the number of GPUs, the sub-domain sizes decreases. Since each MPI task is in charge of writing just the information for its own sub-domain, the performance is increased with respect to 1 GPU.

According to Figure 1, different processes can be distinguished in a single run.

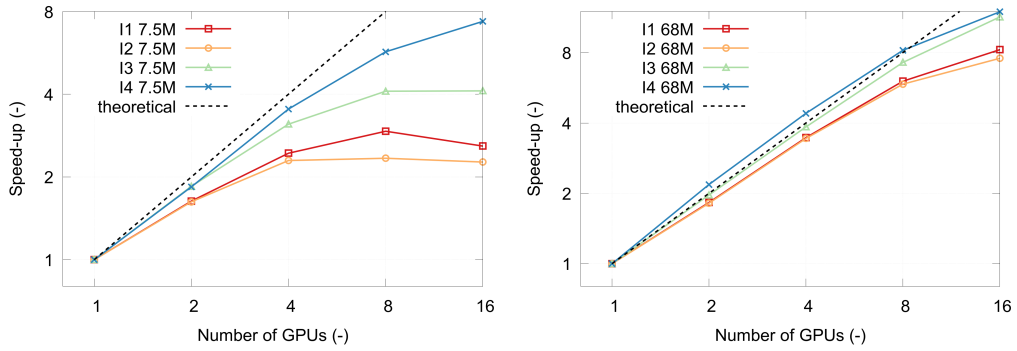

Figure 8: Speed-up with respect to 1 GPU achieved by each implementation (I1-I4) for test case C3-30m (left) and test case C3-10m (right)

Therefore, CPU, I/O, MPI and GPU times can be computed separately in order to analyze their weight in the computation burden. The plot of all of them is condensed in Figure 9 for the C1-30m test case.

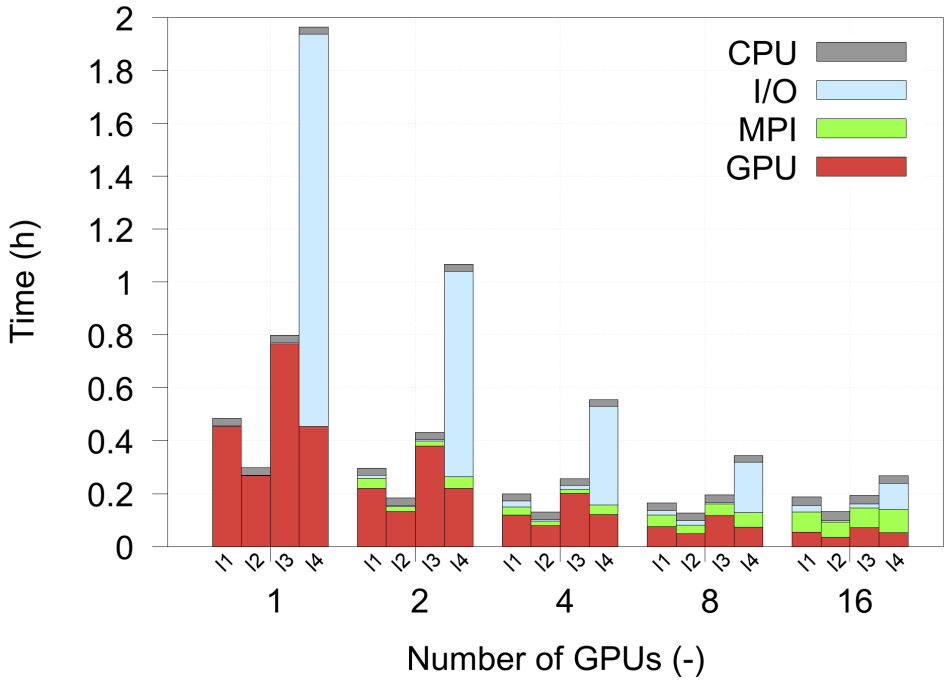

Figure 9: CPU, I/O, MPI and GPU time consumed by each implementation (I1-I4) for test case C3-30m

Several conclusions can be extracted from this plot:

- I/O time in ASCII format (I4) represents a bottleneck, spending even more time on it than for the computation itself. As long as the number of GPUs increases, the I/O time decrease since each MPI task handles the writing of each sub-domain.
- I/O time in binary for a double precision computation (I1, I3) is higher than for a single precision computation (I2) although the difference is not remarkable.
- GPU computation is consistent: first, there is no difference between I1 and I4 implementations. Then, I2 is faster than I1 since the graphic card is optimized for single precision. Furthermore, I3 is slower than I1 since the computations

is done in all the domain instead of only on wet cells. Finally, the higher the number of GPUs is, the lower the GPU time is.

- CPU time is almost constant for all the simulations and implementations.
- As expected, as long as the number of GPUs increases, the number of communications and the amount of data to be transferred among sub-domains also increases, even dominating over the GPU time (for instance in the case of 16 GPUs). Therefore, MPI time turns out to be a bottleneck for big number of GPUs, even more when using the row-wise domain decomposition.
- The MPI time for I2 implementation is lower than for the I1 implementation. The reason is purely a question of amount of information (number of bytes) to be exchanged between the sub-domains.
- The MPI time for I3 implementation is lower than for the I1 approach. The explanation for this is due to the MPI load imbalance that is behind the I1 implementation. It is next detailed.

As a part of the analysis, the time consumed by each MPI task for each process (CPU, I/O, MPI and GPU) is also computed. The results are only shown for the 16 GPU simulations for both C3-30m (Figure 10) and C3-10m (Figure 11) test cases. On the left side, the I1 implementation is displayed while the I3 approach is plotted on the right side.

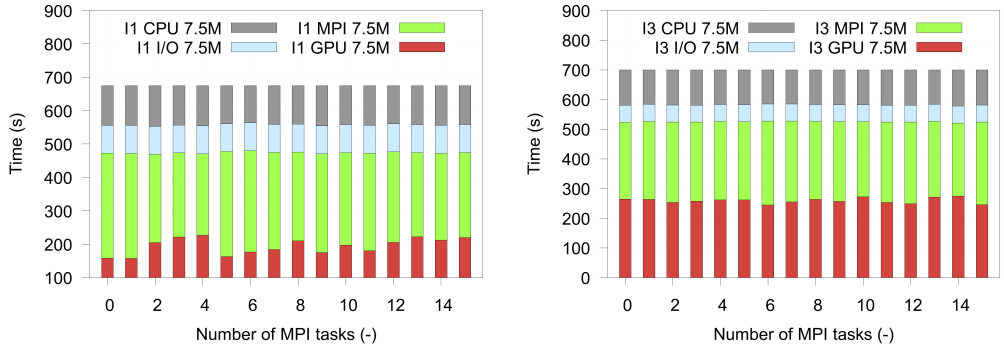

Figure 10: MPI load for test case C3-30m with 16 GPUs for implementations I1 (left) and I3 (right)

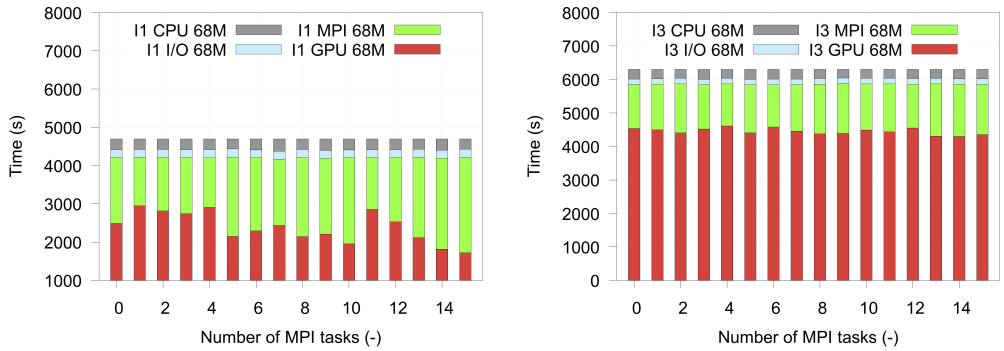

Figure 11: MPI load for test case C3-10m with 16 GPUs for implementations I1 (left) and I3 (right)

As can be seen, the CPU and I/O times are almost equivalent between MPI tasks. However an imbalance is detected for the GPU computation in the I1 implementation. Consequently, this derives to an imbalance in the MPI time too, since the

sub-domains have to exchange information between them and they cannot do it until the neighbor sub-domain has finished its computation. As stated in section 3.2, the process of sending/receiving the information to/from the neighboring sub-domains acts as a synchronization, mimicking the *MPI_Barrier* statement. On the other hand, the I3 implementation, computed over all cells, makes a better balance between MPI tasks, resulting in a improvement in the scalability.

6 Conclusions

Different implementations for the same hydraulic model are analyzed in this work. Looking at the results, a single precision approach could be a good choice. Actually most scientific computations generally do not need to be that accurate, because measurement errors and the degree of uncertainty in parameters largely overwhelms the errors introduced by the floating point rounding. Despite the potential errors, the results in comparison with the double precision in terms of accuracy are identical nevertheless the computation is faster since GPUs are usually optimized for single precision floating point operations. Moreover, when using multiple GPUs, the amount of information to be exchanged with a single precision approach is exactly half of what is exchanged with a double precision implementation, increasing the performance. On the other hand, writing the information in binary format results crucial to avoid the bottleneck produced by the ASCII format when dealing with large number of cells. Furthermore, the inclusion of all cells in the computation in contrast to computing only on wet cells improves the scalability for multi-GPU although the computation is slower.

The main conclusion for this study is that simulating ten days of one of the most extreme event in the US history in a spatial domain around 7000 km^2 can be done in less than 10 minutes for a 30m resolution grid and in less than one hour for a 10m resolution grid. This fact opens the door to real-time 2D shallow water based modeling for flood prediction and forecasting.

References

[1] Brodtkorb, A.R. and Saetra, M.L. and Altinakar, M., Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation, Computers & Fluids (55) 1-12, 2012

[2] Delestre, O. and Lucas, C. and Ksinant, P. and Darboux, F. and Laguerre, C. and Vo, T. and James, F. and Cordier, S. SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies. Int. J. Numer. Meth. Fluids (72): 269-300, 2013

[3] Dullo, T. T. and Gangrade, S. and Kalyanapu, A. J. and Kao, S.-C. and Ghafoor, S. K. and Evans, K. J., High-resolution Modeling of Hurricane Harvey Flooding for Harris County, TX using a Calibrated GPU-Accelerated 2D Flood Model, American Geophysical Union 2018 Fall Meeting, Dec. 10-14, Washington, D.C. 2018

[4] Itu, L.M. and Suciu, C. and Moldoveanu, F. and Postelnicu. Comparison of Single and Double Floating Point Precision Performance for Tesla architecture GPUs. Bulletin of the Transilvania University of Brasov Series I: Engineering Sciences Vol. 4 (53) No. 2, 2011

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

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

[7] Morales-Hernández, M. and Lacasta, A. and Murillo, J. and Brufau, P. and García-Navarro, P. A Riemann Coupled Edge (RCE) 1D-2D finite volume inundation and solute transport model, Environmental Earth Sciences(74:11), 7319-7335, 2015.

[8] Morales-Hernández, M. and Hubbard, M.E. and García-Navarro, P. A 2D extension of a Large Time Step explicit scheme (CFL $>$ 1) for unsteady problems with wet/dry boundaries, Journal of Computational Physics(263) 303-327, 2014.

[9] Murillo, J. and García-Navarro, P. Weak solutions for partial differential equations with source terms: Application to the shallow water equations, Journal of Computational Physics(229:11) 4327-4368, 2010.

[10] Tallent, N. R. and Adhianto, L. and Mellor-Crummey, J.M. Scalable Identification of Load Imbalance in Parallel Executions Using Call Path Profiles, Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, 2010, pp. 1-11., 2010.

[11] Xia, X. and Liang, Q. A new efficient implicit scheme for discretising the stiff friction terms in the shallow water equations, Advances in Water Resources (117) 87-97, 2018.