A parallelized algorithm to speed up 1D free-surface flow simulations in irrigation canals

Lucas Bessone, Joan Soler-Guitart and Pablo Gamazo

ABSTRACT

A parallel algorithm for 1D free-surface flow simulations in irrigation canals is shown. The model is based on the Hartree method applied to Saint-Venant equations. Due to the close-to-steady flow nature in irrigation canals, external and internal boundary conditions are linearized to preserve the parallel character. Gate trajectories, off-take withdrawals, and external boundary conditions are modeled as piece-wise functions of time, so there are discontinuities. To achieve a fully parallelized algorithm, an explicit version of the Hartree method is chosen, and external and internal boundary conditions are linearized around operation point. This approach is used to build a computer simulator, written in C-CUDA language. Two tests by ASCE Committee on Canal Automation Algorithms have been used to evaluate accuracy and performance of the algorithm. The Maricopa Stanfield benchmark has been used to prove its accuracy, and the Corning Canal benchmark to evaluate performance in terms of processing time. Surprisingly, solving a 12 hr-long prediction horizon with a cell size of about $\Delta x = 10$ m is less than 1 s on a Nvidia K40 card. Results were compared with a serial and a multi-CPU version of the same algorithm. The implementation that showed the best performance on different platforms is the one that uses GPU.

Key words | 1D Saint-Venant equations, CFL condition CUDA, GPGPU, Hartree method, HPC

Lucas Bessone
Pablo Gamazo
Water Department,
Centro Universitario Regional Litoral Norte (CENUR LN), Universidad de la República,
Av. Gral. Rivera 1350, 50000 Salto,
Uruguay

Joan Soler-Guitart (corresponding author)
CR. Sant Lluís 43, 08790 Gelida,
Spain
E-mail: *norcat50@gmail.com*

HIGHLIGHTS

- A high-performance parallel algorithm for solving 1D-Saint Venant Equations in irrigation canals has been implemented. It is highly efficient enabling many simulations in a reasonable period of time. In this way, Genetic Algorithms can be used in the context of inverse problems in free-surface flow phenomena.
- The GPU parallel algorithm used in the paper is very efficient making it possible to obtain steady flow solutions by using transient equations as an alternative to the step method.
- The philosophy inherent in the methodology could also be applied to other phenomena, such as the flow in pipes. The resolution time of the transients is so short that the steady state could be found from the final state of a transient in a very large networks in real time.

INTRODUCTION

The development of numerical models that represents real systems usually involves a calibration phase, which implies the determination of unknown parameters of the phenomenon governing equations from the observed data. One of the most used methodologies are genetic algorithms which can involve thousands of simulations (Whitley 1994).

doi: 10.2166/hydro.2020.049

Therefore, having efficient simulation algorithms is very important. Herein, the phenomenon studied is the free-surface flow in open canals which is described by means of the 1D-Saint-Venant's equations (*1D-SVE*). In the context of irrigation canal control – more specifically when model predictive control strategy is adopted – simulations are needed to have good predictions on responses to control actions (Martín-Sanchez & Rodellar 1996). Efficient simulators could also be useful in this field.

The use of graphical processing units (*GPUs*) for scientific calculation has increased in the last decade since scientists and engineers have been looking for alternatives to central processing units (*CPUs*) for high-performance computing (Anders & Kandrot 2010). Nevertheless, most *GPU* applications have been developed in the field of 2D and 3D computational fluid dynamics, and the number of 1D applications in civil engineering on hydraulics/hydrology is significantly lower. In this paper, we implement a parallel algorithm for solving *1D-SVE* in irrigation canals that uses *GPUs*. Several works in the literature report that explicit methods have a greater potential than implicit to take advantage of the parallelization in the *GPUs* (Aissa *et al.* 2017; Bondarenco *et al.* 2017; Bessone *et al.* 2018). Thus, a numerical explicit scheme has been chosen for the algorithm.

According to Wylie (1969), all numerical methods, whether they are explicit, implicit, or characteristics, give similar results when compared to the observed data, depending more on the accuracy of the starting data than on the particular methodology. In this paper, a Hartree method with linear space interpolations has been implemented, that is, a first order version of the method of characteristics (*MOC*). The *MOC* was chosen since it is well well known for having many merits regarding the aspects of theoretical and physical interpretation for *1D-SVE* modeling (Yang & Chiu 1993). Details of the Hartree method can be found in Katapodes (2016) and Litrico & Fromion (2009). Gómez (1988) considers the *MOC* one of the best methods to solve unsteady flow problems in prismatic canals. Soler *et al.* (2015) used the Hartree method in the context of inverse problems to reproduce the flow in a laboratory canal, and were able to accurately reproduce observed water depth hydrographs in a transient.

The application of the numerical techniques requires to discretize the domain x/t (space/time) and find the solution at certain points only. The canal is divided into cross sections and two variables are considered at each section: the water depth y_i and the flow velocity v_i. As the spatial discretization is fixed, the solution is always obtained at the same points of the canal. The solution for each instant depends on the solution in the previous instant. The specified-time-interval scheme has been used commonly in applying the *MOC* to unsteady open-canal flow problems. However, with the use of this scheme, the numerical error for the simulation results can always be induced due to the interpolation used to approximate the characteristics trajectory. In order to overcome the lack of accuracy, Yang & Chiu (1993) proposed to use *MOC* with higher-order splines for interpolation, and for demonstration example, cell sizes from $\Delta x = 200$ m to $\Delta x = 800$ m were used. Alternatively, it is possible to use the four-point implicit scheme, also known as the box scheme finite difference or Preissmann's method. This scheme is known to be a conservative method. Although the mesh size in the box scheme can be larger than in the *MOC*, the two methods give good results, both in terms of surface profiles in a time instant and as in terms of flow hydrograph at a section (Ostad-Ali-Askari *et al.* 2018). Another strategy for increasing accuracy is fine discretization, which has been adopted herein.

The fine linear discretization strategy has been adopted since the increasing number of computational nodes is compensated by its simplicity of computing. A fine discretization forces the reduction of the time step size according to the Courant–Friedrichs–Lewy (*CFL*) condition, so that the efficiency in terms of computation time is reduced. However, the concept of efficiency is not as simple as it first seems, since explicit schemes can make the most of parallel architecture. When considering explicit schemes, the solution of each node depends exclusively on the solution in the surrounding nodes. Thus, each node can be updated independently of the rest, therefore, it is not necessary to look for values in the memory far from the point which is being updated. Under these conditions, each point can be resolved in a different process unit at the same time, either *CPU* or *GPU*. To adopt an explicit scheme is amenable to multi-threading.

The box scheme finite difference along with external and internal boundary conditions results in a system of linear equations which must be solved for each time step. For a single canal, the coefficient matrix has a band width of five and can be solved by one of many banded matrix

solvers. Direct solvers for banded linear systems cannot be performed in parallel without communication. This can be accomplished through a shared memory bus and synchronicity, things that may cause delays. Moreover, for network canal problems, sparse terms destroy the banded structure and the consequent loss in the locality of data deteriorates the performance in many-core platforms. On the other hand, the memory use of implicit solvers strongly limits the size of problem that can be solved.

Currently, the lines of research on new numerical schemes for *1D-SVE* are aimed at improving the techniques based on finite volume. Specifically, new ways of solving the Riemann solver are investigated, such as in Hodges & Liu (2019). These techniques are very suitable for the complex flows that occur in rivers, such as in the presence of hydraulic jumps and very irregular geometries. These techniques do not contribute much in the case of the flow in irrigation canals, where it is not usual to find hydraulic jumps or other types of discontinuity. It is well known that the finite volume technique is based on the principle of conservation of momentum, and therefore, it is adequate to be applied in local phenomena with discontinuities where energy losses are unknown, like it occurs just downstream a gate (Cozzolino *et al.* 2015). In other words, finite volume techniques give non-continuous solutions. Conversely, when hydraulic variables are continuous, as happens the most in irrigation canals, the energy principle seems more appropriate to be applied, since the resulting solutions are always continuous.

According to these points, a totally explicit scheme based on the Hartree method with linear interpolations has been chosen. The method is included in the MOC category, and its presentation is the objective of the following section. In the past, the method was rejected due to its lack of accuracy, but it was a time when computers had low power so that they had to use very sparse space discretizations ($\Delta x = 300$ m in Gómez (1988) for example). Using *CPU-GPU* acceleration makes it possible to use strong discretizations ($\Delta x = 1$ m, for example) in large-scale canals (a 28 km-long canal, for example), so that, lack of accuracy is overcome.

The remaining sections of this paper are as follows. In the next section, the parallel algorithm is presented, some arguments that justify each one of its steps are given, and the reasons why the Hartree method has been chosen for the resolution of the *1D-SVE*. Moreover, it describes in detail the used numerical scheme. It includes the discretization scheme used for the *1D-SVE* and the treatment of boundary conditions, both the external and the internal. In the section titled 'Illustrative examples', a pair of hydraulically complicated scenarios are presented for testing the algorithm. These scenarios are based on benchmarks that can be found in the canal control literature. The examples show the advantages of parallel algorithms as opposed to traditional serial algorithms. The algorithm performance in terms of processing time to complete the scenario simulation is compared between different platforms.

PARALLEL ALGORITHM

Although the *1D-SVE* can be applied in several research fields, such as river analysis systems or drainage system modeling, this paper is focused on the flow in irrigation canals. The flow in irrigation canals, which are often lined and prismatic-shaped, shows characteristics that differentiate it from river drainage system flow, and that influence the choice of the numerical scheme. Usually, water management in irrigation canals implies the use of electro-hydraulic devices that control the flow (such as the in-line gates) that remain in the same position for long periods of time. When a change of position occurs relatively quickly, it can be considered as an instantaneous movement, especially if the movement time is compared with the time the gates remain stopped. All that results in a close-to-steady state sub-critical regime where things occur very slowly. This behavior is common in irrigation canal control systems, and it is amenable to be described mathematically by means of piece-wise functions of time as the gate trajectories shown in Figure 1. As can be seen below, this definition is interesting for numerical parallelization purposes.

In the context of irrigation canal control, the flow control devices stay in the same position without moving most of the time. Only occasionally, the control algorithm changes the device positions according to a desired target (the period between movements is called sampling period, but is referred to as ΔT-long *Regulation Period* in this paper). As a result, unsteady flow varies weakly and slowly around the operation point at a close-to-steady canal state.

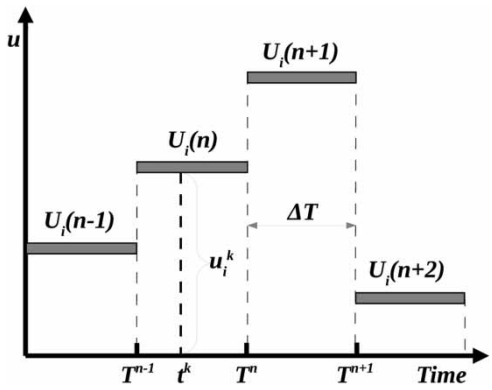

Figure 1 | Sketch of a piece-wise function of time. The piece $U_i(n)$ represents the control variable inside of the n^{th}-*Regulation Period*. That is the period of time between instants T^{n-1} and T^n. A position of the *i*-device at instant t^k is denoted in minor letters by u_i^k. Notice that there is a jump discontinuity at the end of each *Regulation Period*.

In that way, it is possible to consider the following assumptions:

- Flow control trajectories can be defined as ΔT-long piece-wise functions of time. Figure 1 plots a generic piece-wise function.
- An explicit scheme with fine discretizations for the Saint-Venant equations can be used in the numerical model.
- External and internal boundary conditions can be linearized in a small region around the operating point.
- The number of CFL stability condition verification, and the corresponding time step size computations, can be reduced a great deal because it is not necessary to do this at every time step.

Taking into account these points, it is possible to apply efficiently the three previous basic strategies for parallelization by solving flow equations for a *Regulation Period* entirely in the GPUs with no global memory accesses, as can be seen below.

At the end of each *Regulation Period*, external and internal boundary conditions change instantaneously. For example, actuators change opening gates. Therefore, between two consecutive *Regulation Periods*, there is a discontinuity called, herein, *Jump Step*. Non-linearized original equations at discontinuities have to be solved using iterative techniques, in which the number of floating point operations varies very much depending on the flow conditions. This is the reason why it has been decided to compute *Jump Step* in the *CPU*. Broadly speaking, the problem of simulation in open canals is divided into two parts, depending on where they are solved: on the *GPU* or on the *CPU*. An algorithm that meets all of these prescriptions is written as follows:

- Load initial data and initial conditions to the *CPU*: (Y_C^0, V_C^0)
- Compute the first time step size: $\Delta t^0 = CFL(Y_C^0, V_C^0)$

CPU external loop

1. Initialize *Regulation Period* Counter: $n = 1$
2. Compute *Jump Step*: $(Y_C^n, V_C^n) = Jump\ Step\ Model\ (\Delta t^{n-1}, Y_C^{n-1}, V_C^{n-1})$
3. Compute the time step size Δt^n:$\Delta t_C^n = CFL(Y_C^n, V_C^n)$
4. Copy from *CPU-memory* to *GPU-memory*: $\Delta t_G <= \Delta t_C^n$; $(Y_G^0, V_G^0) <= (Y_c^n, V_c^n)$

GPU inner loop

1. $k = 0$; $t = (n-1) \cdot \Delta T$; $t_{end} = n\ \Delta T$; $dstOut = true$
2. *if (dstOut = 'true')*: $(Y_G^1, V_G^1) = Regulation\ Period\ Model\ (\Delta t_G, Y_G^0, V_G^0)$

Otherwise: $(Y_G^0, V_G^0) = Regulation\ Period\ Model\ (\Delta t_G, Y_G^1, V_G^1)$

3. $k = k + 1$; $t = t + \Delta t_G$; $dstOut = !\ dstOut$;
4. *If* $t < t_{end}$ *go to GPU inner loop step 2*
5. *if (dstOut = 'false'):* $(Y_G^0, V_G^0) = (Y_G^1, V_G^1)$
6. Copy from *GPU-memory* to *CPU-memory*: $(Y_C^n, V_C^n) <= (Y_G^0, V_G^0)$
7. $n = n + 1$
8. If $n \leq \lambda$ go to *CPU* external loop step 2.

where: (Y_G^0, V_G^0) and (Y_G^1, V_G^1) are two versions of water depth and velocity vectors in the *GPU*, whose components are values at sections, (Y_C, V_C) are the same vectors but in the *CPU*, $CFL(Y,V)$ is the *CFL* condition function for computing time step size, and $JumpStepModel(\Delta t, Y, V)$ and $RegulationPeriodModel(\Delta t, Y, V)$ are the parallel-in-space updating functions, which are described at the end of this section, and λ is the finite time prediction horizon. The algorithm continues until predictive horizon ends.

The algorithm has two loops: the external one implemented in the *CPU*, and the inner in the *GPU*. The NVIDIA compiler (*nvcc*) separates source code into *CPU* and *GPU* components: *GPU* functions are processed by *nvcc*, and the *CPU* functions are processed by standard compiler, like the *gcc*.

The following comments about the algorithm can be made:

- Theoretically, the *CFL* condition has to be satisfied always, that is, at all time steps. Due to the close-to-steady flow nature, it is possible to update the time step size once (*CPU external loop* step 3), just before getting into the *GPU inner loop*. Since the largest changes in flow conditions are produced when control devices are re-positioned, updating the time step size has been located just after *Jump step* computation. Its computation is avoided in the inner loop. Vectors are accessed at global memory (herein, referred as *CPU-memory*).
- Instances of communication and synchronization between different sub-tasks are usually some of the biggest obstacles to obtaining good performance in parallel programming. Updating the time step size according to the *CFL* condition is considered as one example of these because it cannot be fully parallelized. Searching a minimum value of a vector is a task non-parallelizable by definition since the candidate to be a minimum has to be compared with all components one by one. This is another reason why the *CFL* has also been placed into the *CPU*.
- *CUDA* threads may access data from multiple memory spaces during their execution. Although each thread can access the same global memory, they have a private local memory (herein, referred to as *GPU-memory*), which is considered to be accessed in a way faster than in the global. Threads launched by the kernel *RegulationPeriodModel* to compute Y_G^1 and V_G^1, have access to read Y_G^0 and V_G^0 from the private local memory. The kernel *RegulationPeriodModel*(Δt_G, Y_G^0, V_G^0) has been programmed in a way that it never needs to have access to global memory, and it only uses local private memory. That is shown in the *GPU-inner loop* steps 4 and 5.
- Optimizing memory usage starts with minimizing data transfers with low bandwidth between the *CPU* and the *GPU* since large time intervals are spent on data movement rather than arithmetic. The *CPU-GPU* data transfer, and vice versa, has been limited once every external loop iteration.
- Applying the Hartree method to the *1D-SVE* results in an explicit system of two linear equations for each computational node. This feature makes this method very suitable for parallelization. Moreover, boundary and interior conditions can be linearized to preserve this characteristic. Moreover, since within a *Regulation Period* control variables keep constant, governing equations are simplified because their time-derivatives are canceled. Lumping together whole equations, a fully parallelizable set is obtained. The subroutine for solving the set is denoted by *RegulationPeriodModel* (*GPU inner loop* step 2). Details of how this set is obtained is one objective of this section. Vectors are accessed at local private memory.
- Instantaneous changes in hydraulic variables are represented as piece-wise functions of time. Since the equations are nonlinear, they cannot be linearized because their variables are not continuous, and then their derivatives do not exist. Therefore, the original equations have to be solved by using iterative techniques. The set solving subroutine is called *JumpStepModel* function. This kind of method is inadequate for parallelization since the number of iterations required by each node depends on flow conditions and every time step. This is the reason why *JumpStepModel* has been isolated in the *CPU*, at *external loop* step 2.
- In the second step of *GPU inner loop*, vectors (Y_G^0, V_G^0) and (Y_G^1, V_G^1) are alternatively exchanged to avoid having to update every iteration, and only vector pointers are passed through the function as parameters. This fact speeds up the inner loop.

In the rest of the section, numerical formulation details about the Hartree method and boundary conditions are given. It shows how these three functions are compiled: *JumpStepModel*(Δt,*Y*,*V*), *RegulationPeriodModel*(Δt,*Y*,*V*), and *CFL*.

Governing equations for 1D free-surface flow

The *1D-SVE* describe the free-surface flow in prismatic canals. They are the result of the application of the

principles of mass and momentum conservation:

$$\left.\begin{aligned} \frac{\partial y}{\partial t} + v\frac{\partial y}{\partial x} + \frac{A(y)}{T(y)}\frac{\partial v}{\partial x} = 0 \\ \frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} + g\frac{\partial y}{\partial x} = g[S_0 - S_f(y, v)] \end{aligned}\right\} \quad (1)$$

where x and t are the independent space and time variables, y is the water depth, v is the average velocity, $A(y)$ is the wet area, $T(y)$ is the free-surface width, S_0 is the canal bottom slope, and S_f (y, v) is the slope friction. For a trapezoidal section,

$$\left.\begin{aligned} A(y) = by + my^2 \\ T(y) = b + 2my \\ P(y) = b + 2\sqrt{(1+m^2)}y \end{aligned}\right\} \quad (2)$$

where m is the side slope, b is the bottom width, and $P(y)$ is the wet perimeter.

The system (1) is applicable under flow conditions given in irrigation canals, where:

- the curvature of the free-surface is small (vertical accelerations are neglected, and vertical pressure distribution is hydrostatic),
- the canal slope is supposed sufficiently small such that its *sin* is practically null,
- the energy dissipation term is being specified through Manning's equation which was defined for a steady state:

$$S_f(y, v) = n^2\frac{v|v|}{R_H^{4/3}} \quad \text{with } R_H = \frac{A(y)}{P(y)} \quad (3)$$

where n is Manning's coefficient, R_H is the hydraulic radius, and S_f is the friction slope.

- the changes in flow conditions are not fast enough to generate wave fronts.

Realize that flow conditions required by *1D-SVE* to be applied are similar to that are often observed in irrigation canals. The *MOC* transforms the set (1) of partial differential equations into an ordinary differential equation set, named

$$\left.\begin{aligned} &\text{(a)}\ \frac{dv}{dt} + \frac{g}{c(y)}\frac{dy}{dt} = g[S_0 - S_f(y, v)] \\ &\text{(b)}\ \frac{dx^+}{dt} = v + c(y) \\ &\text{(c)}\ \frac{dv}{dt} - \frac{g}{c(y)}\frac{dy}{dt} = g[S_0 - S_f(y, v)] \\ &\text{(d)}\ \frac{dx^-}{dt} = v - c(y) \end{aligned}\right\} \quad (4)$$

with:

$$c = \sqrt{\frac{gA(y)}{T(y)}}$$

The mathematical process of transformation of system (1) to the equivalent (4) can be found in many references, like Ames (1977) or Ducheteau & Zachmann (1988), so will not be discussed further here. The system (4) describes the conditions of flow in a canal in the same way as (1), and it adds no new hypothesis in the transformation. However, it is limited in the way it is applied. The variable x, which was initially independent, is now dependent on time t, as it is understood in (4); then, (4(a)) will be true only along the curves, denoted by x^+, that satisfy the equation (4(b)) and, in the same way (4(c)) will be true along the curves, denoted by x^-, that satisfy the equation (4(d)). In this way, the four equations of the system must be solved simultaneously.

Linear space interpolations

The Hartree method belongs to the *MOC* family, and it uses linear interpolations in the space direction (Figure 2).

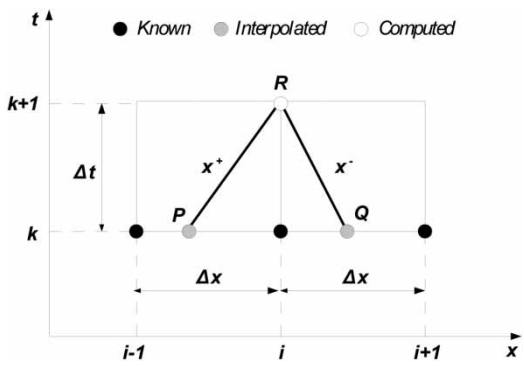

Figure 2 | Numerical resolution for the Hartree method.

The method basically consists of replacing the time-derivatives in (4) by the corresponding finite differences, giving the following algebraic set:

$$\left.\begin{array}{c}\text{(a) } v_R - v_P + \dfrac{g}{c(y_P)}(y_R - y_P) = g\Delta t[S_0 - S_{fP}] \\ \text{(b) } x_R - x_P = \Delta t[v_P + c(y_P)] \\ \text{(c) } v_R - v_Q - \dfrac{g}{c(y_Q)}(y_R - y_Q) = g\Delta t[S_0 - S_{fQ}] \\ \text{(d) } x_R - x_Q = \Delta t[v_Q - c(y_Q)]\end{array}\right\} \quad (5)$$

where $S_{fP} = S_f(y_P, v_P)$ and $S_{fQ} = S_f(y_Q, v_Q)$. (5(b)) makes it possible to define the interpolated values at point P, as follows:

$$\left.\begin{array}{l} y_P = y_i^k - \dfrac{(v_i^k + c_i^k)(y_i^k - y_{i-1}^k)}{(\Delta x/\Delta t) + (v_i^k - v_{i-1}^k) + (c_i^k - c_{i-1}^k)} \\ v_P = v_i^k - \left(\dfrac{v_i^k - v_{i-1}^k}{y_i^k - y_{i-1}^k}\right)(y_i^k - y_P) \\ c_P = c_i^k - \left(\dfrac{c_i^k - c_{i-1}^k}{y_i^k - y_{i-1}^k}\right)(y_i^k - y_P)\end{array}\right\} \quad (6)$$

where

$$c_i^k = c(y_i^k) \text{ and } c_{i-1}^k = c(y_{i-1}^k) \quad (7)$$

In the same way, similar expressions are obtained from (5(d)) for the point Q, as follows:

$$\left.\begin{array}{l} y_Q = y_i^k - \dfrac{(v_i^k - c_i^k)(y_{i+1}^k - y_i^k)}{(\Delta x/\Delta t) + (v_{i+1}^k - v_i^k) - (c_{i+1}^k - c_i^k)} \\ v_Q = v_i^k + \left(\dfrac{v_{i+1}^k - v_i^k}{y_{i+1}^k - y_i^k}\right)(y_Q - y_i^k) \\ c_Q = c_i^k + \left(\dfrac{c_{i+1}^k - c_i^k}{y_{i+1}^k - y_i^k}\right)(y_Q - y_i^k)\end{array}\right\} \quad (8)$$

where

$$c_{i+1}^k = c(y_{i+1}^k) \quad (9)$$

When y_P, v_P, c_P, y_Q, v_Q and c_Q have been computed, values at point R can be obtained by using expressions (5(a)) and (5(c)), as follows:

$$\begin{bmatrix} 1 & \dfrac{g}{c_P} \\ 1 & \dfrac{-g}{c_Q} \end{bmatrix} \begin{pmatrix} v_i^{k+1} \\ y_i^{k+1} \end{pmatrix} = \begin{bmatrix} 1 & \dfrac{g}{c_P} & 0 & 0 \\ 0 & 0 & 1 & \dfrac{-g}{c_Q} \end{bmatrix} \begin{pmatrix} v_P \\ y_P \\ v_Q \\ y_Q \end{pmatrix} + \begin{pmatrix} g\Delta t[S_0 - S_{fP}] \\ g\Delta t[S_0 - S_{fQ}] \end{pmatrix} \quad (10)$$

The solution at instant $k+1$ for node i is explicit and it depends only on the values at the three nodes $i-1$, i, and $i+1$, at instant k. Since these values are stored in a vector, they are closely stored in the memory, and the processor can quickly access them. In contrast, that does not occur when using implicit methods, where the solution at instant $k+1$ for node i depends on whole vectors at $k-1$ and k. Another interesting property of (10) is that the system can be solved in a fixed number of float operations for all i. Both are key factors for a good parallelization because processing time remains constant for all nodes i at time instant k, and they are simultaneously solved by the corresponding codes without synchronization requirements. It can launch one thread for one node without waiting time for synchronization.

The free-surface flow in open canals is governed by the *1D-SVE*, but other equations are needed to describe the flow throughout structures like gates, siphons, joints, jumps, weirs, off-takes, etc. Appropriate boundary conditions need to be specified at off-takes, check gates, head and tail end of the canal system, so that, additional relationships are also needed for describing upstream and downstream boundary conditions, and internal conditions for checkpoints located throughout the canal.

Upstream external boundary condition (gate at canal header)

The incoming flow rate to the irrigation canals is often regulated by means of gates located at the canal header, and the trajectory of the water level just upstream of the gate is known. Thus, upstream boundary condition can be defined by using one of the following gate-discharge relationships

(Bautista & Clemmens 1999):

$$\begin{aligned}&\text{(a) } A(y)\cdot v - b_g c_d u\sqrt{2g[H_{up}(t)+d-y]} = 0 \text{ if } y > d+0.5u\\ &\text{(b) } A(y)\cdot v - b_g c_d u\sqrt{2g[H_{up}(t)+d-0.5u]} = 0 \text{ if } y \le d+0.5u\end{aligned} \tag{11}$$

where Equation (11a) is defined for submerged flow conditions and (11b) for free orifice flow; $H_{up}(t)$ = water level trajectory at boundary, defined by a piece-wise function; $A(y)\cdot v$ = undershot sluicegate flow; c_d = discharge coefficient; b_g = gate width; d = drop at gate; u = gate opening; and, y = water depth downstream gate.

Flow in irrigation canals is usually sub-critical, and then, the second equation in (10) has to be satisfied along the x^- (Figure 3). According to this, and taking into account uniqueness theorems for hyperbolic systems (see e.g., Crandall 1956), the second equation in (10) together with (11) has to be satisfied at the same time.

Interpolating flow conditions at Q (y_Q and v_Q) by using (8) makes it possible to have a relationship between y_R and v_R at boundary point R. Therefore, a new set for solving the external upstream boundary condition is posed, namely:

$$\left.\begin{aligned}&f_{UP}(v_1^{k+1}, y_1^{k+1}) = 0\\ &v_1^{k+1} - v_Q - \frac{g}{c_Q}(y_1^{k+1} - y_Q) - g\Delta t[S_0 - S_{fQ}] = 0\end{aligned}\right\} \tag{12}$$

where f_{UP} is a generalized expression which represents one of (11), or any other upstream boundary condition.

The set (12) is nonlinear in v_1^{k+1} and y_1^{k+1}, so it has to be solved by using iterative techniques, like the Newton–Raphson method. As mentioned above, this kind of technique is inappropriate for parallelization. In order to avoid using it, a linearization of the equations around v_1^k and y_1^k, is carried out. Since the opening gate $u(t)$ and $H_{UP}(t)$ are defined as piece-wise functions of time, there are time instants where functions are not continuous, that is at *Jump Step*. This is a problem with linearization because their derivatives do not exist. Therefore, it is mandatory to use iterative techniques for solving (12) to solve *Jump Step*. This is the first reason why *Jump Steps* are solved at *CPU* external loop step 2.

Inside of a *Regulation Period* the system (12) can be linearized. Then, taking first order time derivatives, it gives:

$$\begin{aligned}&\text{(a) } A(y)\cdot\frac{dv}{dt} + \left[v\cdot T(y) + \frac{gbc_d u}{h_1}\right]\frac{dy}{dt} = [bc_d h_1]\frac{du}{dt} + \left[\frac{gbc_d u}{h_1}\right]\frac{dH_{up}}{dt} \text{ if } y > d+0.5u\\ &\text{(b) } A(y)\cdot\frac{dv}{dt} + v\cdot T(y)\cdot\frac{dy}{dt} = \left[bc_d h_2 - \frac{0.5gbc_d u}{h_2}\right]\frac{du}{dt} + \left[\frac{gbc_d u}{h_2}\right]\frac{dH_{up}}{dt} \text{ if } y \le d+0.5u\\ &\text{where } h_1(H_{up}, y) = \sqrt{2g(H_{up}+d-y)} \text{ and } h_2(H_{up}, y) = \sqrt{2g(H_{up}+d-0.5u)}.\end{aligned} \tag{13}$$

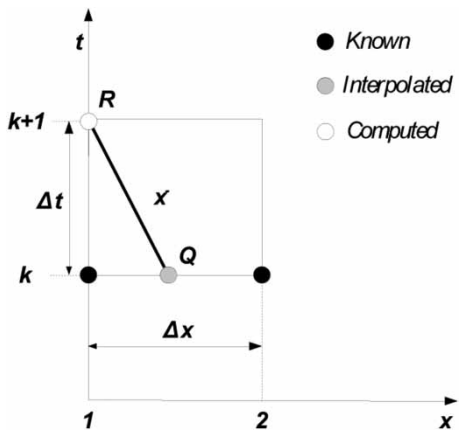

Figure 3 | Numerical resolution for upstream boundary.

Since H_{up} and u keep constant, time-derivatives become null, and (13) can be simplified. Then, taking finite differences around (y_1^k, v_1^k) in (13), gives:

$$\begin{aligned}&\text{(a) } A(y_1^k)\cdot\Delta v + \left[v_1^k T(y_1^k) + \frac{gbc_d u^k}{h_1(H_{up}, y_1^k)}\right]\Delta y = 0 \text{ if } y_1^k > d+0.5u\\ &\text{(b) } A(y_1^k)\cdot\Delta v + [v_1^k T(y_1^k)]\cdot\Delta y = 0 \text{ if } y \le d+0.5u\\ &\text{where: } \Delta y = y_1^{k+1} - y_1^k\ ;\ \Delta v = v_1^{k+1} - v_1^k\end{aligned} \tag{14}$$

A new 2×2 system constituted by (14) and (5(c)) can be compiled, as follows:

$$\begin{bmatrix} A(y_1^k) & v_1^k T(y_1^k)+q_{UP} \\ 1 & \dfrac{-g}{c_Q} \end{bmatrix} \begin{pmatrix} v_1^{k+1} \\ y_1^{k+1} \end{pmatrix} = \\ = \begin{bmatrix} A(y_1^k) & v_1^k T(y_1^k)+q_{UP} & 0 & 0 \\ 0 & 0 & 1 & \dfrac{-g}{c_Q} \end{bmatrix} \begin{pmatrix} v_1^k \\ y_1^k \\ v_Q \\ y_Q \end{pmatrix} + \begin{pmatrix} 0 \\ g\Delta t[S_0 - S_{fQ}] \end{pmatrix} \tag{15}$$

where, for the case $y_i^k > d + 0.5u$:

$$q_{UP} = \frac{gbc_d u^k}{h_1(H_{up}, y_1^k)} \tag{16}$$

and for the case $y < d + 0.5u$, $q_{UP} = 0$. The determinant of the square matrix at the left-hand side of (15) never vanishes, so that, the solution can always be found.

Downstream external boundary condition (canal tail)

At the downstream end, canals often discharge water to a storage basin where a pumping station is usually found. Moreover, an orifice exists to empty the canal. Taking $Q_{dw} = Q_{dw}(t)$ as the scheduled pumped off-take withdrawal, the downstream external boundary condition can be defined according to the following mass conservation equation:

$$\begin{aligned} &\text{(a) } A(y)\cdot v - k_o\sqrt{y - y_0} - Q_{dw}(t) = 0 \text{ if } y > y_0 \\ &\text{(b) } A(y)\cdot v - Q_{dw}(t) = 0 \text{ if } y \le y_0 \end{aligned} \tag{17}$$

where Equation (17a) is for the case in which the orifice be operative, and (17b) for the non-operative case; $A(y)\cdot v =$ downstream canal tail discharge; $k_o =$ discharge coefficient of the orifice; $y_o =$ center of the orifice from canal bottom; and $Q_{dw}(t) =$ pump discharge hydrograph, defined as a piece-wise function.

The first equation in (10) has to be satisfied along x^+ in Figure 4, and (17) has to be satisfied at the boundary. Interpolation flow conditions at P (y_P and v_P) make it possible to have a relationship between y_R and v_R at point R. Therefore,

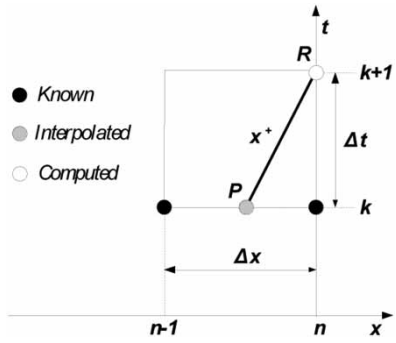

Figure 4 | Numerical resolution for downstream node.

a new set for solving the external downstream boundary condition is posed, namely

$$\left.\begin{aligned} f_{DW}(v_n^{k+1}, y_n^{k+1}) = 0 \\ v_n^{k+1} - v_P + \frac{g}{c_P}(y_n^{k+1} - y_P) - g\Delta t[S_0 - S_{fP}] = 0 \end{aligned}\right\} \tag{18}$$

where f_{DW} is a generalized expression which represents (17) or any other downstream boundary condition.

At a *Jump Step*, (18) has to be solved at *CPU* by Newton–Raphson as it happens in the upstream boundary conditions. However, inside of a *Regulation Period*, the system can be linearized and solved in the *GPU*, since $Q_{DW}(t)$ keeps constant. Thus, taking first order time-derivatives in (18), it becomes

$$\begin{aligned} &\text{(a) } A(y)\cdot\frac{dv}{dt} + \left[v\cdot T(y) - \frac{k_o}{2\sqrt{(y - y_o)}}\right]\frac{dy}{dt} - \frac{dQ_{dw}}{dt} = 0 \text{ if } y > y_o \\ &\text{(b) } A(y)\cdot\frac{dv}{dt} + v\cdot T(y)\cdot\frac{dy}{dt} - \frac{dQ_{dw}}{dt} = 0 \text{ if } y \le y_o \end{aligned} \tag{19}$$

Neglecting dQ_{dw}/dt, and taking finite differences around (y_n^k, v_n^k), obtains

$$\begin{aligned} &\text{(a) } A(y_n^k)\cdot\Delta v + \left[v_n^k T(y_n^k) - \frac{k_o}{2\sqrt{(y_n^k - y_o)}}\right]\Delta y = 0 \text{ if } y_n^k > y_o \\ &\text{(b) } A(y_n^k)\cdot\Delta v + [v_n^k T(y_n^k)]\cdot\Delta y = 0 \text{ if } y_n^k \le y_o \\ &\text{where: } \Delta y = y_n^{k+1} - y_n^k \;;\; \Delta v = v_n^{k+1} - v_n^k \end{aligned} \tag{20}$$

A new 2×2 system for downstream boundary is compiled, namely

$$\begin{bmatrix} 1 & \frac{g}{c_P} \\ A(y_n^k) & v_n^k T(y_n^k) - q_{DW} \end{bmatrix} \begin{pmatrix} v_n^{k+1} \\ y_n^{k+1} \end{pmatrix} = \\ = \begin{bmatrix} 0 & 0 & 1 & \frac{g}{c_P} \\ A(y_n^k) & v_n^k T(y_n^k) - q_{DW} & 0 & 0 \end{bmatrix} \begin{pmatrix} v_n^k \\ y_n^k \\ v_P \\ y_P \end{pmatrix} + \begin{pmatrix} g\Delta t[S_0 - S_{fP}] \\ 0 \end{pmatrix} \tag{21}$$

where, for the case of turned on

$$q_{DW} = \frac{k_o}{2\sqrt{(y_n^k - y_o)}} \tag{22}$$

and for the case of turned off, $q_{DW} = 0$. The determinant of the square matrix at the left-hand side of (21) never vanishes, so that, solution is ensured.

Internal boundary condition (checkpoints)

Checkpoints are cross structures for canal flow control which contain gates in-line, off-take withdrawals, lateral weirs, pumping stations, and so on. Their function is to interfere with stream to regulate flow. Canal models are often sectioned between pools by checkpoints. Flow through checkpoints are usually regulated by gates. Herein, checkpoints are considered to have only a sluicegate and a pumped off-take withdrawal for simplicity, as seen in Figure 5.

Consider a sluice gate in a checkpoint located among two canal pools as Figure 6 shows. On the left, there is the upstream cell defined by points x_n and x_{n-1}, and the downstream one defined by x_1 and x_2. The objective is to compute a solution at $t^{k+1}(y_n^{k+1}, v_n^{k+1}, y_1^{k+1}, v_1^{k+1})$ from the solution at $t^k(y_{n-1}^k, v_{n-1}^k, y_n^k, v_n^k, y_1^k, v_1^k, y_2^k, v_2^k)$, that is, at points R_U and R_D from interpolated variables at P and Q.

The flow interaction of this control structure is described by two relationships, named the mass continuity and gate equations. Both equations are defined in terms of hydraulic variables at points R_U and R_D, as follows:

$$\begin{aligned} &\text{(a) } A(y_n)\cdot v_n - A(y_1)\cdot v_1 - Q_p(t) = 0 \\ &\text{(b)} \begin{cases} A(y_1)\cdot v_1 - b_g c_d u\sqrt{2g\left(y_n + \frac{v_n^2}{2g} + d - y_1\right)} = 0 \text{ if } y_1 > d + 0.5u \\ A(y_1)\cdot v_1 - b_g c_d u\sqrt{2g\left(y_n + \frac{v_n^2}{2g} + d - 0.5u\right)} = 0 \text{ if } y_1 \le d + 0.5u \end{cases} \end{aligned} \tag{23}$$

where $A(y_n)\cdot v_n$ = flow coming from upstream; $A(y_1)\cdot v_1$ = flow going downstream; Q_P = pumped off-take withdrawal; c_d = discharge coefficient; b_g = gate width; d = drop at gate; u = gate opening; y_n = water depth upstream gate; and, y_1 = water depth downstream gate. Equations (23) constitute the so-called internal boundary conditions, and have to be solved together with Equations (10) in a coupled way, giving

$$\begin{aligned} &\text{(a) } f_{N1}(v_1^{k+1}, y_1^{k+1}, v_n^{k+1}, y_n^{k+1}) = 0 \\ &\text{(b) } v_1^{k+1} - v_P + \frac{g}{c(y_P)}(y_1^{k+1} - y_P) = g\Delta t[S_0 - S_{fP}] \\ &\text{(c) } v_n^{k+1} - v_Q - \frac{g}{c(y_Q)}(y_n^{k+1} - y_Q) = g\Delta t[S_0 - S_{fQ}] \\ &\text{(d) } f_{N2}(v_1^{k+1}, y_1^{k+1}, v_n^{k+1}, y_n^{k+1}) = 0 \end{aligned} \tag{24}$$

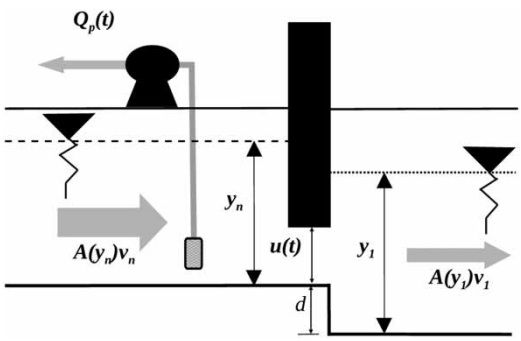

Figure 5 | Checkpoint sketch.

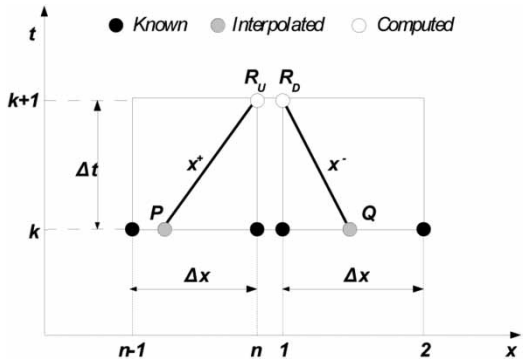

Figure 6 | Numerical resolution for internal boundary.

where f_{N1} and f_{N2} are generalized expressions which represent (23(a)) and (23(b)), respectively, or any other equivalent checkpoint relationship.

To solve this set of four nonlinear equations, iterative techniques must be used, such as the Newton–Raphson method. At *Jump Step* these equations cannot be linearized by the same reasons as above, but inside of a *Regulation Period* they are linearized. Carrying out (24) a similar process of linearization around $(y_1^k, v_1^k, y_n^k, v_n^k)$ like on the external boundary conditions, the following system is obtained:

$$\begin{bmatrix} A(y_n^k) & v_n T(y_n^k) & -A(y_1^k) & -v_1 T(y_1^k) \\ 1 & \frac{g}{c_P} & 0 & 0 \\ 0 & 0 & 1 & \frac{-g}{c_Q} \\ -q_I\left(\frac{v_n^k}{g}\right) & -q_I & A(y_1^k) & v_1^k T(y_1^k) + q_I \end{bmatrix} \begin{pmatrix} v_n^{k+1} \\ y_n^{k+1} \\ v_1^{k+1} \\ y_1^{k+1} \end{pmatrix} =$$
$$= \begin{bmatrix} A(y_n^k) & v_n T(y_n^k) & -A(y_1^k) & -v_1^k T(y_1^k) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -q_I\left(\frac{v_n^k}{g}\right) & -q_I & A(y_1^k) & v_1 T(y_1^k) + q_I \end{bmatrix} \begin{pmatrix} v_n^k \\ y_n^k \\ v_1^k \\ y_1^k \end{pmatrix} +$$
$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & \frac{g}{c_P} & 0 & 0 \\ 0 & 0 & 1 & \frac{-g}{c_Q} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} v_P \\ y_P \\ v_Q \\ y_Q \end{pmatrix} + \begin{pmatrix} 0 \\ g\Delta t[S_0 - S_{fP}] \\ g\Delta t[S_0 - S_{fQ}] \\ 0 \end{pmatrix} \tag{25}$$

where, for the case $y_n^k > d + 0.5u$:

$$q_I = \frac{g b_g^2 c_d^2 u^2}{q_1} \; ; \; q_1 = b_g c_d u \sqrt{2g\left[y_n^k + \frac{(v_n^k)^2}{2g} + d - y_1^k\right]} \tag{26}$$

and $q_1 = 0$, otherwise.

In the context of the parallelization, all nodes are solved at the same time through the corresponding thread, and they have to be finished before starting the next time step, that means synchronization among threads. When solving (25), a 4×4 matrix is inverted for obtaining a double solution, so that, the checkpoint node requires the largest computational time required by any node. Then, it is highly recommendable to optimize the source code for this kind of node. One way for optimizing is to invert the matrix by Cramer's rule, since four elements of 16 are zero, so it is possible to optimize the source code by eliminating many operations. Note that this node has the largest number of floating point operations among all types of nodes defined in this paper. Optimizing reduces computing time significantly.

Courant–Friedericks–Levy stability condition

The Hartree method enables the use of the characteristics method with a fixed grid in the (x,t) plane. Let us assume flow variables are known at time instant k, and that we want to compute the flow at point $k+1$. The *MOC* is subject to the stability *CFL* condition – see Abbott (1979) or Ames (1977), for example – so that, the time step size becomes:

$$\Delta t^k = \Delta x \cdot N_C \cdot Min\left(\frac{1}{|v_i^k \pm c(y_i^k)|}\right) \; \forall i = 0, .., N_x \tag{27}$$

where $N_C \leq 1$ is the so-called Courant number and N_x is the number of cells in which the canal is divided.

Computing Δt^k with (27) is not totally parallelizable because it is necessary to compare all wave-speed values with each other. If a thread for each i to compute wave-speed is launched, it is not able to compare values among them until all threads are finished because threads are working independently from others. However, it could divide the domain into groups of cells, launch a thread per group to obtain the minimum value in the group and, when all threads are finished, compare values among groups. However, this is not a parallelized process, and threads need to use shared memory.

The computation of the *CFL* is the greatest obstacle to getting good parallel program performance, especially when it is calculated always for all time instant k. However, when flow is close-to-steady, it is possible to update the time step size once before getting into a *Regulation Period*. It has been decided to compute the time step size between two consecutive *Regulation Periods*, just after *Jump Step* at the third step of the algorithm. In this way, computation in the *GPU* is avoided. During *Regulation Period* time-size remains constant.

Theoretically, it is possible to take $N_C = 1$ in (27) when the time step size is always computed; but if it is only computed at the *Jump Step*, it can help to take $N_C = 0.95$.

Parallel-in-space updating functions

Lumping together into a set, (10) for all nodes inside of a canal pool, (15) for the upstream node, (21) for the downstream one, and (25) for checkpoints, a model for *Regulation Periods* can be implemented, which is denoted herein by:

$$(Y^{k+1}, V^{k+1}) = RegulationPeriodModel(\Delta t^k, Y^k, V^k) \quad (28)$$

The locality principle from computer science says that items whose addresses are near one another tend to be referenced close together in time. When computing a thread, addresses of y_{i+1}, v_{i+1}, y_i, v_i, y_{i-1}, and v_{i-1} seem to be close enough to follow the locality principle. The updating is carried out by calling a thread per each node i in the *GPU*.

At the same time, lumping together into another set, (10) for standard nodes, (12) for the upstream node, (17) for the downstream one, and (24) for checkpoints, the *Jump Step Model* can be implemented, and denoted by:

$$(Y^{k+1}, V^{k+1}) = JumpStepModel(\Delta t^k, Y^k, V^k) \quad (29)$$

By using one of these models, it is possible to update hydraulic unknowns y_i^{k+1} and v_i^{k+1} for all sections from the variables at instant k. Thanks to this strategy, programming both *CPU* and *GPU* solvers is trivial. The parallel part of the algorithm consists of a single kernel that is processed in the *GPU*, in which each thread calculates the couple y_i^{k+1} and v_i^{k+1}, so it does not present any difficulty. With respect to the *CPU* version implemented in FORTRAN, it only requires the 'omp parallel do' directive before the loop that runs through all the nodes in the inner loop by calling the update function of the instant variables $k+1$; this function is similar to the one implemented in the *GPU* kernel.

ILLUSTRATIVE EXAMPLES

The ASCE Task Committee on Canal Control Algorithms (Clemmens *et al.* 1998) developed a series of test cases to test the general suitability of canal-control scheme logic. The cases consisted of defining several scenarios happening over two canals, and the objective was to find the way to canal control. The test cases covered a reasonable range of the proprieties and operating characteristics of irrigation canals. Among them, there is a control part where flow-rate changes are scheduled, and then, the control problem has feedforward character. In this part, the objective of the tests is to bring the canal from an initial steady-state to a final one.

GoRoSo is a feedforward control algorithm for irrigation canals based on sequential quadratic programming (Soler *et al.* 2013a), with which it is possible to calculate gate trajectories that smoothly carry the canal from an initial steady-state to a final steady-state by keeping water depths constant at overall end points of each pool, that is at checkpoints. Constraints on gate movements are required for stabilizing the control algorithm, and to avoid gate stroking movements which are amenable to generate possible changes in flow regime. The algorithm makes it possible to calculate gate trajectories by constraining gate movements into a user-defined amount. The application of the GoRoSo method to the tests can be found in Soler *et al.* (2013b). The two canals defined in these test cases together with internal and boundary conditions obtained from the feedforward control algorithm define two scenarios for validation of the parallel algorithm.

The first example shows how by using a fine discretization it is possible to overcome the numerical diffusion that is inherited from the Hartree scheme, and the second example is provided in order to show the high performance in terms of computing speed. The second example is also solved by means of the HEC-RAS platform in order to compare simulation results and diffusive errors (https://www.hec.usace.army.mil/software/hec-ras/). The HEC-RAS is a computer program to model the water flows through open canal systems and the calculation of surface water profiles (Barkau 1992). For unsteady flow, HEC-RAS solves the full, dynamic, *1-D SVE* using the above-mentioned box

four-point implicit scheme, first used by Preissmann (1961). Since flow conditions in the scenario are close-to-steady state, and subcritical, HEC-RAS does not show the numerical instability problems associated with regime changes.

To evaluate the numerical diffusion and quantify the diffusive losses generated by the first-order interpolation, it is proposed to apply the conservation mass principle to the whole domain x/t of integration by calculating the following numerical diffusion error index:

$$\varepsilon = \frac{\int_{x=0}^{x=L}[A(\lambda\Delta t, x) - A(0, x)]dx - \int_{t=0}^{t=\lambda\Delta t}[q(t, 0) - q(t, L)]dt - T\sum_{i=1}^{i=8}\sum_{j=1}^{j=\lambda} Q_{P_i}(j) - \int_{t=0}^{t=\lambda\Delta t} q_o[y(t, L)]dt}{\int_{t=0}^{t=\lambda\Delta t}[q(t, 0)]dt} \tag{30}$$

where $Q_{Pi}(j)$ is the pumped off-take withdrawal i during the j^{th}- *Regulation Period*, λ is the number of regulation periods in which the prediction horizon is divided, L is the canal length, and q_0 is the gravity off-take and canal tail defined in (17).

Description of the test cases

Scenarios are implemented on two different canals, each with eight pools separated by checkpoints with sluicegates (Figure 7).

The first canal is based on the 9.5 km-long lateral canal WM within the Maricopa Stanfield Irrigation and Drainage district in Central Arizona. It is a steep canal characterized by high Froude number and little storage. More details are given in Table 1.

The second canal is based on the 28 km-long upstream portion of Corning Canal in California, and is essentially flat and has significant storage. More details are given in Table 2.

Even though original tests present gravity off-takes, withdrawals are modeled as scheduled pumping stations, and take place even when water depth is lower than the orifice center (y_0). This is one of the more difficult situations to control because withdrawal ratios are scheduled and are independent from water depth in the canal. Only the last canal pool is discharged through orifices. Sluicegate discharge coefficients are $c_d = 0.61$. Prediction horizon length is 12 hr in both scenarios. The control objective consists of bringing the canal from an initial steady-state to a final one by following the respective set scheduled gate trajectories.

In the Maricopa Stanfield Canal test, prediction horizon is divided into $\lambda = 144$ regulation periods about $\Delta T = 300$ [s] (5 min), and in the Corning Canal test into $\lambda = 48$ pieces about $\Delta T = 900$ [s] (15 min). Pumping discharge ratios remain constant during the first 2 hours of the prediction horizon until, at instant 2 h, whole ratios change to the final state ratio until the horizon ends. Then, they remain constant again until the prediction horizon ends. The upstream boundary condition keeps constant along the prediction horizon at a water depth: $H_{UP} = 1.1$ [m] for the Maricopa Stanfield case and $H_{UP} = 3.0$ [m] for the Corning Canal case.

Figure 9 shows the gate trajectories that bring both canals from the initial state to the final one described in Figure 8. Feasible gate opening changes are constrained into 3 cm for the case of Maricopa Stanfield and 10 cm for the Corning Canal case.

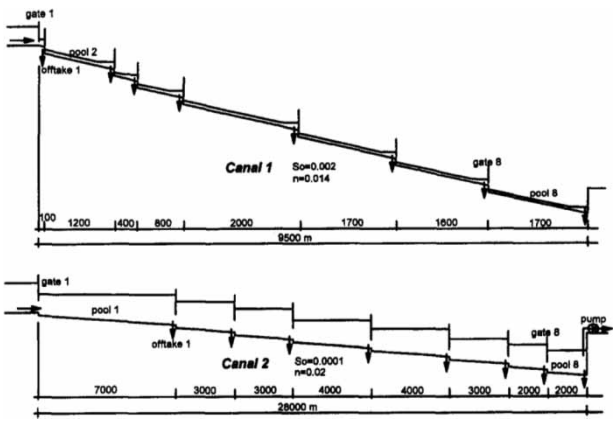

Figure 7 | Canal profiles for the tests (taken from Liu *et al.* (1998)).

Simulation results

When gate trajectories shown in Figure 9 are applied to the models described by the characteristics given in Tables 1 and 2, the evolution of the water depths at checkpoints is shown in Figure 10.

Table 1 | Maricopa Stanfield Canal characteristics

Pool	S_0 (m/m)	Pool length (m)	n	b (m)	m (m/m)	Canal depth (m)	d (m)	k_o, y_o (m)
I	0.0020	100	0.014	1.0	1.5	1.1	1.0	–
II	0.0020	1,200	0.014	1.0	1.5	1.1	1.0	–
III	0.0020	400	0.014	1.0	1.5	1.0	1.0	–
IV	0.0020	800	0.014	0.8	1.5	1.1	1.0	–
V	0.0020	2,000	0.014	0.8	1.5	1.1	1.0	–
VI	0.0020	1,700	0.014	0.8	1.5	1.0	1.0	–
VII	0.0020	1,600	0.014	0.6	1.5	1.0	1.0	–
VIII	0.0020	1,700	0.014	0.6	1.5	1.0	1.0	0.9486833/0.4

Table 2 | Corning canal characteristics

Pool	S_0 (m/m)	Pool length (m)	n	b (m)	m (m/m)	Canal depth (m)	d (m)	k_o/y_o (m)
I	0.0001	7,000	0.020	7.0	1.5	2.5	0.2	–
II	0.0001	3,000	0.020	7.0	1.5	2.5	0.2	–
III	0.0001	3,000	0.020	7.0	1.5	2.5	0.2	–
IV	0.0001	4,000	0.020	6.0	1.5	2.3	0.2	–
V	0.0001	4,000	0.020	6.0	1.5	2.3	0.2	–
VI	0.0001	3,000	0.020	5.0	1.5	1.9	0.2	–
VII	0.0001	2,000	0.020	5.0	1.5	1.9	0.2	–
VIII	0.0001	2,000	0.020	5.0	1.5	1.9	0.2	1.08465229/0.85

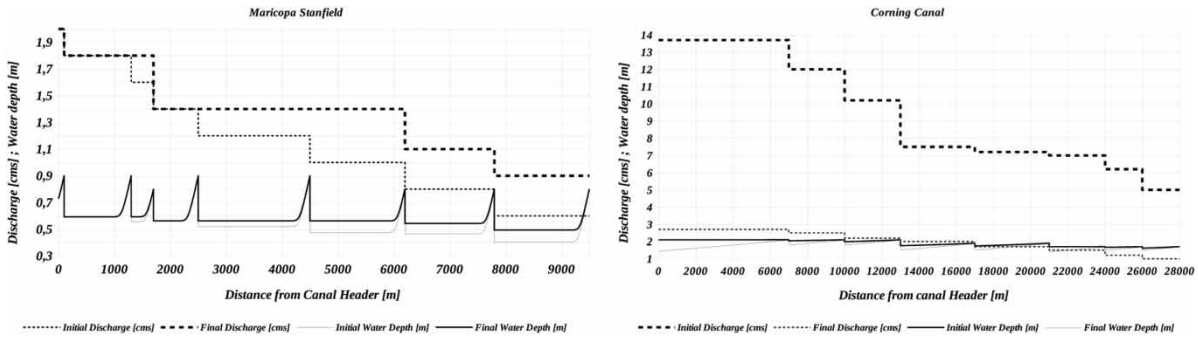

Figure 8 | Initial and final steady-states for the Maricopa Stanfield and Corning Canal test cases.

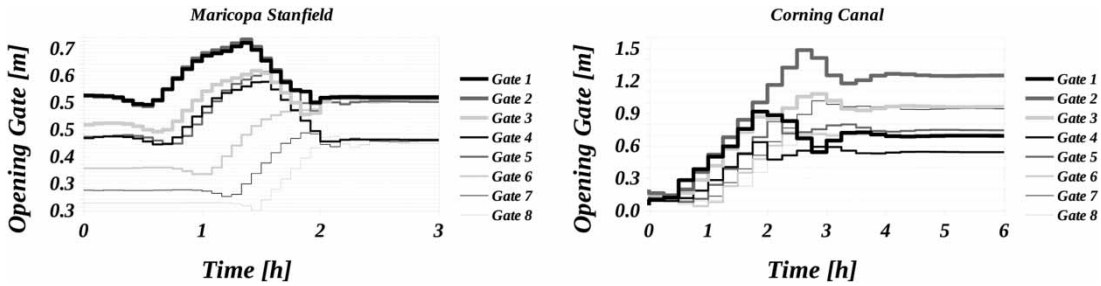

Figure 9 | Piece-wise gate trajectories along the prediction horizon. They are constituted by $\lambda = 144$ pieces of $\Delta T = 300$ s for the Maricopa Stanfield example and by $\lambda = 48$ pieces of 900 s for the Corning Canal case.

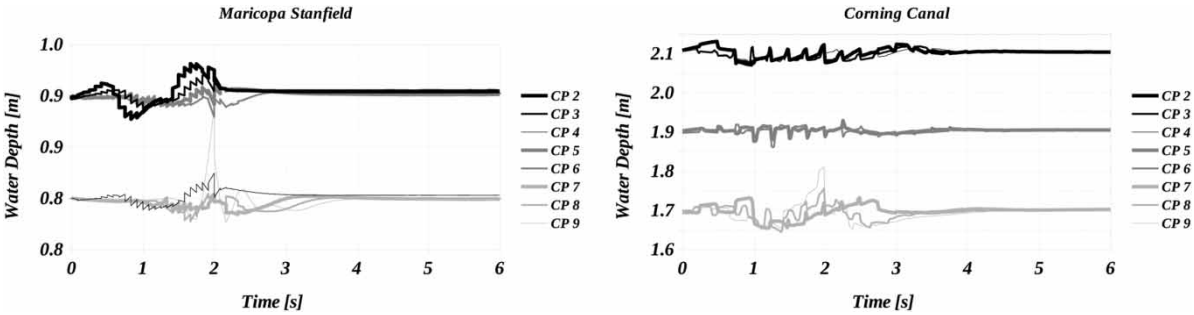

Figure 10 | Water depth hydrographs at checkpoints 2 to 9 obtained by the parallel algorithm along the 12 hr-long prediction horizon.

As seen in Figure 10, water depths show small oscillations around values of the target, which are given in Table 3. These oscillations are sufficiently small to consider being good enough for the regulation. This occurs except at *CP 9*, where the water level rises up about 10 cm from the target for approximately 15 min, but without overflowing.

Maricopa Stanfield diffusive error

The Maricopa Stanfield case has been chosen to analyze the accuracy of the parallel algorithm since the water depth profiles are highly curved, leading to large diffusive errors. The left-hand panel of Figure 8 shows the highly curved water depth profiles for the Maricopa Stanfield scenario. In this situation, using straight lines in interpolation generates significant errors, which can be reduced by using fine discretization. In order to show this, four different levels of discretization have been implemented, that is, from $\Delta x = 10.0$ to $\Delta x = 0.5$. As Table 4 shows, diffusive error is reduced when discretization is fine enough. Therefore, the accuracy of the Hartree scheme is good enough when a fine discretization is used.

A computer with a CPU (Intel® Core™ 2 Duo CPU E7500 @ 2.93 GHz × 2) and a GPU (NVIDIA GeForce GT 710/PCIe/SSE2 710 CUDA cores running at 954 MHz) has been used to simulate these models. As the last column in Table 4 shows, processing times are low enough to use this numerical scheme in the real-time canal control context.

Table 4 | Maricopa Stanfield scenario results

Cell size Δx (m)	Nodes	ϵ	Processing time (s)
10	958	0.04213	2.4
5	1,908	0.02317	5.7
1	9,508	0.00497	63.0
0.5	19,008	0.00251	180.0

Table 3 | Definition of the initial and final steady-states of the prediction horizons

Pool	Maricopa initial and final water depths at downstream (m)	Maricopa initial flow (m^3/s)	Maricopa final flow (m^3/s)	Corning Canal initial and final water depths at downstream (m)	Corning Canal initial steady-state discharge (m^3/s)	Corning Canal final steady-state discharge (m^3/s)
I	0.9	2.0	2.0	2.1	2.7	13.7
II	0.9	1.8	1.8	2.1	2.5	12.0
III	0.8	1.6	1.4	2.1	2.2	10.2
IV	0.9	1.4	1.4	1.9	2.0	7.5
V	0.9	1.2	1.4	1.9	1.7	7.2
VI	0.8	1.0	1.4	1.7	1.5	7.0
VII	0.8	0.8	1.1	1.7	1.2	6.2
VIII	0.8	0.6	0.9	1.7	1.0	5.0

Corning canal scenario diffusive error

When the HEC-RAS model discretized by a space cell size about $\Delta x = 10$ m runs constrained by a time step size lower than 30 s, similar results to the Hartree method are obtained (Figure 11).

From Figure 11, the following comments can be made:

- Simulation results can be considered as equivalent.
- The gate trajectories carry the canal to the final steady state.
- The canal only needs a transient of about 3 hours to achieve the final steady-state.
- The differences between both hydrographs are due to the fact that the gate formulation used in the HEC-RAS model is different from the one used here (11).
- Curves are drawn using 721 points, that is a point every minute.

As Table 5 shows, the parallel algorithm has an equivalent diffusive error as the HEC-RAS model.

Table 5 | Numerical diffusive error index from parallel algorithm and HEC-RAS model

Parallel algorithm	$\varepsilon = 0.0007987$
HEC-RAS model	$\varepsilon = 0.0007374$

Corning canal scenario processing time evaluation

To test the parallel algorithm performance, five Corning Canal models with different level of spatial discretization were implemented, and the processing time required by each model was compared.

Two parallel versions were implemented, one in *CPU* and one in *CPU-GPU*, the first being a version implemented in FORTRAN and the second one using C-CUDA. The characteristics of the hardware where the tests were run are:

Equipment 1: A computer with a *CPU* (Intel® Core™ 2 Duo CPU E7500 @ 2.93 GHz × 2) and a *GPU* (NVIDIA GeForce GT 710/PCIe/SSE2 710 CUDA cores running at 954 MHz). It can be considered as a representation of the cheapest equipment that can found in the market.

Equipment 2: A computer Dell PowerEdge R720 (2 processor Intel Xeon (R) CPU E5-2620v2 @ 2.1 GHz × 6) and a *GPU* (NVIDIA K40 GPU 2880 CUDA cores running at 745 MHz, Kepler Architecture, 12GB DDR5). This can be considered as a moderate server of a data center.

Equipment 3: A computer Dell PowerEdge R740 (2 Intel Xeon Gold 6138 @ 2 GHz × 20 processor) and a *GPU* (NVIDIA Tesla V100 GPU 5120 CUDA cores running at 1,370 MHz, 32GB DDR5). This corresponds to a last generation server from a data center.

Performance test 1

In this test, the processing time corresponding to three levels of spatial discretization are compared ($\Delta x = 10$ m, $\Delta x = 5$ m, and $\Delta x = 1$ m). For this test, three different solvers were implemented. The first is a serial version on *CPU* using the FORTRAN language, the second is a parallel *CPU* implementation using OpenMP with a shared memory strategy, and the third is a parallel version in C-CUDA which runs on *CPU-GPU* as described in the 'Parallel algorithm' section. It is worth mentioning that the gfortran-7 version is used for the *CPU* code, so the code is translated to C by the compiler before compilation. On the other hand, the

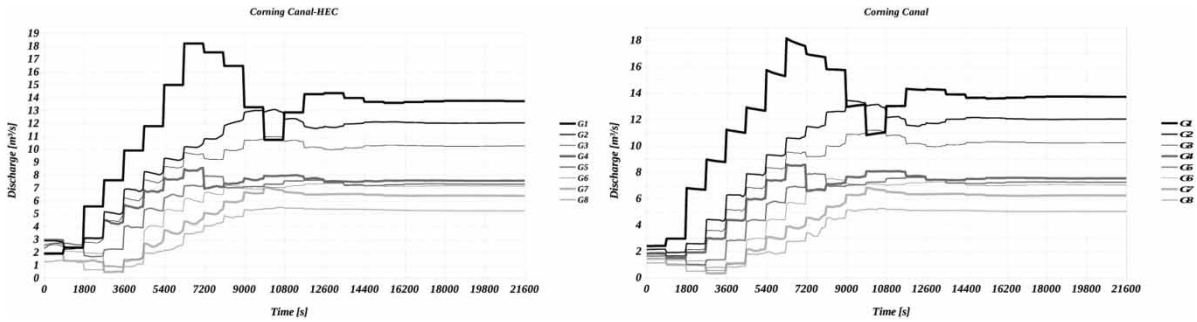

Figure 11 | Discharge hydrograph by HEC-RAS model (left) and parallel algorithm (right).

parallel *CPU* version is compiled with the '-fopenmp' directive, that allows activation of the parallel execution. The three solvers were run on Equipment 1, 2, and 3, using 2, 12, and 40 CPU-cores for the parallel *CPU* version, respectively, while the *CPU-GPU* version uses the graphics card of each equipment. Table 6 summarizes the execution times obtained in each case.

From Table 6, the following comments can be made:

- When the HEC-RAS model runs on Equipment 1 constrained with a user-defined time step size about 30 s, it takes about 17 s to complete the simulation. When the step size is 10 s, it takes about 28 s. It is important to note that only a HEC-RAS serial version is available.
- The processing times for the serial version of the code are similar in the three sets of computer equipment. This is due to the fact that the clock frequencies of the *CPUs* have not evolved in recent years (2006, 2013, 2017, respectively, for each *CPU*), moreover, Equipment 2 has a lower clock frequency consuming more time. Regarding the parallel *CPU* version, it is shown that the code scales considerably with the number of parallel processes, obtaining gains of 1.9, 9.1, and 17.6 compared to the serial versions in the case of $\Delta x = 1$ m, which is the most expensive computationally.
- For the *GPU* versions, a substantial improvement can be seen in computing time regarding the parallel solver in *CPU*. The speed-up obtained with the different *GPU* versions is compared with the faster *CPU* parallel version (Equipment 3). The speed-up for the GT710 card is 0.6 for all discretization levels, due to the saturation of the relatively few *GPU* cores that the card has. Speed-ups from 2.5 to 11.5 were obtained on the K40 card and from 7.3 to 47.2 on the V100 card, showing that the code scales very well when refining the mesh and/or increasing the number of *GPU* cores available.
- As the finest case ($\Delta x = 1$ m) shows, it is possible to solve relatively large problems on *GPU* in a reasonable time. Moreover, when using the parallel *GPU* version, the time required to solve the problem depends only on the number of available threads. Since this method implies trivial parallelism, it would be easy to use multi-*GPU* allowing larger problems to be solved over long periods of time. Implementation of the Hartree method in fine meshes reduces interpolation errors and, according to the times obtained in *GPU*, it is suitable to use on optimization problems.
- Finally, it is worth mentioning that the cost of a small card like the GT710 (2017) is around 50 USD, while the Xeon Gold 6138 (2017) processors are around 5,224 USD, considering that the processing time for both configurations is of the same order (speed-up of 0.6 in favor of the *CPU*). If the costs of the K40 and V100 cards (approximately USD 750 and USD 10,000, respectively) are considered, a speed-up of 11.5 and 47.2 in the processing time may not be as convenient from a cost/speed-up point of view. However, we emphasize that a hydraulic moderator can obtain acceptable results in terms of time and precision at a very low cost when using low-profile graphics cards.

Table 6 | Processing time of the performance test 1

Equipment	Cell size Δx # Nodes	10 m 2,808	5 m 5,608	1 m 28,008	# Cores
1	Serial version	25.80 s	103.8 s	2,672.4 s	1 (CPU)
	Parallel CPU	15.01 s	55.80 s	1,382.4 s	2 (CPU)
	Parallel GPU	3.44 s	10.57 s	232.7 s	710 (GPU)
2	Serial version	42.22 s	169.6 s	4,193.5 s	1 (CPU)
	Parallel CPU	5.07 s	19.38 s	460.9 s	12 (CPU)
	Parallel GPU	0.837 s	1.705 s	12.34 s	2,880 (GPU)
3	Serial version	25.17 s	100.2 s	2,501.2 s	1 (CPU)
	Parallel CPU	2.122 s	6.976 s	142.0 s	40 (CPU)
	Parallel GPU	0.288 s	5.578 s	3.011 s	5,120 (GPU)

Performance test 2

The purpose of the second test is to check the relationship between the processing time required by a time step and the cell size of the model, while keeping the number of time steps constant. As required by the *CFL* condition, the time step size Δt depends on the cell size Δx, and if the number of time steps is required to be changed, only the processing time can be modified. However, since the Hartree scheme is linear, one can easily estimate the processing time in an approximate way. For example, by doubling the cell size the processing time must be divided by 2 to obtain the same number of time steps.

Five numerical models were implemented, each one with different discretization levels ($\Delta x = 1$ m, $\Delta x = 2.5$ m,

$\Delta x = 5$ m, $\Delta x = 7.5$ m, and $\Delta x = 10$ m). A processing time has been determined for each level. The tests were performed on Equipment 1 using the *GPU* parallel solver. Results can be found in Table 7.

Viewing the last column, it is possible to conclude that each cell size has its own approximate ratio η, independently from *Prediction Horizon*, so that, η depends uniquely on cell size Δx. It means that the processing time has to be proportional to *Prediction Horizon*, as expected. Figure 12 shows the curve η vs Δx which is obtained from Table 7.

From Figure 12, it can be concluded that the curve η vs Δx is clearly nonlinear. The ratio η by the models $\Delta x = 5$ m, $\Delta x = 7.5$ m, and $\Delta x = 10$ m shows small variations among each other, but models $\Delta x = 1$ m, $\Delta x = 2.5$ m, and $\Delta x = 5$ m show larger variations. That could indicate a limit point below which, the ratio of time-per-step η increases strongly. That suggests the existence of a trend change limit point. Obviously, this point is associated with a specific hardware, in this case, Equipment 1, and decays considerably in $\Delta x =$ 2.5 m. In other words, discretization below $\Delta x = 5$ m mean that this execution configuration does not have enough 'parallel capacity' and, therefore, the model is inappropriate for such configuration.

Table 7 | Results of the performance test 2

Δx (m)	*PH* (hr)	*TS* (step)	*PT* (s)	$\eta = PT/TS$ (ms/step)
1.0	1.2	28,498	24.216	0.850
2.5	3	27,390	9.828	0.359
5.0	6	27,375	5.263	0.192
7.5	9	27,366	3.829	0.140
10.0	12	27,358	3.439	0.126
1.0	1.75	39,825	33.915	0.852
2.5	4	36,514	13.116	0.359
5.0	8	36,485	7.188	0.197
7.5	12	36,470	5.146	0.141
10.0	16	36,461	4.294	0.118
1.0	2.5	57,145	48.548	0.850
2.5	6	54,740	19.739	0.361
5.0	12	54,689	10,567	0.193
7.5	18	54,662	7,634	0.140
10.0	24	54,637	6,433	0.118

Δx [m] = cell size; *PH* [hr] = prediction horizon; *TS* [st] = total number of time steps; *PT* [ms] = processing time; and time-per-step ratio $\eta = PT/TS$ in milliseconds per step.

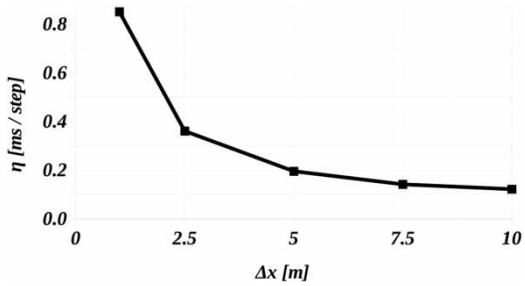

Figure 12 | Trial 2 results: time-per-step ratio *vs* level of discretization.

SUMMARY AND CONCLUSIONS

Due to its lack of precision, using the Hartree method was dropped in open canal flow simulations. However, it should be taken into account that giving up this method occurred in a context with less computing power, where the thick grid used was mandatory and the interpolation error very large. The Maricopa Stanfield benchmark test 1.2 by the ASCE Task Committee on Canal Automation Algorithms has been used for evaluating the accuracy of the parallel algorithm. The example has been chosen since water depth profiles are highly curved, and this fact is amenable to obtaining large diffusive errors. It has been found that diffusive error is reduced when discretization is fine enough. Therefore, the accuracy of the Hartree scheme can be considered good enough when a fine discretization is used. The use of the parallel codes makes it possible to implement models with a high level of discretization that can be solved in relatively small times.

Parallel codes in *CPU* and *GPU* were developed for the resolution of free-surface flow for open channels. Numerical parallelization was achieved by using the explicit Hartree method applied to the Saint-Venant equations, and by making linear the equations for external and internal boundary conditions. As the trajectories of the gates, boundary conditions, and off-take withdrawals can be modeled with piece-wise functions due to the flow behavior in irrigation canals, the prediction horizon has been discretized in *Regulation Period* and *Jump Step*. Taking these ideas into account, a solver has been implemented that solves the

non-linear *Jump Step* part on the *CPU* and the linear *Regulation Period* part on the *GPU*.

The Corning Canal benchmark test 2.2 by the ASCE Task Committee on Canal Automation Algorithms was used for the performance evaluation. It was found that the parallel version in C-CUDA in *GPU* allows the problem to be solved in much less time than the FORTRAN version in *CPU*.

Finally, a study of the sensitivity of the ratio of computation speed to the level of discretization has been carried out. As a result, the existence of a trend change limit point is suggested. This point is associated with specific computer equipment used by the calculations. Discretization below this point gives lower speed ratios and their behavior seems to be linear and less sensitive to changes in discretization. In contrast, dense discretization is more sensitive to changes in the cell size.

DATA AVAILABILITY STATEMENT

All relevant data are included in the paper or its Supplementary Information.

REFERENCES

Abbott, M. B. 1979 *Computational Hydraulics. Elements of the Theory of Free Surface Flows*. Pitman Publishing, London, 324 p.

Aissa, M., Verstraete, T. & Vuik, C. 2017 Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes. *Computers & Mathematics with Applications* **74** (1), 201–217.

Ames, W. F. 1977 *Numerical Methods for Partial Differential Equations*. Academic Press Inc., Orlando, FL, USA.

Anders, A. & Kandrot, E. 2010 *CUDA by Example. An Introduction to General-Purpose GPU Programming*. Pearson Education, Boston, MA, USA.

Barkau, R. L. 1992 *UNET, One-Dimensional Unsteady Flow Through a Full Network of Open Channels*. Computer Program, St. Louis, MO, USA.

Bautista, E. & Clemmens, A. J. 1999 Response of ASCE task committee test cases to open-loop control measures. *Journal of Irrigation and Drainage Engineering* **125** (4), 179–188.

Bessone, L., Gamazo, P. & Storti, M. 2018 Performance evaluation of different schemes for the resolution of a nonlinear diffusion equation in GPGPU. *Mecánica Computacional* **XXXVI**, 605–625.

Bondarenco, M., Gamazo, P. & Ezzatti, P. 2017 A comparison of various schemes for solving the transport equation in many-core platforms. *The Journal of Supercomputing* **73** (1), 469–481.

Clemmens, A. J., Grawitz, B. & Schuurmans, W. 1998 Test cases for canal control algorithms. *Journal of Irrigation and Drainage Engineering* **124** (1), 23–30.

Cozzolino, L., Cimorelli, L., Covelli, C., Morte, R. D. & Pianese, D. 2015 The analytic solution of the Shallow-Water Equations with partially open sluice-gates: the dam-break problem. *Advances in Water Resources* **80**, 90–102. http://doi.org/10.1016/j.advwatres.2015.03.010.

Crandall, S. H. 1956 *Engineering Analysis – A Survey of Numerical Procedures*. Wiley Interscience, New York, USA.

Ducheteau, P. & Zachmann, D. W. 1988 *Ecuaciones Diferenciales Parciales. (Partial Differential Equations)*. McGraw-Hill, Mèxico City, Mexico.

Gómez, M. 1988 *Contribución al estudio del movimiento variable en lámina libre, en las redes de alcantarillado. Aplicaciones. (Contribution to Study of the Unsteady Free Surface Flow in Sewer Systems. Applications.) Doctoral thesis*, UPC, Barcelona, Spain.

Hodges, B. R. & Liu, F. 2019 Timescale interpolation and no-neighbour discretization for a 1D finite-volume Saint-Venant solver. *Journal of Hydraulic Research*. https://doi.org/10.1080/00221686.2019.1671510

Katapodes, N. D. 2016 *Free-Surface Flow: Computational Methods*. Butterworth-Heinemann, Oxford, UK.

Litrico, X. & Fromion, V. 2009 *Modeling and Control of Hydrosystems*. Springer, Dordrecht, Heidelberg, London, New York.

Liu, F., Feyen, J., Malaterre, P. O., Baume, J. P. & Kosuth, P. 1998 Development and evaluation of canal automation algorithm CLIS. *Journal of Irrigation and Drainage Engineering* **124** (1), 40–46. http://doi.org/10.1061/(ASCE)0733-9437(1998)124:1(40).

Martín-Sanchez, J. M. & Rodellar, J. 1996 *Adaptive Predictive Control: From the Concepts to Plant Optimization*. Prentice Hall, London, UK.

NVIDIA Corporation 2012 Cuda C Best Practices Guide, Version 4.1, Santa Clara, CA, USA.

Ostad-Ali-Askari, K., Shayannejad, M. & Eslamian, S. 2018 Comparison of solutions of Saint-Venant equations by characteristics and finite difference methods for unsteady flow analysis in open channel. *International Journal of Hydrology Science and Technology* **8** (3). doi:10.1504/IJHST.2018.10014535.

Preissmann, A. 1961 *Propagation des intumescences dans les canaux et rivières. (Propagation of the Mobile Jump in Canals and Rivers)*. First Congress of the French Association for Computation, Grenoble, France, pp. 433–442.

Soler, J., Gómez, M. & Rodellar, J. 2013a Goroso: a feedforward control algorithm for irrigation canals based on sequential quadratic programming. *Journal of Irrigation and Drainage Engineering* **139** (1), 41–54.

Soler, J., Gómez, M., Rodellar, J. & Gamazo, P. 2013b Application of the GoRoSo feedforward algorithm to compute the gate trajectories for a quick canal closing in the case of an emergency. *Journal of Irrigation and Drainage Engineering* **139** (12), 1028–1036. 10.1061/(ASCE)IR.1943-4774.0000640.

Soler, J., Gómez, M. & Bonet, E. 2015 Canal monitoring algorithm. *Journal of Irrigation and Drainage Engineering* **142** (3). https://doi.org/10.1061/(ASCE)IR.1943-4774.0000982

Whitley, D. 1994 A genetic algorithm tutorial. *Statistics and Computing* **4** (2), 65–85. https://doi.org/10.1007/BF00175354.

Wylie, E. B. 1969 Control of transient free-surface flow. *Journal of the Hydraulics Division* **95**, 347–362.

Yang, J. C. & Chiu, K. P. 1993 Use of characteristics method with cubic interpolation for unsteady-flow computation. *International Journal for Numerical Methods in Fluids* **16**, 329–345.

First received 21 March 2020; accepted in revised form 24 July 2020. Available online 17 September 2020