Parallelization of the SurfWB-UC code for the numerical simulation of nonlinear depth-averaged shallow water waves

José Galaz

Parallelization of the SurfWB-UC code or the numerical simulation of non-linear depth averaged shallow water waves

José Galaz Mora

June 17, 2014

Abstract

The SurfWB-UC model is an implementation of a set of algorithms which aims at solving the Non-Linear Shallow Water equations, which applications vary from river to estuarine and tsunami flows. As a predictive tool, the computed results could be required in real time, and for research, reducing the computation time, say from days to hours, could help improve results and develop a deeper understanding of the simulated phenomena. In this work an MPI parallel version of the code is presented and a performance analysis is made for a particularly simple yet challenging test case, which was run in the cluster machine of the Escuela de Ingeniería at the Pontificia Universidad Católica de Chile. Experiments suggest that the code may be scalable, and allow to achieve an speed up of order 10 for the range of cores currently available in the cluster. Further tests should be run for debugging and handle exceptions and also to complement the performance analysis herein developed.

1 Introduction

The SurfWB-UC code [5, 6] is an algorithm that aims at simulating the propagation of non-linear waves over shallow water by numerically integrating a depth averaged version of the Navier-Stokes equations. Its applications cover a variety of problems such as glacial lake outburst floods, dam break problems, and also the representation of tsunami waves. In general, it is cappable of representing phenomena of incompressible shallow fluids with free surface, given that a characteristic wave length is much greater than a characteristic water depth, hence the term shallow. Also, it is able to accurately represent shock waves, which behave similar to breaking waves, and also wetting and drying processes, which makes of it an ideal tool for studying nearshore processes.

One of the most important applications is in the area of tsunami modelling, where characteristic wave lengths surround $200km$ and a characteristic water depth can be thought of about $4km$, which characterize the ocean as relatively shallow (with a water depth - wave length ratio of 2%). Even though the tsunami wave measures hundreds of kilometers, nearshore we are interested in the scale of $meters$, and so, currently simulating $4hrs$ in a $1.2km$ by $1.2km$ bay (as in Talcahuano, for example), using a $2m$ resolution grid, takes approximately one week of real time [1]. This makes of studing, for example, $24h$ of simulation, a practically impossible task. Thus, there is a need to improve the performance of the code; parallel computing appears then as an attractive solution.

[1] Using an Intel Core i5-3317U CPU @ 1.70GHz x 4 machine with 4GB of RAM

In this work, an MPI parallel implementation of the SurfWB-UC code is presented. In section 2, the numerical model is described in terms of the governing equations and the numerical scheme. Section 3 shows an overview of the algorithm implemented in the original sequential version of SurfWB-UC and the main modifications that were made along with some formal description that allows one to say that the parallelization is load balanced. Section 4 shows a concrete application of the parallelized version in order to study how the performance of the new code behaves. Finally, section 5 summarizes with conclusions and some lines of future work.

2 Numerical model

Let $\Omega \subset \mathbb{R}^3$ be the simulation domain, $t_f > 0$ the simulation time, $\boldsymbol{v} : [0, t_f] \times \Omega \to \mathbb{R}^3$ the velocity flow, and $p : [0, t_f] \times \Omega \to \mathbb{R}$ the pressure distribution, then the Navier-Stokes equations for an incompressible fluid of density $\rho \in \mathbb{R}^+$, with free surface $\eta : [0, t_f] \times \Omega' \to \mathbb{R}$ over a bed (topography-bathymetry) of shape given by $b : \Omega' \to \mathbb{R}$, under body forces $\boldsymbol{f}_b : \Omega \to \mathbb{R}^3$, and neglecting difussion terms, are [8]

$$\begin{aligned} \nabla \cdot \boldsymbol{v} &= 0 \\ \frac{\partial}{\partial t}\boldsymbol{v} + \nabla \cdot \boldsymbol{v} \otimes \boldsymbol{v} &= -\frac{1}{\rho}\nabla p + \boldsymbol{f}_b \\ v_3|_{z=\eta} &= \frac{\partial \eta}{\partial t} + (\boldsymbol{v} \cdot \nabla)\eta \\ v_3|_{z=b} &= \frac{\partial b}{\partial t} + (\boldsymbol{v} \cdot \nabla)b \\ p|_{z=\eta} &= 0 \end{aligned} \tag{1}$$

Under the assumption that waves are sufficiently long, then vertical accelerations can be neglected, and by means of vertical integration between the bed b and free-surface elevation η, the Non-Linear Shallow Water equations read

$$\begin{aligned} (h)_t + (hu)_x + (hv)_y &= 0 \\ (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x + (huv)_y &= -ghb_x \\ (hu)_t + (huv)_x + (hv^2 + \frac{1}{2}gh^2)_y &= -ghb_y \end{aligned} \tag{2}$$

Following figure 1, $h(t, x, y) = \eta(t, x, y) - b(x, y)$ is the water depth and (u, v) the horizontal depth-averaged velocity components, defined as

$$(u, v) = \frac{1}{h}\int_b^\eta (v_1, v_2)dz$$

Using $\boldsymbol{q} = (h, hu, hv)^T$, $\boldsymbol{F}(\boldsymbol{q}) = (hu, \frac{1}{2}gh^2 + hu^2, huv)^T$, $\boldsymbol{G}(\boldsymbol{q}) = (hv, huv, \frac{1}{2}gh^2 + hv^2)^T$, $\boldsymbol{S}(\boldsymbol{q}) = (0, -\frac{1}{2}ghb_x, -\frac{1}{2}ghb_y)^T$, one can write in more compact notation

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial}{\partial x}\boldsymbol{F}(\boldsymbol{q}) + \frac{\partial}{\partial y}\boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{S}(\boldsymbol{q}) \tag{3}$$

The set of equations (3), restricting hydrodynamic variables to be defined in a domain $\Omega' \subset \mathbb{R}^2$, and with proper boundary conditions in $\partial\Omega'$ and initial conditions in Ω', form a system of non-linear hyperbolic partial differential equations. A semidiscrete formulation can be obtained by integrating over a control volume $\Omega_v \subset \Omega'$, which for simplicity can be thought of a rectangular cell of dimensions $\Delta x \times \Delta y$. Using the theorem of divergence one obtains

$$\frac{\partial \boldsymbol{q}_{ij}}{\partial t} + \frac{1}{\Delta x}(\boldsymbol{F}_{i+1/2,j} - \boldsymbol{F}_{i-1/2,j}) + \frac{1}{\Delta y}(\boldsymbol{G}_{i,j+1/2} - \boldsymbol{G}_{i,j-1/2}) = \boldsymbol{S}(\boldsymbol{q}_{ij}) \tag{4}$$

where $\boldsymbol{F}_{i\pm1/2,j} = \boldsymbol{F}(\boldsymbol{q}_{i\pm1/2,j})$ and $\boldsymbol{G}_{i,j\pm1/2} = \boldsymbol{G}(\boldsymbol{q}_{i,j\pm1/2})$, and $\boldsymbol{q}_{i\pm1/2,j}, \boldsymbol{q}_{i,j\pm/2}$ are the conserved variables evaluated at the interfaces of cells $(i \pm 1, j)$ and $(i, j \pm 1)$ respectively, and depend on a vecinity of cells at each respective edge.

The problem of obtaining interfaces values is solved by means of a one dimensional Riemann problem [2, 7, 1], which by a simple variable change is given by the initial value problem

$$\begin{aligned} \frac{\partial \boldsymbol{w}}{\partial t} + \boldsymbol{A}(\boldsymbol{w})\frac{\partial \boldsymbol{w}}{\partial x} &= 0 \\ \boldsymbol{w}(t=0,x) &= \begin{cases} \boldsymbol{w}_l & , \text{ if } x \leq 0. \\ \boldsymbol{w}_r & , \text{ if } x > 0 \end{cases} \end{aligned} \tag{5}$$

where $\boldsymbol{w} = (2c, u, v)^T$, $c = \sqrt{gh}$, $\boldsymbol{w}_L$ and $\boldsymbol{w}_R$ are the initial asociated states of for example $\boldsymbol{q}_{i,j}$ and $\boldsymbol{q}_{i+1,j}$ if one is interested on obtaining $\boldsymbol{q}_{i+1/2,j}$; and $\boldsymbol{A}(\boldsymbol{w})$ is the Jacobian matrix of the flux vector $\boldsymbol{F}$, for this coordinate change. Since solving this problem exactly implies performing approximations through iterative methods, for example, a linearized problem is solved instead by replacing $\boldsymbol{A}(\boldsymbol{w})$ with $\boldsymbol{A}(\widetilde{\boldsymbol{w}})$, where $\widetilde{\boldsymbol{w}} = (\boldsymbol{w}_L + \boldsymbol{w}_R)/2$, this aproach belongs to the family of Roe's linearized Riemann Solvers.

In particular, when dealing with cells which have edges in the boundary of the domain, boundary conditions are made to be satisfied by using *ghost-cells*. These are an actual extension of the domain, and values are specified there in such a way that once the Riemann problem is solved, values at the interface, which belong to the domain's boundary, automatically satisfy the boundary conditions. In other words, boundary conditions are satisfied using an inverse Riemann problem on which, given the values we want to have at an specific interface, we seek for what values to assign to each respective ghost cell.

The jacobian matrix is given by

$$\boldsymbol{A}(\boldsymbol{w}) = \begin{pmatrix} u & \sqrt{gh} & \sqrt{gh} \\ \sqrt{gh} & u & 0 \\ \sqrt{gh} & 0 & u \end{pmatrix} \tag{6}$$

and given its symmetry, if $h > 0$, it can have eigen values $\lambda_1 = u - c$, $\lambda_2 = u$ and $\lambda_3 = u + c$, and an asociated set of orthogonal eigenvectors given by a matrix $\boldsymbol{V}$, which can lead to the decoupled system of linear advection equations

$$\frac{\partial}{\partial t}(\boldsymbol{V}^{-1}\boldsymbol{w})_i + \lambda_i \frac{\partial}{\partial x}(\boldsymbol{V}^{-1}\boldsymbol{w})_i = 0 \tag{7}$$

for every i in $\{1, 2, 3\}$. Each λ_i defines a characteristic trajectory that separates two states, in such a way that at each side of the trajectory given by $x = \lambda_i t$ the value $(\boldsymbol{V}^T\boldsymbol{w})_i$ remains constant. This can be written more formally as

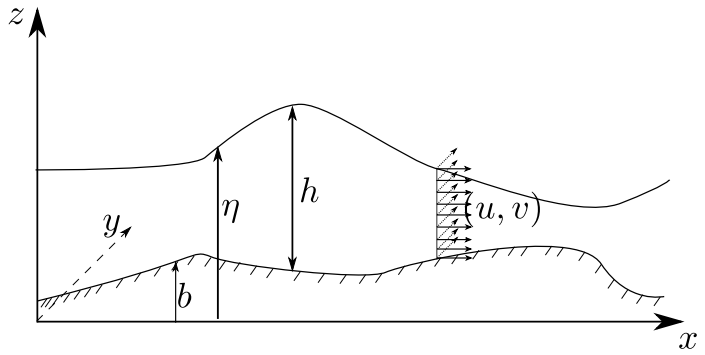


Figure 1: Schematic view of hydrodynamic variables defined for the Non-Linear Shallow Water Equations.

$$\begin{aligned} \boldsymbol{w}(x^-,t) &= \boldsymbol{w}_L + \sum_{\frac{x}{t}>\lambda_i} (\boldsymbol{l}_i^T(\boldsymbol{w}_R - \boldsymbol{w}_L))\boldsymbol{r}_i \\ \boldsymbol{w}(x^+,t) &= \boldsymbol{w}_R - \sum_{\frac{x}{t}<\lambda_i} (\boldsymbol{l}_i^T(\boldsymbol{w}_R - \boldsymbol{w}_L))\boldsymbol{r}_i \end{aligned} \tag{8}$$

with $\boldsymbol{l}_i$ the rows of the matrix $\boldsymbol{V}^{-1}$ and $\boldsymbol{r}_i$ columns of $\boldsymbol{V}$. The notation x^- and x^+ means the vecinity to the left (right) of x and lead to equal values of $\boldsymbol{W}$ when $x/t \neq \lambda_i$. In particular we are interested on $x = 0$, as the value that will remain constant in the interface between cells, for the linearization in a time step $(t_0, t_0 + \Delta t)$.

Before computing interface values, values $\boldsymbol{w}_L, \boldsymbol{w}_R$ are reconstructed using an approximation of the exact solution to the non homogeneous Riemann problem that includes b on its source terms, this reconstruction allows to preserve steady states at rest without adding spurious oscillations.

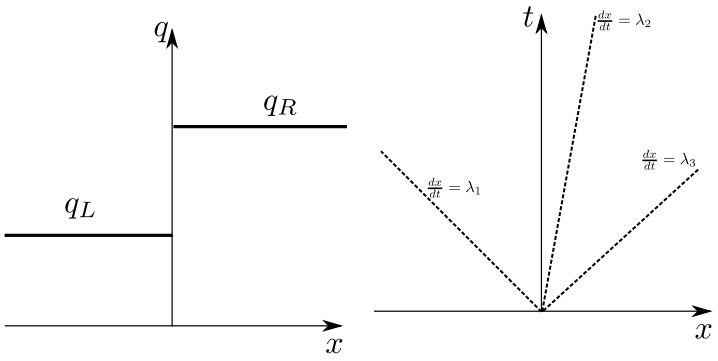
In order Finally, the model $SurfWB{-}UC$ introduces a coordinate transformation $(x, y) \to (\xi, \eta)$ that allows one to use more flexible grids that can be better adapted to the domain geometry, as shown in figure 3, for example. In fact, equations (2), after adimensionalizing, are equivalent to the system

$$\frac{\partial \boldsymbol{q}}{\partial t} + J\frac{\partial \boldsymbol{F}}{\partial \xi} + J\frac{\partial \boldsymbol{G}}{\partial \eta} = \boldsymbol{S}(\boldsymbol{q}) \tag{9}$$

$$\boldsymbol{F} = \frac{1}{J}\begin{pmatrix} hU^1 \\ u(hU^1) + \frac{1}{2Fr^2}h^2\xi_x \\ v(hU^1) + \frac{1}{2Fr^2}h^2\xi_y \end{pmatrix} \quad \boldsymbol{G} = \frac{1}{J}\begin{pmatrix} hU^2 \\ u(hU^2) + \frac{1}{2Fr^2}h^2\eta_x \\ v(hU^2) + \frac{1}{2Fr^2}h^2\eta_y \end{pmatrix}$$

$$\boldsymbol{S}(q) = \begin{pmatrix} 0 \\ -\frac{h}{Fr^2}(b_\xi\xi_x + b_\eta\eta_x) \\ -\frac{h}{Fr^2}(b_\xi\xi_y + b_\eta\eta_y) \end{pmatrix}$$

where $Fr = \frac{\mathcal{U}}{\sqrt{g\mathcal{H}}}$; $\mathcal{U} \in \mathbb{R}$ and $\mathcal{H} \in \mathbb{R}^+$ are arbitrary constants that represent characteristic



(a) Initial condition of the Riemann problem.

(b) Characteristics trajectories that separate regions in the $x-t$ plane.

Figure 2: Schematic representation of important properties of the linearized Riemann problem.

velocity and depth scales; $U^1 = u\xi_x + v\xi_y$, $U^2 = u\eta_x + v\eta_y$ and

$$J = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix}$$

and since this numerical scheme is integrated explicitly over time, the Courant-Friedrich-Lewy condition must be satisfied to enure numerical stability and convergence. The latter is acomplished by setting

$$\Delta t = Cr \frac{\min\{\Delta\xi, \Delta\eta\}}{\max\left(\max_{i,j}(U^1 + C\sqrt{\xi_x^2 + \xi_y^2}), \max_{i,j}(U^2 + C\sqrt{\eta_x^2 + \eta_y^2})\right)} \tag{10}$$

whith $\Delta\xi, \Delta\eta$ the mesh size in the curvilinear framework, and $Cr \in (0, 1)$

3 Algorithm structure overview

Given the description of the numerical model, a simple data flow representation of the algorithm of the SurfWB-UC original sequential code is shown in figure 4. It can be clearly identified that the most expensive part of the calculations in terms of floating point operations belongs to the stage on which the fluxes across each one of the cell's edges are calculated. The main idea of the parallelization method is

- Let the `master` core initialize parameters and initial conditions and broadcast them to the others.
- Implement a load-balanced domain decomposition
- Let each core receive a portion of the domain and to calculate fluxes individually as in the sequential code

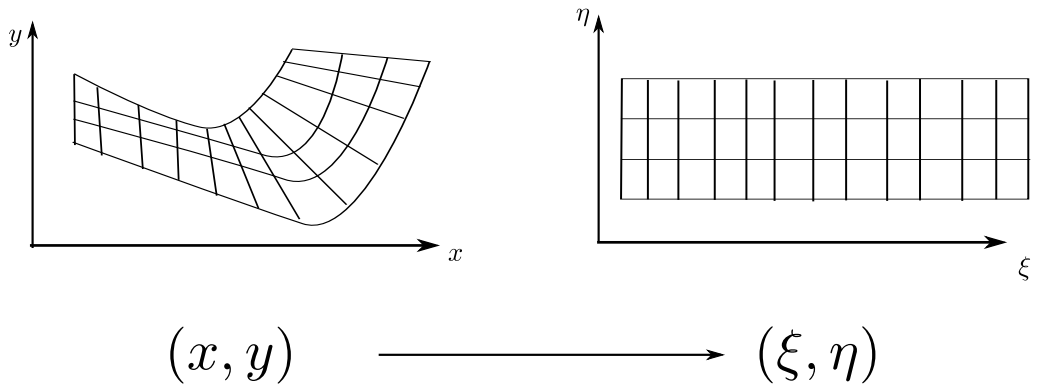


Figure 3: An example of curvilinear transformation, from the cartesian coordinate system (x, y) to the curvilinear system (ξ, η).

- Communicate processors using their respective sub-domains ghost cells, exchanging their values with their surroundings, without interrumpting the others.

3.1 Initialization

The initialization is then performed only by the core with rank `0` and then it broadcasts the read information to the rest of the cores, the initialization routines can be found in the file `$(SURF_PAR)/soure/init.f90`.

In order to broadcast efficiently, and taking advantage of the two-dimensional matrix structure of the input data, a `cartesian-topology` [4] was used to create a new communicator with the help of the functions `mpi_cart_create`, `mpi_dims_create`, so given the number of processors `nproc`, `MPI` can re-distribute the processors in a two dimensional grid and create a new communicator with the reordered processors. The information of what processors are neighbouring the current one, is found using the function `mpi_cart_shift`.

3.2 Domain decomposition

Given the cartesian topology, now it is needed to know which indices of the matrices correspond to each one. Suppose that we want to broadcast a vector of length n between p cores, one tentative approach is to assign $\lfloor \frac{n}{p} \rfloor$ elements to the first $p-1$ processors and $\lfloor \frac{n}{p} \rfloor + mod(n, p)$ to the last one. To clarify the notation, if n_i is the number of elements assigned to the $i-th$ processors, with

$i \in \{1, 2, .., P\}$, then following this idea

$$n_i = \begin{cases} \lfloor \frac{n}{p} \rfloor & \text{if } i \in \{1, 2, .., p-1\} \\ \lfloor \frac{n}{p} \rfloor + mod(n, p) & \text{if } i = p \end{cases} \tag{11}$$

clearly $\sum_{i=1}^{p} n_i = (p-1)\lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p} \rfloor mod(n, p) = n$, but if for example $n = 2p - 1$ then $\lfloor \frac{n}{p} \rfloor = \lfloor 2 - \frac{1}{p} \rfloor = 1$, $mod(n, p) = p - 1$ and

$$n_i = \begin{cases} 1 & \text{if } i \in \{1, 2, .., p-1\} \\ p & \text{if } i = p \end{cases} \tag{12}$$

which is a clear bad workload balance example.

Recall the definitions

$$T_{ave} = \frac{1}{p} \sum_{i=1}^{p} n_i \tag{13}$$

$$T_p = \max\{t_i, i \in \{1, 2, .., p\}\} \tag{14}$$

$$\beta = \frac{T_{ave}}{T_p} \tag{15}$$

we seek for an optimal distribution of the array's elements in terms of load balancing. The latter is clarified with the following proposition.

Proposition 1. *Given p processors and an array with n elements. Let $(l_i)_{i \in \{1,..,p\}}$ be any distribution of elements of this array that is going to be broadcasted to the $i-th$ processor. If t_i indicates the time of execution of the $i-th$ processor in an algorithm, and if there exists a constant $c > 0$ such that $t_i = c \times l_i$, for every $i \in \{1, .., p\}$, then the distribution of elements given by*

$$m_i = \begin{cases} \lfloor \frac{n}{p} \rfloor + 1 & \text{if } i \in \{1, .., d\} \text{ and } d > 0 \\ \lfloor \frac{n}{p} \rfloor & \text{if } i \in \{d+1, d+2, ..., p\} \text{ and } d > 0 \\ \frac{n}{p} & \text{if } d = 0 \end{cases} \tag{16}$$

with $d = mod(n, p)$, is the one with the best load balance.

Proof. It is easy to see that $\sum_{i=1}^{P} m_i = n$.

If d_i is such that $l_i = \lfloor \frac{n}{p} \rfloor + d_i$, it is clear that $\max\{d_i\} \geq 0$, and $\max\{d_i\} = 0$ implies $\lfloor \frac{n}{p} \rfloor = \frac{n}{p}$, on which case $l_i = \frac{n}{p} = m_i$.

If $\max\{d_i\} \geq 1$ then

$$\beta_l = \frac{\frac{c}{p} \sum_{i=1}^{p} l_i}{c \max\{l_i\}} = \frac{n/p}{\lfloor \frac{n}{p} \rfloor + \max\{d_i\}} \leq \frac{n/p}{\lfloor \frac{n}{p} \rfloor + 1} = \beta_m$$

and thus, (m_i) has the best load balance. □

Then, for a processor i, the respective elements of the array go from si to ei where

$s_i = \sum_{k=1}^{i-1} m_i + 1 = s$ to $e_i = \sum_{k=1}^{i} m_i$. Two dimensional arrays are broadcasted applying the same procedure to each cartesian direction, and using the coordinate system of the cartesian topology, the algorithm can be found in the file `$(SURF_PAR)/source/decomp_2d.f90` [2].

[2]Here SURF_PAR is the environment variable that directs to the directory where the SurfWB-UC parallel is installed

3.3 Communication

There are two moments when cores communicate. The first is to reduce the time step Δt to its minimum value among grids, so the CFL condition is respected in the whole domain and numerical stability and convergence can be ensured, this can be found in the file `$(SURF_PAR)/source/setdt.f90`. The second is before computing edge fluxes, when calculating boundary ghost cells values.

Say that the domain Ω' with $n_x \times n_y$ cells is splitted into two subdomains Ω'_1 and Ω'_2 of dimensions $n_{x1} \times n_{y1}$ and $n_{x2} \times n_{y2}$ such that $n_x = n_{x1} + n_{x2}+$, $n_y = n_{y1} + n_{y2}$, such that $\Omega' = \Omega'_1 \cup \Omega'_2$, as in figure 5. As can be seen there, the right boundary ghost cells of domain Ω'_1 are filled with a copy of the information from the green marked block of domain Ω'_2, and also are the left side boundary ghost cells of subdomain Ω'_2 with the information from the red marked block of subdomain Ω'_1. To acomplish with this goal, the safe method `mpi_sendrecv`, allows to communicate this information between processors without leading to deadlock, and using the information of the defined topology of processors. The file `$(SURF_PAR)/source/exchng2d.f90` contains the implementation of such routine.

Given all these aspects, the simple data flow of the parallel algorithm can be found in figure 5.

4 Performance analysis for a particular case

To study the performance of this code, a simple yet challenging test case was chosen. It is defined by the sudden breaking of a dam, which initially contained a volume of water with 10 meters height, over a rectangular flume which initially contained water with 5 meters height. The geometry of the dam and a snapshot of the simulation are shown in figure 6. In order to analyse the performance, the time interval of the simulation was constrained to $t \in (0, 3.0s)$, and for numerical stability a value for $C_r = 0.95$ was picked in all simulations [3].

4.1 Hardware specifications

The simulations were run in the cluster machine installed in the dependencies of the Escuela de Ingeniería at the Pontificia Universidad Católica de Chile. Some technical specifications [4] are

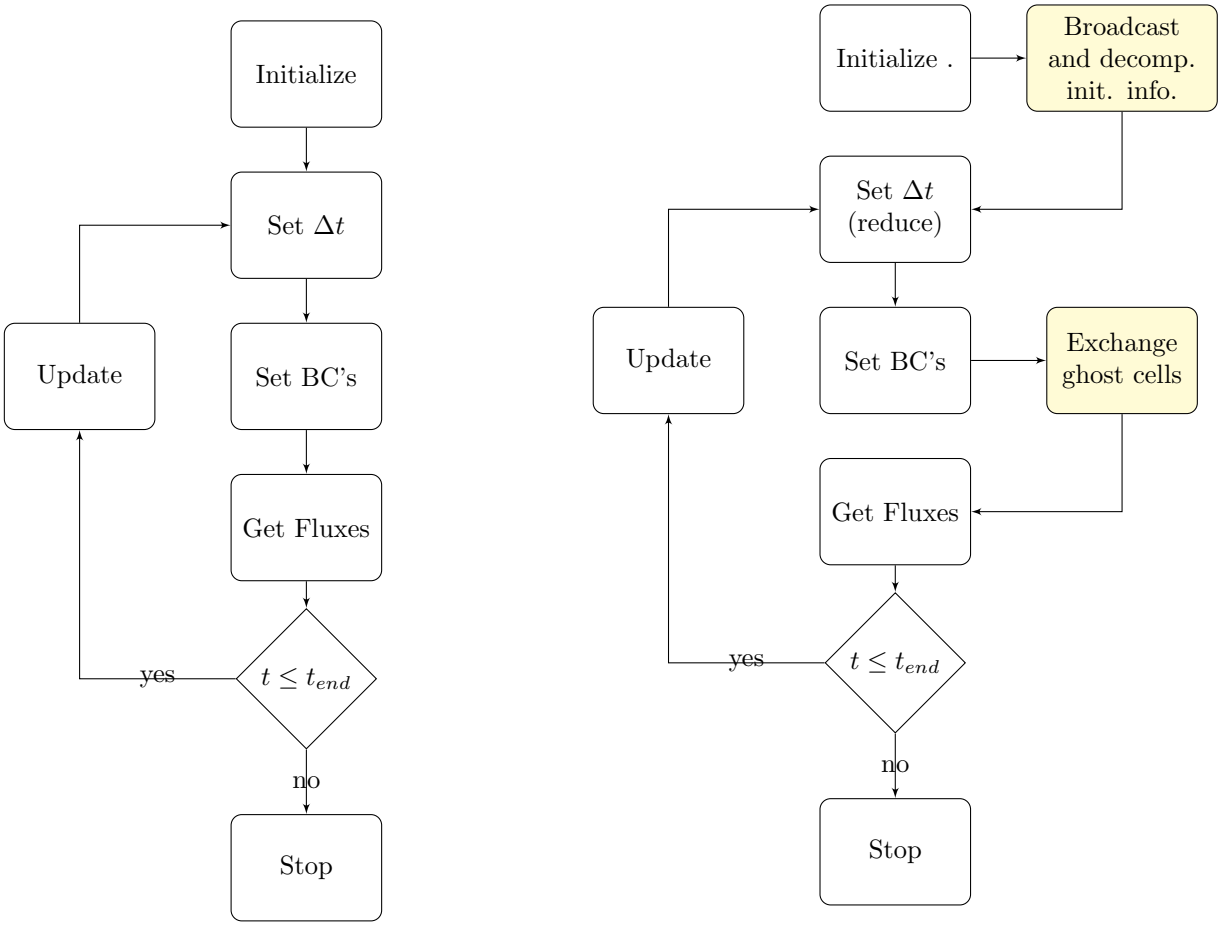
```
Total number of computing nodes = 21
Maximum number of simultaneous processes = 672
Total Memory (RAM) = 672 GB

Computing nodes:
Processor: Intel E5-2470, 8 Cores, 2.3 Ghz
32Gb RAM
Dual port network card, 10 Gb/s
Two 50Gb SSD
```

Also, it is important to mention that the queue allows each user to use at most 96 simultaneous cores per job, and so it is the maximum number considered for the following tests.

[3] The reader is refered to Reference [6, 5] for an extensive validation of this model

[4] More technical specifications available at http://geoserver.ing.puc.cl/wiki



(a) Sequential

(b) Parallel

Figure 4: Schematic data flow for the algorithms of the sequential and parallelized versions of the SurfWB-UC code

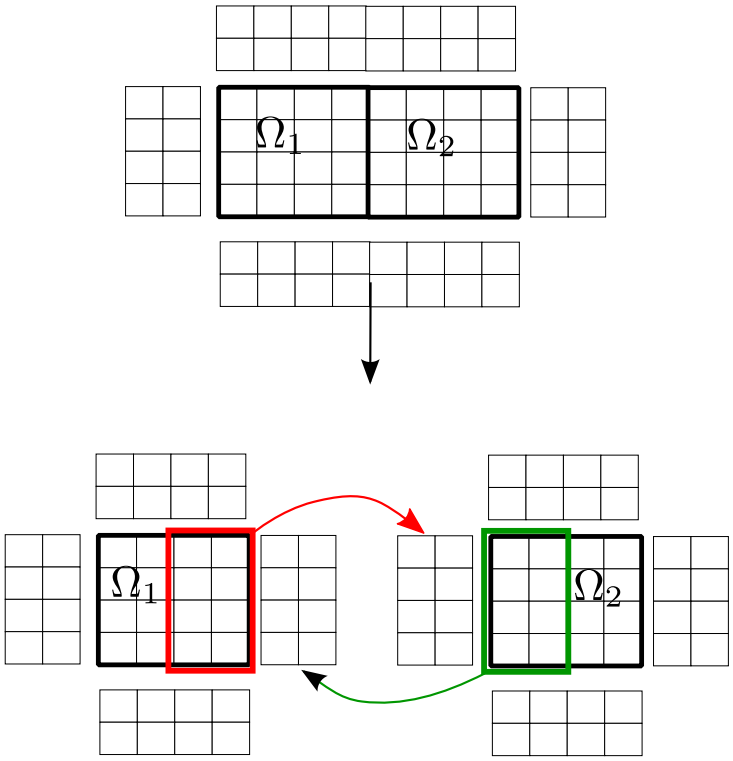


Figure 5: Boundaries ghost cells exchange for a simple case of domain decomposition. The red marked block of cells inside subdomain Ω_1 are copied in the left rows of ghost cells of subdomain Ω_2
, while the green marked block of cells from domain Ω_2 are copied to the right boundary ghost cells of Ω_1.

4.2 Recall of definitions

If an algorithm is executed with p processors, then $T_p = \max\{t_i, i \in \{1, 2, ..., p\}\}$ is the time of execution of the code using p processors and the speed up is defined as

$$S_p = \frac{T_1}{T_p} \tag{17}$$

and the related efficiency

$$E_p = \frac{S_p}{p} \tag{18}$$

Also, the experimental Karp-Flatt metric for measuring the serial fraction of the code is

$$exp_f = \frac{\frac{1}{S_p} - \frac{1}{P}}{1 - \frac{1}{p}} \tag{19}$$

In adition, an algorithm is called scalable if there exists a positive value ϵ such that for any data of size N one can find a number of processors P_N such that $E_{P_N}(N) \geq \epsilon$.

Also, the lesser rigurous definitions of scalability are:

- Strong scalability: If the execution time T_p decreases in inverse proportion to p, for every fixed data size N
- Weak scalability: If there is possible to increase N and p in proportion and keep N/p constant, in such a way that T_p remains constant.

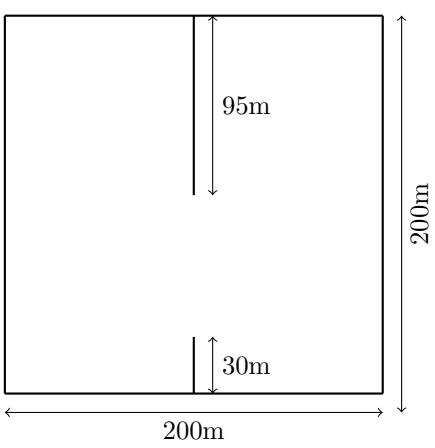
4.3 Results

Figure 7 shows in a $log - log$ plot the execution time as a function of the number of processors for different mesh discretization of the dam, varying from 100×100 up to 700×700 volumes. It can be seen that for a fixed p, t_p is estrictly increasing with respect to N, wich is natural. Also, for a fixed N, t_p decreases in constant logaritmic proportion as p increases, which can suggest strong scalability of the code. Even though, this is not so clear for the case with a grid of 200×200 control volumes, where the execution time seems more scattered and increasing with $p > 10^1$, which by the *rule of thumb*, suggests that it is not convenient to increase the number of processors for that mesh.

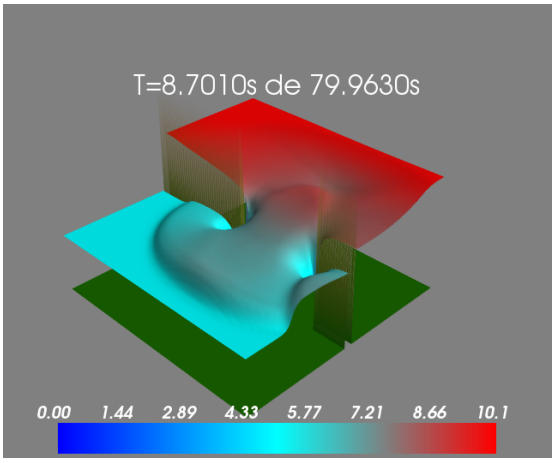
The Speed-Up is shown in figure 8, along with fitted curves of the form $S_p = \alpha P^{\beta}$. First of all, it is possible to see that an speed-up greater than 10 is possible to be obtained which is a good result by itself. It is possible to notice that even though, when increasing the number of processors for a fixed data amount, an increasing performance is observed for all cases except the grid of size 200×200, but when fixing the number p of cores, the speed up seems to obtain its maximum value with the mesh of 400×400 volumes, and decreasing after that. The latter can be confirmed with the exponent β of the fitted curves, which show the same behavour as explained.

In terms of the efficiency of the code, figure 9 suggests that there may be a lower bound for the efficiency as N grows, and thus apparently the algorithm would be rigorously scaled.

Finally, figure 10 shows that for a fixed amount of data the experimental serial fraction of the code remains approximately constant, and thus, communication should not have a great influence



(a) Geometry of the dam



(b) One snapshot of the simulation

Figure 6: Properties of the dambreak study case

in the efficiency of the code but computation and loadbalancing. It is not so clear when looking at a fixed number of cores, if in general exp_f increases or decreases, but from the figure, it is possible that the serial fraction of the code may stay in a constant value around 10^{-1} and, given the Amdahl's law, the maximum possible speedup be of order 10.

5 Conclusions

An MPI parallelized version of the SurfWB-UC code has been developed using an approach of domain decomposition and communicating them using an specific property of this family of numerical methods, which refers to the ghost-cells that are used to satisfy the required boundary conditions. The domain decomposition has been done trying to maximize the load balance and communication betwen grids was done so no deadlock could happen. These were the major changes made to the code, and they refer to the initialization, decomposition and communication between grids.

A performance analysis was done for a particular yet significative case, which lead to different observations. On a first place, the execution time decreased in logaritmical proportion to the number of cores for a given data set, which lead to think on the code being strongly scalable, however, it seemed that for a constant number of processors, there would be an optimal number of data-size N, from where the speed-up would start to decrease. This also would contradict what was seen for the efficiency E_p, which apparently had a positive lower bound, indicating that the code is scalable in a rigurous manner. Finally, the more concrete conclusion was that for this given set of processors the experimental serial fraction of the code was around 10^-1, which suggests that the speed-up would be limited by an upper bound of order 10.

The discussion about the apparent contradictions on if the algorithm is scalable or not have to be subject to the limitations and characteristics of the machine where the code is now mostly executed, that is in the clúster machine at the dependencies of the School of Engineering at the Pontificia Universidad Católica de Chile. Subject to these restrictions, the conclusion that it is

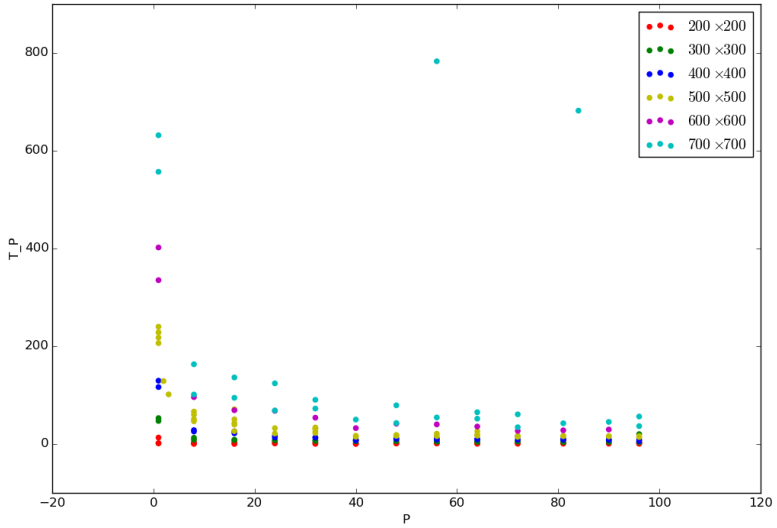


Figure 7: Time of execution with p processors

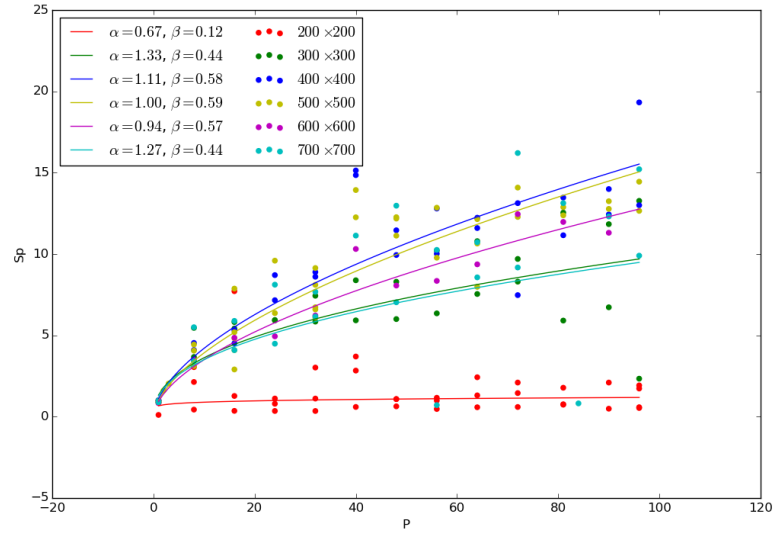


Figure 8: Speed-up S_p along with fitted curves of the form $S_p = \alpha P^{\beta}$

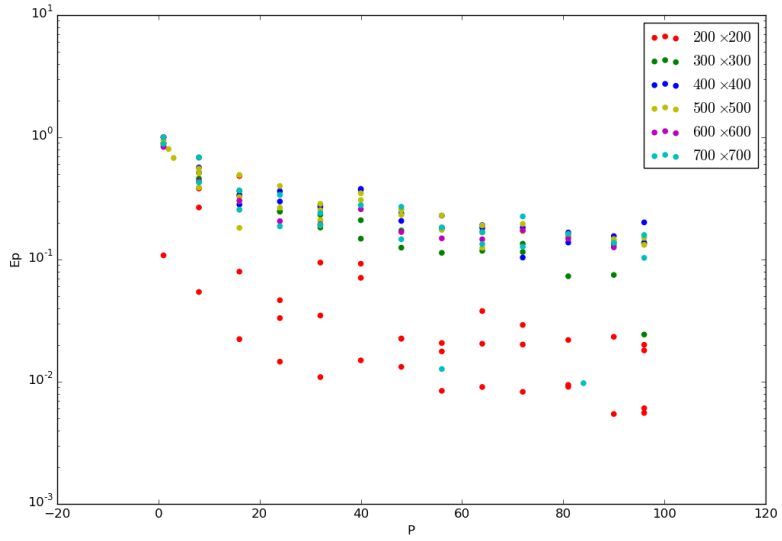


Figure 9: Efficiency E_p

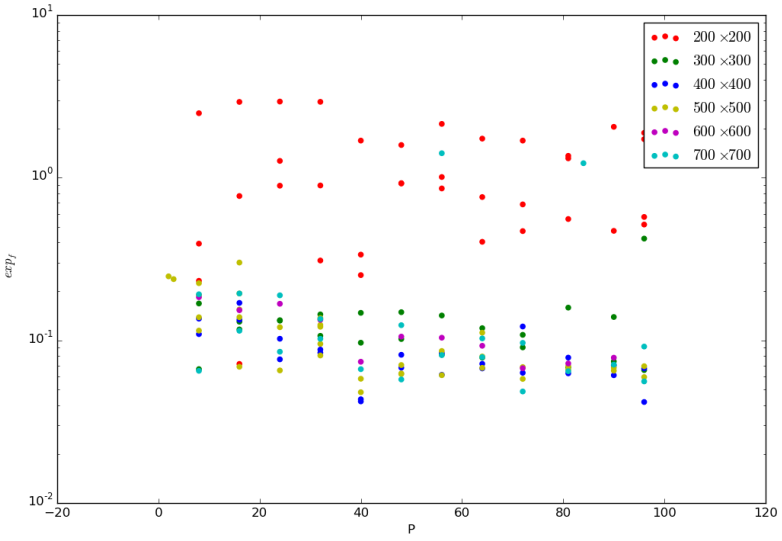


Figure 10: Experimental serial fraction of the code exp_f

possible to obtain an speed-up of order 10 is a positive contribution, and if true, may allow to decrease in at least one order of magnitude the execution time of the code.

Also, a set of tests for debugging is necessary, in order to capture exceptions that arized in the execution of the code that lead it to fail, apparently randomly, when varying the number of processors. This part is important for further use of the parallelized SurfWB-UC code.

Finally, additional extensions may allow to extend the approach followed herein to extend the cappabilities of the numerical model to the family of curvilinear overlapped grids [3], which may allow to communicate different coordinates transformations and hence obtain a better representation of the domain and the solution, and more flexibility with the mesh refinement.

References

[1] E. Audusse et al. "A fast and stable well-balance scheme with hysdrostatic reconstruction for shallow water". In: *SIAM Journal on Scientific Computing* 25.6 (2004), pp. 2050–2065.

[2] T. Gallouet, J. Gerard, and N. Seguin. "Some approximate Godunov schemes to compute shallow-water equations with topography". In: *Computers and fluids* 32 (2003), pp. 479–513.

[3] L. Ge, F. Sotiropoulos, and M.ASCE. "3D Unsteady RANS modeling of complex hydraulic engineering flows. I: Numerical Model". In: *Journal of Hydraulic Engineering* 131.9 (2005).

[4] W. Gropp. *Using MPI : portable parallel programming with the Message Passing Interface*. Cambridge, Mass. : MIT, 1999.

[5] M. Guerra. "Numerical and experimental modeling of extreme flood waves and inundation zones". MA thesis. Departamento de Ingeniería Hidráulica y Ambiental, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 2010.

[6] M. Guerra et al. "Modeling rapid flood propagation over natural terrains using a well-balanced scheme". In: *Journal of Hydraulic Engineering* 140 (2014).

[7] F. Marche et al. "Evaluation of well-balanced bore-capturing schemes for 2D wetting and drying processes". In: *International Journal for Numerical Methods in Fluids* 53 (2006), pp. 867–894.

[8] E. Toro. *Shock capturing methods for free surface shallow flows*. John Wiley, 2010.