Xavier ADRIAENS
Andrea ATTIPOE
Sébastien BRIALMONT
Gabriel DIGREGORIO
Julien DULAR
Romain HANUS

Master in Physical Engineering
MATH-0471 - Multiphysics Integrated Computational Project

Smoothed-Particle Hydrodynamics

Academic year 2016-2017

Contents

Acknowledgement

The authors would like to thank Mr. Louis GOFFIN without whom the following work could not have been accomplished, as it is based on his masters thesis *"Development of a didactic SPH model"*, published in 2013 [1].

The authors are also grateful to Pr. Christophe GEUZAINE and Dr. Romain BOMAN for the assistance and the help provided throughout the writing of both the code and the report, as well as for the many hours spent on this project.

Introduction

This integrated project consists in studying and implementing the Smoothed-Particle Hydrodynamics (SPH) numerical method in order to simulate the physical behaviour of fluids in different geometrical tridimensional configurations. The SPH method is different from many other computational methods like finite element method or finite volume method in the fact that it is a Lagrangian, meshfree method.

By contrast to a grid-based method which needs to generate a mesh to solve a problem, a meshfree method does not use any mesh to represent the problem. In general, in a meshfree method, the nodes are scattered over the domain in order to represent fluids and solids. Meshfree methods are sometimes preferred because they allow to get rid of drawbacks of grid-based methods: the generation of a mesh which a difficult task for the user, the discontinuities which are not well represented, a remeshing can sometimes become necessary but is not easy in 3D... The SPH method is a particle meshfree method which means that the geometry of the problem is represented by a set of particles and to each of them are associated some physical properties (density, pressure, velocity, ...). The boundaries are also defined by particles. Furthermore, this method can be qualified as a Lagrangian method because each particle moves with time and its properties are studied along its path.

The goal of this project is to study the basic principles of the SPH method and to implement an algorithm to solve fluid dynamics problems based on this method. To do so, many geometrical configurations and test cases are defined, each of them involving free surface flow in order to simulate the dynamics of the fluid in 3D and study the evolution of relevant variables.

In the first part of this project, the context of the method is briefly introduced. A short history is presented and some general applications in different fields involving the method as well as other different numerical methods for solving free surface flow problems are described. In the second part, the SPH method is introduced from a mathematical point of view. The kernel and the particle approximations and presented and a way to discretize the Navier-Stokes equations based on these approximations is proposed. The concept of artificial viscosity is introduced as well as the way in which the hydrostatic pressure and the speed of sound are initialized. After that, two integration schemes are introduced. This part ends by presenting several kernels which are implemented, the concept of smoothing length and how boundary conditions are treated. The third part consists in describing the implementation of the algorithm. First, neighbour search algorithms are presented. Parallelization strategies are then introduced: the distributed memory parallelization and the shared memory parallelization. The fourth part is devoted to the analysis of the implementation performance. The performance of the sequential code are first studied. Then, the performance of the parallel code are studied in a strong and weak scaling analyses. The last part consists in testing the code on a few test cases in order to validate the method numerically and physically. The different

test cases are designed to verify that specific physical variables evolve properly. In these test cases, different parameters are compared in order to find the ones that lead to good results with a reasonable computational cost.

Practically, the programming language that is used to implement the method is C++, the details to compile and run the code are given in appendix A.1. The results are visualized with ParaView, an open source multi-platform application for interactive, scientific visualization [36]. The post-processing of data as well as the different plots are realized with Matlab [37]. Figures and schematics are designed with IPE [38].

Part 1

Context of the method

1.1 Applications involving SPH method

The smoothed-particle hydrodynamics method was first introduced by Lucy in 1977 to model astronomical phenomena [2]. Since then, it has been extended to many different fields such as fluid dynamics and especially, the study of free surface flows [7], [8]. This is the goal of this project. In 1993, the SPH method was first applied to the dynamics of elastic-plastic solids by Libersky *et. al.* [3]. An overview of applications in other fields is given in the following paragraphs.

Gas dynamics A first simple SPH test is to use it in order to study linear wave phenomena [16]. Several tests have been performed and have shown that good agreement with theory is obtained when a sufficient number of particles is considered. In particular, SPH was applied to the problem of supersonic flow over a step and over a cylinder. In these problems the remaining issue is the boundary model. Another phenomenon involving gas dynamics which has also been studied with the SPH method is the development of a compressible Rayleigh-Taylor instability in an isothermal gas.

Binary stars and stellar collision The very first application for which the SPH method was used involved astrophysics. The problem of binary stars and polytropes was first studied using SPH by Gingold and Monaghan in 1977 using a small number of particles (less than 400) [11]. These results were in good agreement with stability theory. The collision between white dwarfs has also been studied by Benz *et. al.* in 1989 [20].

Fragmentation and cloud collisions Lattanzio *et. al.* applied the SPH method to the complex problem of interacting isothermal clouds [14]. Many authors studied these problems using first an identical range of interaction for all particles and then generalized the results by using spatially variable ranges of interaction. The collapse of an isothermal cloud which is initially uniformly rotating and non-axisymmetric has been studied by Monaghan and Gingold in 1981 [12].

Nearly incompressible flows In most hydraulic and aerodynamic applications, SPH was applied on compressible flow problems [16]. The method can however be extended to nearly incompressible flows using an artificial equation of state constructed so that the compressibility effects can be neglected (below the 1% level). A condition for making this assumption is that the Mach number of the flow should be smaller about 0.1. Then, once the new equation of state is constructed, the

SPH method can be applied and some free-surface problems (bursting dams, tidal bores, waterfall...) can be easily treated. The boundaries can be replaced by fixed particles which interact with water particles through molecular force models.

1.2 Methods for free surface flow simulation

As already mentioned, the meshfree methods has been developed quite recently compared to grid-based methods like finite difference method (FDM) or finite element method (FEM). The SPH method is one of the earliest meshfree methods which have been developed in the nineteenth century. In the following paragraphs, other methods that can be used to model free surface flows are presented.

In 1992, the diffuse element method (DEM) has been presented in order to generalize the finite element method (FEM) [4]. The success of the FEM is mainly due to the local character of approximation and the ability to deal with complex geometry. However this method presents two main drawbacks. First, the approximation provided by the FEM presents a limited regularity, sometimes, even if the solution is continuous, the derivatives are not necessarily continuous at elements boundaries which leads to difficulties of interpretation. Second, generating an adequate mesh is a difficult task, in particular in a complex tridimensional geometry. The idea of the DEM is to replace the FEM interpolation, valid on an element, by a local weighted least squares fitting, valid in a small neighbourhood of the considered point, based on some nodes close to that point.

Another meshfree method which can be used in various fields of application and particularly in fluid mechanics is the finite point method (FPM). It has been developed in 1996. This method consits in solving partial differential equations on scattered distribution of points. This technique consists in an approximation of local clouds of points using the so-called weighted least-squares methods. Then the discretization of the equations is performed by using a collocation method. Current efforts are still oriented to exploit the capabilities of the FPM to work in a parallel environment to solve problems where meshfree methods are useful: complex geometry, moving or deformable domain, etc.

In 1998, the meshless local Petrov-Galerkin (MLPG) method has been developed for solving a wide range of problems such as fracture mechanics problems, beam and plate bending problems or some fluid dynamics problems such as steady flows around cylinders, steady convection and diffusion flows and lid-driven cavity flows in a 2D box. More recently, this method has been extended to deal with free surface flow and especially nonlinear water wave problems [5]. This method is based on a local weak form over local sub-domains (circles for two-dimensional problems and spheres for three-dimensional problems).

Some methods combine the characteristics of meshfree methods with the generation of a particular mesh. For instance, the material point method (MPM) is a numerical technique used to simulate the behaviour of solids, liquids, gases and other continuum material. In this method, a body is described by a number of small Lagrangian elements called material points. The particularity of the MPM is the fact that these material points are surrounded by a mesh used only to compute gradient terms *e.g* the deformation gradient. Despite the fact that a background mesh is generated, this method belong to the meshfree methods. Actually, this method does not encounter the drawbacks of

mesh-based methods which makes it a powerful tool in computational mechanics.

A last method that is interesting to describe is the so-called particle finite element method (PFEM) [6]. The PFEM is a kind of extended FEM based on a Lagrangian formulation used to model problems involving interaction between fluids and structures. The general idea of the PFEM is to treat the mesh nodes in the fluid and solid domains as particles which can freely move and even separate from the main fluid domain. This method is not really a meshfree method because a finite element mesh connects the nodes defining the discretized domain where the governing equations are solved using a standard finite element method. The main advantage to use a Lagrangian formulation is that the convective terms disappear from the fluid equations. However, the difficulty lies in the problem of adequately (and efficiently) moving the mesh nodes. Furthermore, for large nodes movement, a remeshing procedure may be a frequent necessity.

The methods that have been described in this section are some of the most interesting methods for modelling fluid dynamics problems, especially for free surface flows. However, the numerical meshfree methods which have been developed since the 90's are plentiful. Among others, one can also cite the diffuse particle dynamics (DPD), the element-free Galerkin method (EFGM), the natural element method (NEM), the particle-in-cell method (PIC), the boundary node method (BNM), the method of fundamental solutions (MFS), etc.

Part 2

Mathematical and physical bases

In this second part, important principles of the SPH method are presented. The governing equations of fluid mechanics are manipulated and approximations are introduced to build a formulation that is compatible with a numerical implementation. Some aspects of the obtained equations are then discussed and several possibilities for solving them are proposed.

2.1 Derivation of the semi-discrete equations

The SPH formulation of the continuity and the momentum equations can be obtain using two major approximations. The first is called the kernel approximation. It states that the value of any field function at some point can be approached by an average of this function around the considered point. The second is often referred to as the particle approximation. It consists in describing the domain by a finite number of *particles*. Integrals over the domain or parts of it then becomes weighted sums over the particles.

Both approximations and their applications to mass and momentum equations are developed in the following sections.

2.1.1 Kernel approximation

Considering $\delta(\,\cdot\,)$ to be the Dirac function, any function f defined on a domain $\Omega \in \mathbb{R}^3$ admits the integral representation

$$f(\boldsymbol{x}) = \int_\Omega f(\boldsymbol{x'})\delta(\boldsymbol{x} - \boldsymbol{x'})d\boldsymbol{x'}. \tag{2.1}$$

The kernel approximation consists in substituting $\delta(\boldsymbol{x})$ by some smoothing function $W(\boldsymbol{x}, h)$, with h defined below. This smoothing function should satisfy a given set of properties. These properties are not strict. Important ones are listed below (adapted from [9]):

1. **Unity.** The smoothing function *must* be normalized over its support, *i.e.*,

$$\int_\Omega W(\boldsymbol{x} - \boldsymbol{x'}, h)d\boldsymbol{x'} = 1. \tag{2.2}$$

2. **Compact support.** The smoothing function *should* have a compact support, *i.e.*,

$$W(\boldsymbol{x} - \boldsymbol{x'}, h) = 0, \quad \forall \, \|\boldsymbol{x} - \boldsymbol{x'}\| > \kappa h, \tag{2.3}$$

where κh defines the size of the domain of dependence of point $\boldsymbol{x}$. The parameter h is called the smoothing length and the scaling factor κ is kernel dependent. Note that the product κh is sometimes also referred to as the smoothing length.

3. **Positivity.** The smoothing function *should not* be negative, *i.e.*, $W(\boldsymbol{x}-\boldsymbol{x'},h) \geq 0,\ \forall \boldsymbol{x'}$.

4. **Decay.** The smoothing function *should* be monotonically decreasing when $\|\boldsymbol{x}-\boldsymbol{x'}\|$ increases.

5. **Delta function.** The smoothing function *should* converge toward the Dirac function when h tends to zero, *i.e.*, $\lim_{h\to 0} W(\boldsymbol{x},h) = \delta(\boldsymbol{x})$.

6. **Symmetry.** The smoothing function *should* be an even function, *i.e.*,

$$W(\boldsymbol{x}-\boldsymbol{x'},h) = W(\boldsymbol{x'}-\boldsymbol{x},h), \quad \forall \boldsymbol{x},\boldsymbol{x'}. \tag{2.4}$$

7. **Smoothness.** The smoothing function *should* be sufficiently smooth.

The kernel approximation operator $\langle\, \cdot \,\rangle$ is then defined,

$$\langle f(\boldsymbol{x})\rangle = \int_\Omega f(\boldsymbol{x'})W(\boldsymbol{x}-\boldsymbol{x'},h)d\boldsymbol{x'}, \tag{2.5}$$

and it can be demonstrated that $\langle f(\boldsymbol{x})\rangle = f(\boldsymbol{x}) + \mathcal{O}(h^2)$ [9]. Moreover, it can be shown that the kernel operator applied to the divergence of any vector valued function $\boldsymbol{f}$ defined on $\Omega \in \mathbb{R}^3$ writes

$$\langle \boldsymbol{\nabla}\cdot\boldsymbol{f}(\boldsymbol{x})\rangle = \int_{\delta\Omega} W(\boldsymbol{x}-\boldsymbol{x'},h)\boldsymbol{f}(\boldsymbol{x'})\cdot\boldsymbol{n}\, dS - \int_\Omega \boldsymbol{f}(\boldsymbol{x'})\cdot\boldsymbol{\nabla}_{\boldsymbol{x'}}W(\boldsymbol{x}-\boldsymbol{x'},h)d\boldsymbol{x'} \tag{2.6}$$

where $\delta\Omega$ is the boundary of Ω and $\boldsymbol{n}$ is the unit outwards normal to $\delta\Omega$ and that the kernel approximation of any scalar valued function is given by

$$\langle \boldsymbol{\nabla} f(\boldsymbol{x})\rangle = \int_{\delta\Omega} f(\boldsymbol{x'})W(\boldsymbol{x}-\boldsymbol{x'},h)\,\boldsymbol{n}\, dS - \int_\Omega f(\boldsymbol{x'})\boldsymbol{\nabla}_{\boldsymbol{x'}}W(\boldsymbol{x}-\boldsymbol{x'},h)d\boldsymbol{x'}. \tag{2.7}$$

Provided that $\boldsymbol{x}$ is such that the compact support of $W(\boldsymbol{x}-\boldsymbol{x'},h)$ stands within the interior of the domain Ω, Eq. (2.6) and (2.7) simplify as

$$\langle \boldsymbol{\nabla}\cdot\boldsymbol{f}(\boldsymbol{x})\rangle = -\int_\Omega \boldsymbol{f}(\boldsymbol{x'})\cdot\boldsymbol{\nabla}_{\boldsymbol{x'}}W(\boldsymbol{x}-\boldsymbol{x'},h)d\boldsymbol{x'} \tag{2.8}$$

and

$$\langle \boldsymbol{\nabla} f(\boldsymbol{x})\rangle = -\int_\Omega f(\boldsymbol{x'})\boldsymbol{\nabla}_{\boldsymbol{x'}}W(\boldsymbol{x}-\boldsymbol{x'},h)d\boldsymbol{x'}. \tag{2.9}$$

2.1.2 Particle approximation

Replacing the volume integral that appears in the kernel operator by a finite sum over new objects called *particles*, Eq. (2.5) can been approximated by

$$\langle f(\boldsymbol{x}_i)\rangle = \int_\Omega f(\boldsymbol{x'})W(\boldsymbol{x}_i - \boldsymbol{x'}, h)d\boldsymbol{x'}, \tag{2.10}$$

$$\approx \sum_{j=1}^{N_i} f(\boldsymbol{x}_j)W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)\Delta V_j \tag{2.11}$$

$$\approx \sum_{j=1}^{N_i} f(\boldsymbol{x}_j)W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)\frac{m_j}{\rho_j}, \tag{2.12}$$

where $\boldsymbol{x}_i$ is the position of particle i where f is evaluated, $\boldsymbol{x}_j$ is the position of a particle in the vicinity of particle i (referred to as a neighbor of particle i), such that $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| < \kappa h$, N_i is the number of neighbors of particles i and ΔV_j, ρ_j and m_j are respectively, the volume, the density and the mass associated to particle j.

Similarly, the divergence of any function $\boldsymbol{f}$ (Eq. (2.8)) becomes

$$\langle \boldsymbol{\nabla}\cdot\boldsymbol{f}(\boldsymbol{x}_i)\rangle = -\int_\Omega \boldsymbol{f}(\boldsymbol{x'})\cdot\boldsymbol{\nabla}_{\boldsymbol{x'}}W(\boldsymbol{x} - \boldsymbol{x'}, h)d\boldsymbol{x'} \tag{2.13}$$

$$\approx -\sum_{j=1}^{N_i} \boldsymbol{f}(\boldsymbol{x}_j)\cdot\boldsymbol{\nabla}_{\boldsymbol{x}_j}W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)\Delta V_j \tag{2.14}$$

$$\approx -\sum_{j=1}^{N_i} \boldsymbol{f}(\boldsymbol{x}_j)\cdot\boldsymbol{\nabla}_{\boldsymbol{x}_j}W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)\frac{m_j}{\rho_j}, \tag{2.15}$$

and the gradient of any scalar valued function f is approached by

$$\langle \boldsymbol{\nabla} f(\boldsymbol{x}_i)\rangle \approx -\sum_{j=1}^{N_i} f(\boldsymbol{x}_j)\boldsymbol{\nabla}_{\boldsymbol{x}_j}W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)\frac{m_j}{\rho_j}. \tag{2.16}$$

As explained in [33], it can be chosen that the mass m_j of each particle is constant and is thus given by

$$m_j = \frac{\Delta V_j(t_0)}{\rho_j(t_0)}, \tag{2.17}$$

where t_0 is the initial time. In this project, it has been chosen to discretize the domain into cubic particles of size s which means that the masses are computed through

$$m_j = \frac{s^3}{\rho_j(t_0)}. \tag{2.18}$$

2.1.3 Continuity and momentum equations

Denoting by ρ [kg/m^3] the density, $\boldsymbol{v}$ [m/s] the velocity, $\boldsymbol{\sigma}$ [N/m^2] the stress tensor, $\boldsymbol{g}$ [m/s^2] the body accelerations and $\frac{D\cdot}{Dt}$ the particular derivative operator, the Lagrangian formulation of conservation

of mass and momentum balance is given by

$$\begin{cases} \dfrac{D\rho}{Dt} = -\rho \boldsymbol{\nabla} \cdot \boldsymbol{v}, \\ \dfrac{D\boldsymbol{v}}{Dt} = \dfrac{1}{\rho} \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \boldsymbol{g}, \end{cases} \tag{2.19}$$

which can be written in the following form

$$\begin{cases} \dfrac{D\rho}{Dt} = \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho - \boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}), \\ \dfrac{D\boldsymbol{v}}{Dt} = \boldsymbol{\nabla} \cdot \left(\dfrac{\boldsymbol{\sigma}}{\rho} \right) + \dfrac{\boldsymbol{\sigma}}{\rho^2} \cdot \boldsymbol{\nabla} \rho + \boldsymbol{g}, \end{cases} \tag{2.20}$$

using the identities

$$\begin{cases} \boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}) = \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho + \rho \boldsymbol{\nabla} \cdot \boldsymbol{v}, \\ \boldsymbol{\nabla} \cdot \left(\dfrac{\boldsymbol{\sigma}}{\rho} \right) = \dfrac{1}{\rho} \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} - \dfrac{\boldsymbol{\sigma}}{\rho^2} \cdot \boldsymbol{\nabla} \rho. \end{cases} \tag{2.21}$$

Divergences and gradients can then be replaced by their kernel approximations (Eq. (2.15) and (2.16)),

$$\boldsymbol{\nabla} \rho \rightarrow \qquad \langle \boldsymbol{\nabla} \rho_i \rangle \approx - \sum_{j=1}^{N_i} \rho_j \boldsymbol{\nabla}_{\boldsymbol{x}_j} W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) \frac{m_j}{\rho_j}, \tag{2.22}$$

$$\boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}) \rightarrow \qquad \langle \boldsymbol{\nabla} \cdot (\rho_i \boldsymbol{v}_i) \rangle \approx - \sum_{j=1}^{N_i} \rho_j \boldsymbol{v}_j \cdot \boldsymbol{\nabla}_{\boldsymbol{x}_j} W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) \frac{m_j}{\rho_j}, \tag{2.23}$$

$$\boldsymbol{\nabla} \cdot \left(\frac{\boldsymbol{\sigma}}{\rho} \right) \rightarrow \qquad \left\langle \boldsymbol{\nabla} \cdot \left(\frac{\boldsymbol{\sigma}_i}{\rho_i} \right) \right\rangle \approx - \sum_{j=1}^{N_i} \frac{\boldsymbol{\sigma}_j}{\rho_j} \cdot \boldsymbol{\nabla}_{\boldsymbol{x}_j} W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) \frac{m_j}{\rho_j}, \tag{2.24}$$

where simplified notations $g_k \equiv g(\boldsymbol{x}_k)$, for any vector or scalar valued function g, have been used. Using the symmetry property of the smoothing functions,

$$\boldsymbol{\nabla}_{\boldsymbol{x}_j} W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) = -\boldsymbol{\nabla}_{\boldsymbol{x}_i} W(\boldsymbol{x}_i - \boldsymbol{x}_j, h), \tag{2.25}$$

and inserting Eq. (2.22) to (2.24) into the continuity and momentum Eqs. (2.20) finally yields

$$\begin{cases} \dfrac{D\rho_i}{dt} = \displaystyle\sum_{j=1}^{N} m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \boldsymbol{\nabla}_i W(\boldsymbol{x}_i - \boldsymbol{x}_j, h), \\ \dfrac{D\boldsymbol{v}_i}{Dt} = \displaystyle\sum_{j=1}^{N} m_j \left(\frac{\boldsymbol{\sigma}_i}{\rho_i^2} + \frac{\boldsymbol{\sigma}_j}{\rho_j^2} \right) \cdot \boldsymbol{\nabla}_i W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) + \boldsymbol{g}_i, \end{cases} \tag{2.26}$$

or

$$\begin{cases} \dfrac{D\rho_i}{dt} = \displaystyle\sum_{j=1}^{N} m_j \boldsymbol{v}_{ij} \cdot \boldsymbol{\nabla}_i W_{ij}, \\ \dfrac{D\boldsymbol{v}_i}{Dt} = \displaystyle\sum_{j=1}^{N} m_j \left(\frac{\boldsymbol{\sigma}_i}{\rho_i^2} + \frac{\boldsymbol{\sigma}_j}{\rho_j^2} \right) \cdot \boldsymbol{\nabla}_i W_{ij} + \boldsymbol{g}_i, \end{cases} \tag{2.27}$$

using the shortcuts $\boldsymbol{v}_i - \boldsymbol{v}_j = \boldsymbol{v}_{ij}$ and $W(\boldsymbol{x}_i - \boldsymbol{x}_j, h) = W_{ij}$.

2.1.4 Position update

The continuity and momentum equations (2.27) describe the time evolution of the density and velocity of particle i. The position can be deduced from the velocity by integration,

$$\frac{D\boldsymbol{x}_i}{Dt} = \boldsymbol{v}_i. \tag{2.28}$$

The resulting formulation is the classical SPH formulation. Monaghan has introduced a variant of this method, known as the XSPH method [16]. It consists in updating the position with an averaged velocity. The update equation for the position is then defined as

$$\frac{D\boldsymbol{x}_i}{Dt} = \boldsymbol{v}_i + \varepsilon \sum_{j=1}^{N} \frac{m_j}{\rho_j}(\boldsymbol{v}_j - \boldsymbol{v}_i) W(\boldsymbol{x}_i - \boldsymbol{x}_j, h), \tag{2.29}$$

where ε [-] is a constant parameter that tunes the importance of the XSPH correction. It is often fixed to 0.25. This correction does not introduce any dissipation. However, it introduces a higher dispersion of the motion because of the average. The motivation for this extension is to avoid interpenetration of particles in compressible flows. According to [21] and [22], it also provides more ordered particle distributions for incompressible flows. The influence of this correction is analyzed in part 5.

2.2 Viscous effects

The stress tensor $\boldsymbol{\sigma}$ in the momentum equation (Eq. 2.27) can be decomposed into a pressure part (p) and a viscous part ($\boldsymbol{\tau}$) as

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \boldsymbol{\tau}. \tag{2.30}$$

This viscous stress tensor $\boldsymbol{\tau}$ can take different forms according to the considered constitutive law. However, inserting Eq. (2.30) into the momentum equation leads to a formulation which does not conserve linear and angular momentum [18].

Based on these considerations and in order to avoid numerical instabilities due to oscillations in the velocity vector field, an artificial viscous stress can be introduced instead of the physical viscous stress $\boldsymbol{\tau}$. This yields the following form for the semi-discrete equations:

$$\begin{cases} \dfrac{D\rho_i}{dt} = \displaystyle\sum_{j=1}^{N} m_j \boldsymbol{v}_{ij} \cdot \boldsymbol{\nabla}_i W_{ij}, \\ \dfrac{D\boldsymbol{v}_i}{Dt} = -\displaystyle\sum_{j=1}^{N} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij} \right) \boldsymbol{\nabla}_i W_{ij} + \boldsymbol{g}_i, \end{cases} \tag{2.31}$$

where Π_{ij} is the artificial viscosity term between particle i and j.

Artificial viscosity equation An artificial viscosity model was proposed by Monaghan [13], [16]. It states as follows

$$\Pi_{ij} = \begin{cases} \dfrac{-\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\bar{\rho}_{ij}} & \boldsymbol{v}_{ij} \cdot \boldsymbol{x}_{ij} < 0, \\ 0 & \boldsymbol{v}_{ij} \cdot \boldsymbol{x}_{ij} > 0, \end{cases} \tag{2.32}$$

where

- $\bar{c}_{ij}$ is the mean speed of sound of particle i and j,
- $\bar{\rho}_{ij}$ is the mean density of particle i and j,
- μ_{ij} expresses the relative speed dependence of Π_{ij} and is defined by

$$\mu_{ij} = \frac{h\boldsymbol{v}_{ij} \cdot \boldsymbol{x}_{ij}}{\|\boldsymbol{v}_{ij}\|^2 + \tilde{\varepsilon}h^2}, \tag{2.33}$$

- α, β and $\tilde{\varepsilon}$ are numerical parameters to be tuned.

This artificial viscosity vanishes when $\boldsymbol{v}_{ij} \cdot \boldsymbol{x}_{ij} > 0$, *i.e.*, when particles i and j move away from each other.

The linear term with respect to the velocity contains a factor α [-]. It aims to produce shear and bulk viscosity. It conserves linear momentum and angular momentum if $W(\boldsymbol{x}_{ij}, h) = W(\|\boldsymbol{x}_{ij}\|, h)$ (isotropic kernel) [28]. It can also be shown that for the continuous limit and with $\tilde{\varepsilon} \to 0$, it mimics Navier-Stokes both first and second bulk viscosity coefficient [28]. The value of α is case-dependent but Monaghan advises to choose $\alpha \approx 0.5$ [16].

The second term, proportional to β [-], mimics a normal bulk viscosity and a Von Neumann-Richtmyer bulk viscosity which are large for shocks but very small otherwise [19]. It is useful at high Mach numbers to prevent interpenetration of two particles. If not dealing with high speed flow or shocks, β could be fixed to zero.

The quantity $\tilde{\varepsilon}$ [-] is introduced to prevent a singularity when $\|\boldsymbol{r}_{ij}\| \to 0$. It should be small enough to avoid severe smoothing of the viscous term in high density region. It has been fixed $\tilde{\varepsilon} = 0.01$.

2.3 Pressure effects

A pressure is attached to each particle. There are several ways to compute this pressure, depending on the considered SPH variant.

Compressible fluids When writing the momentum and the continuity equations (Eqs. 2.19), one implicitly considers that the fluid is compressible as the time derivative of the density of each particle does not vanish *a priori* and is not corrected *a posteriori*.

For compressible flows, the pressure field can be linked to the density field using a state equation. In the scope of this project, weakly compressible fluids have been considered, leading to a variant of the method called weakly compressible smooth particle hydrodynamics (WCSPH). It can be shown that the state equation related to these fluids is given by

$$p = \frac{c_0^2 \rho_0}{\gamma} \left(\left(\frac{\rho}{\rho_0} \right)^\gamma - 1 \right) = B \left(\left(\frac{\rho}{\rho_0} \right)^\gamma - 1 \right). \tag{2.34}$$

where ρ_0 and c_0 are respectively the density and the speed of sound under no external pressure, $B = c_0^2 \rho_0 / \gamma$ is the bulk compressibility and $\gamma = 7$ should be taken [17].

Speed of sound Using this state equation, the speed of sound c, which is defined as

$$c^2 = \left(\frac{\partial p}{\partial \rho}\right)_S \quad \text{(at constant entropy)}, \tag{2.35}$$

is given by

$$c = c_0 \sqrt{\left(\frac{\rho}{\rho_0}\right)^{\gamma-1}}. \tag{2.36}$$

The initial speed of sound c_0 should be taken such that the information is able to propagate fast enough to reproduce the physics of the problem. It also has to guarantee that the variation of density which are $\mathcal{O}\left(M^2\right)$ with $M = u_{max}/c$ the Mach number, are small, *i.e* smaller than 1%. Nevertheless, c_0 can not be taken to be the physical speed of sound because numerical stability would then require a very small time step.

In the case of shallow water waves, the theoretical speed of waves is given by $c = \sqrt{gH}$ where H is the depth of water at rest, and the numerical speed of sound c_0 is often taken to be $c_0 = 10\sqrt{gH}$, a value that fulfills the above mentioned restriction. More details are given in [32] for example.

Hydrostatic pressure setting As the density is the only independent variable from which the pressure can be deduced in the case of a compressible fluid, setting an (initial) hydrostatic pressure field

$$p = \rho_0 g h, \tag{2.37}$$

h being the depth from the free surface, must be done through the setting of an appropriate density field. This density field is obtained by injecting Eq. (2.37) into Eq. (2.34) and solving for ρ, which yields

$$\rho = \rho_0 \left(1 + \frac{1}{B}\rho_0 g h\right)^{\frac{1}{\gamma}}. \tag{2.38}$$

Incompressible fluids Incompressible fluids can be modelled using a predictor-corrector scheme where the corrector is used to imposed the variation of density to zero through the setting of an appropriate pressure field [31]. This leads to a variant called divergence-free smooth particle hydrodynamics (DFSPH) which has not been considered in this project.

2.4 Integration schemes

The time discretization of Eqs. (2.31), (2.28) and (2.29) can be performed using an explicit Euler integration scheme or a two-step Runge-Kutta integration scheme for example. Other possibilities exist but are not considered in this project.

Gathering all the independent variables into one single 7-vector $\boldsymbol{V}$ defined as

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v} & \boldsymbol{x} & \rho \end{pmatrix}^{\mathrm{T}}, \tag{2.39}$$

and synthesizing the semi-discretized mass and momentum equations (Eqs. (2.31)) and the position update (Eq. (2.28) for classical SPH or (2.29) for XSPH) into a single operator $\boldsymbol{F}$, one can write the

governing equations of the SPH formulation in the following compact form

$$\frac{D\boldsymbol{V}}{Dt} = \boldsymbol{F}(\boldsymbol{V}(t)). \tag{2.40}$$

Moreover, in the following sections, the time step is denoted by k, the numerical approximation of $\boldsymbol{V}(t)$ is denoted by $\boldsymbol{V}_t$ and it is assumed that an approximation of $\boldsymbol{V}(t)$ is known for $t = t_0$.

2.4.1 Explicit Euler

Using the explicit Euler scheme, the time discretization is performed through

$$\boldsymbol{V}_{t_0+k} = \boldsymbol{V}_{t_0} + k\boldsymbol{F}(\boldsymbol{V}(t_0). \tag{2.41}$$

This first order scheme has the advantage of being very simple. However, a small time step k might be required for stability as will be seen in Part 5.

2.4.2 Two-step Runge-Kutta

The first step of the Runge-Kutta scheme considered here is to compute $\boldsymbol{F}(\boldsymbol{V}_{t_0})$, *i.e.*, the mass, momentum and position updates for the current configuration.

Then, an intermediate configuration is withdrawn from this latter derivatives using an Euler integration scheme with a time step equal to $k/2\theta$, with θ a parameter of the method whose value leads to different variants. Mathematically, this intermediate configuration is thus given by

$$\boldsymbol{V}_{t_0+\frac{k}{2\theta}} = \boldsymbol{V}_{t_0} + \frac{k}{2\theta}\boldsymbol{F}(\boldsymbol{V}_{t_0}). \tag{2.42}$$

Once the intermediate configuration is known, $\boldsymbol{F}(\boldsymbol{V}_{t_0+\frac{k}{2\theta}})$ *i.e* the derivatives in the intermediate configuration, are computed.
Finally, the configuration at time $t_0 + k$ is given by an Euler integration scheme where the derivative is now a weighted mean between the derivatives in the current and the intermediate configuration, which writes

$$\boldsymbol{V}_{t_0+k} = \boldsymbol{V}_{t_0} + k\left((1-\theta)\boldsymbol{F}(\boldsymbol{V}_{t_0}) + \theta\boldsymbol{F}(\boldsymbol{V}_{t_0+\frac{k}{2\theta}})\right). \tag{2.43}$$

Despite requiring two configurations computations, the Runge-Kutta integration scheme has the advantage to be a second order method. Moreover, it will be shown in Part 5 that the stability condition on the time step k is less restrictive than with an explicit Euler integration scheme.

2.4.3 Adaptive time step

For an adaptive time step k, Monaghan advises to consider

$$k = \min(0.25k_f, 0.4k_{cv}), \tag{2.44}$$

where k_f is related to body forces and k_{cv} to the Courant condition and artificial viscosity [16]. They are defined by

$$\begin{cases} k_f = \min\limits_i \left(\dfrac{h_i}{\|\boldsymbol{F_i}\|}\right), \\ k_{cv} = \min\limits_i \left(\dfrac{h_i}{c_i + 0.6(\alpha c_i + \beta \max_j \mu_{ij})}\right), \end{cases} \tag{2.45}$$

where the index i spans the particles, j spans its neighbors. The quantity $\boldsymbol{F_i}$ is the body force applied on particle i. Most often, the body force is constant and is the gravitational acceleration.

2.5 Kernels and smoothing length

The kernel approximation constitutes a basis of the SPH method. Its compact support defines a domain of dependence for the particles. In this section, several kernels are presented and the smoothing length influence is discussed.

2.5.1 Kernel examples

The kernels considered in this project are isotropic, $W(\boldsymbol{x}_{ij}, h) = W(r, h)$, with $r = \|\boldsymbol{x_{ij}}\|$. They are defined for $r \in \mathbb{R}^+$. Introducing an adimensional length $\lambda = r/h$ [-], $W(r, h) = W(\lambda, h)$. Five different kernels are considered:

$$\text{Bell-shaped} \qquad W(\lambda, h) = \frac{105}{16\pi h^3} \begin{cases} (1+3\lambda)(1-\lambda)^3 & \lambda \leq 1, \\ 0 & \lambda > 1. \end{cases} \tag{2.46}$$

$$\text{Quadratic} \qquad W(\lambda, h) = \frac{5}{4\pi h^3} \begin{cases} \frac{3}{16}\lambda^2 - \frac{3}{4}\lambda + \frac{3}{4} & \lambda \leq 2, \\ 0 & \lambda > 2. \end{cases} \tag{2.47}$$

$$\text{Cubic spline} \qquad W(\lambda, h) = \frac{3}{2\pi h^3} \begin{cases} \frac{3}{2} - \lambda^2 + \frac{1}{2}\lambda^3 & \lambda \leq 1, \\ \frac{1}{6}(1-\lambda)^3 & 1 < \lambda \leq 2, \\ 0 & \lambda > 2. \end{cases} \tag{2.48}$$

$$\text{Quintic} \qquad W(\lambda, h) = \frac{21}{16\pi h^3} \begin{cases} \left(1 - \frac{\lambda}{2}\right)^4 (2\lambda + 1) & \lambda \leq 2, \\ 0 & \lambda > 2. \end{cases} \tag{2.49}$$

$$\text{Quintic spline} \qquad W(\lambda, h) = \frac{3}{359\pi h^3} \begin{cases} (3-\lambda)^5 - 6(2-\lambda)^5 + 15(1-\lambda)^5 & \lambda \leq 1, \\ (3-\lambda)^5 - 6(2-\lambda)^5 & 1 < \lambda \leq 2, \\ (3-\lambda)^5 & 2 < \lambda \leq 3, \\ 0 & \lambda > 3. \end{cases} \tag{2.50}$$

The main difference between them is the radius κh of their (spherical) domain of dependence. The factor in front of their expression is for normalization. Similar kernels can be used in 1D or 2D, but with different normalization factors.

The scaling parameter κ has a large influence on the method. For a given smoothing length h, a large κ will lead to many neighbors per particle. This will strongly affect the computation time.

2.5.2 Smoothing length

The smoothing length h defines the extent of the kernel. It can be either constant or specific to each particle (variable in *space*). Moreover, it can be chosen constant in time or varying (variable in *time*).

In the latter case, an update equation should be introduced for h. The motivation for considering a variable smoothing length in time is for trying to keep a constant number of neighbors during the simulation.

In many fluid mechanics applications, the smoothing length κh is constant in space for all particles so that Newton's third law is easily fulfilled. However, in some particular cases, one may take advantage of a variable smoothing length [10]. A variable smoothing length is also relevant in astrophysical models.

In this project, the smoothing length has been chosen constant both in space and in time.

The value of the smoothing length has a high importance. Similarly to the scaling parameter κ, a large h leads to many neighbors, then a high computation time. However, a minimum h is required to obtain physical results. If s is the particle spacing, authors recommend to take $h \in [s, 2s]$ (see for example [16]). The influence of h will be analyzed in part 5. Having a sufficient number of neighbors while keeping a reasonable computational cost will give rise to a trade-off.

2.6 Boundary conditions

To be able to model solid boundaries, a way of handling the boundary conditions has to be introduced. As proposed by Crespo [34], it has been chosen to implement the so-called *dynamic* boundary conditions. It consists of modelling the boundaries with the same particles that constitutes the fluid except that the momentum (Eq. (2.27)) and the position (Eq. (2.28) or (2.29)) equations are replaced by pre-defined laws described in appendix A.3.1. The case of fixed particles in space is separated from the moving particles has it does not require any position update nor velocity computation. Consequently, particles are classified in three categories:

- Fluid particles (also called *free* particles),
- Stationary boundary particles (also called *fixed* particles),
- Time dependant boundary particles (also called *moving* particles).

Each of them can be created when defining the initial geometry as explained in appendix A.3. In the following parts, the belonging to one of these three categories will be referred to as the *type* of a particle.

Part 3

Implementation

The theoretical concepts of Part 2 have been implemented in a `C++` open source code available at this address

`https://github.com/GabrielDigregorio/SPH_method`

which can be compiled and run following the instructions given in appendix A.1.

The first section of this part gives an overview of the implemented code and details its most important parts. The second section concentrates on the neighbor search algorithm possibilities.

The sequential simulation of a large number of particles is time consuming. Two levels of parallelization have then been implemented in order to speed-up the simulation time. For both parallelization strategies, the main ideas will first be presented. Then, technical details will be provided. A performance analysis is conducted in the next part.

3.1 Organization

As described in Part 2, several properties are attached to each particle: their position $\boldsymbol{r}$, velocity $\boldsymbol{v}$, density ρ, pressure p, mass m and type. It has been chosen that all this information is stored in several vectors belonging to one data structure called `field`. The parameters that are not related to a particular particle are stored in another data structure called `parameter`.

The algorithm is divided into several parts and distributed among the computer nodes available for the simulation, as represented in the general flowchart given in Figure 3.1. More details about each step are given in the following paragraphs.

Initialization The very first step is to initialize the simulation from the input files provided by the user. Two files have to be provided: a geometry file, defining the domain, the position and the type of each particles and a parameter file, defining the properties which are not attached to one specific particle. Once both files are read, positions and types of every particle are generated. Then the densities are set to a constant value or computed using Eq.(2.38). The pressures and the masses are withdrawn from Eq.(2.34) and Eq.(2.18). Finally, the speeds are set to zero except for moving particles whose velocities are computed from their position law. The proper way to write both input files as well as the available position laws are described in appendix A.2.

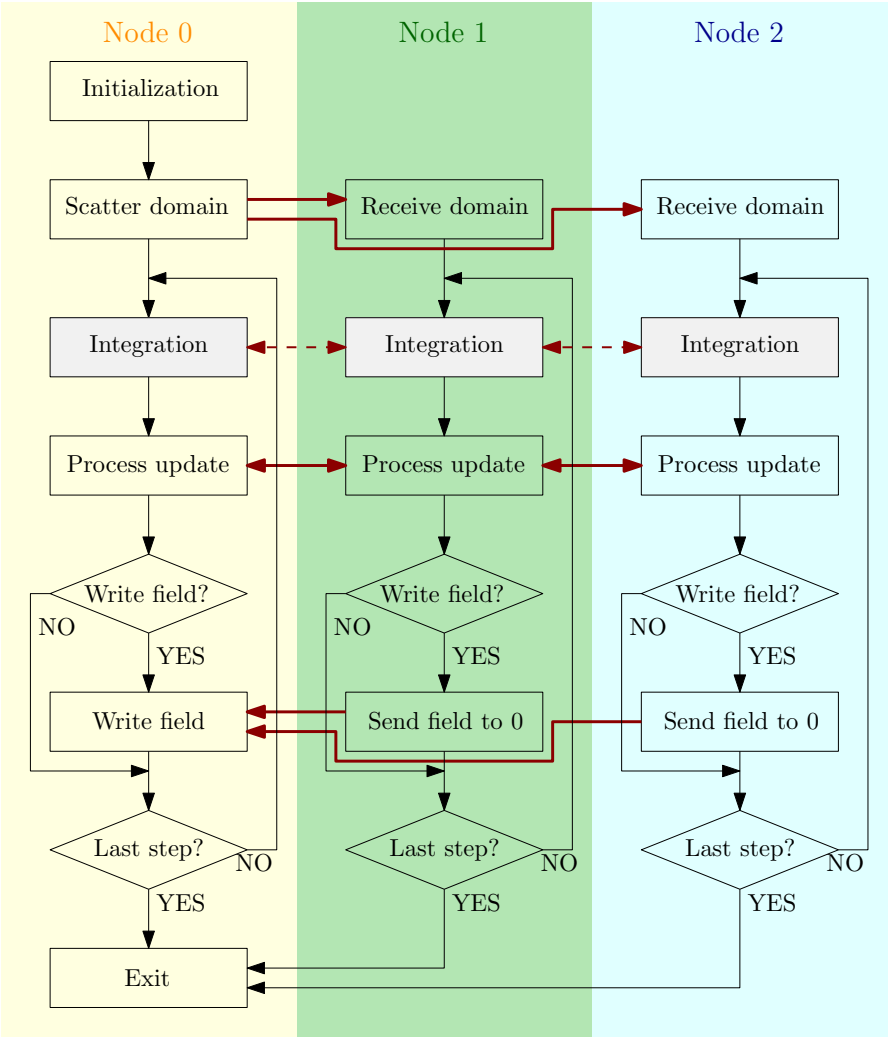


Figure 3.1: General flowchart of the implemented code (example for 3 nodes).

Domain scattering As the first step is only performed on the master node, it is necessary to share the information between the different available nodes. While `parameter` is copied on each node, `field` is split between the nodes. The domain is divided by equidistant planes parallel to the yz-plane. Each node receives the information about the particles lying in its sub-domain and its left and right boundaries. More details about the implementation of this workload division are given in section 3.3. For now, the only thing to know is that each node stores information about the closest particles of its domain (that belong to adjacent domains). These particles constitute the *halo* of the node.

Time loop Once each node has its local `field` filled, the time loop can start. Each iteration of the main loop can basically be divided into three parts: the integration, the process update and the results writing.

Integration The integration consists in applying the equations from Part 2 to derive the information on each particle at a later time. This procedure is detailed in Figure 3.2.

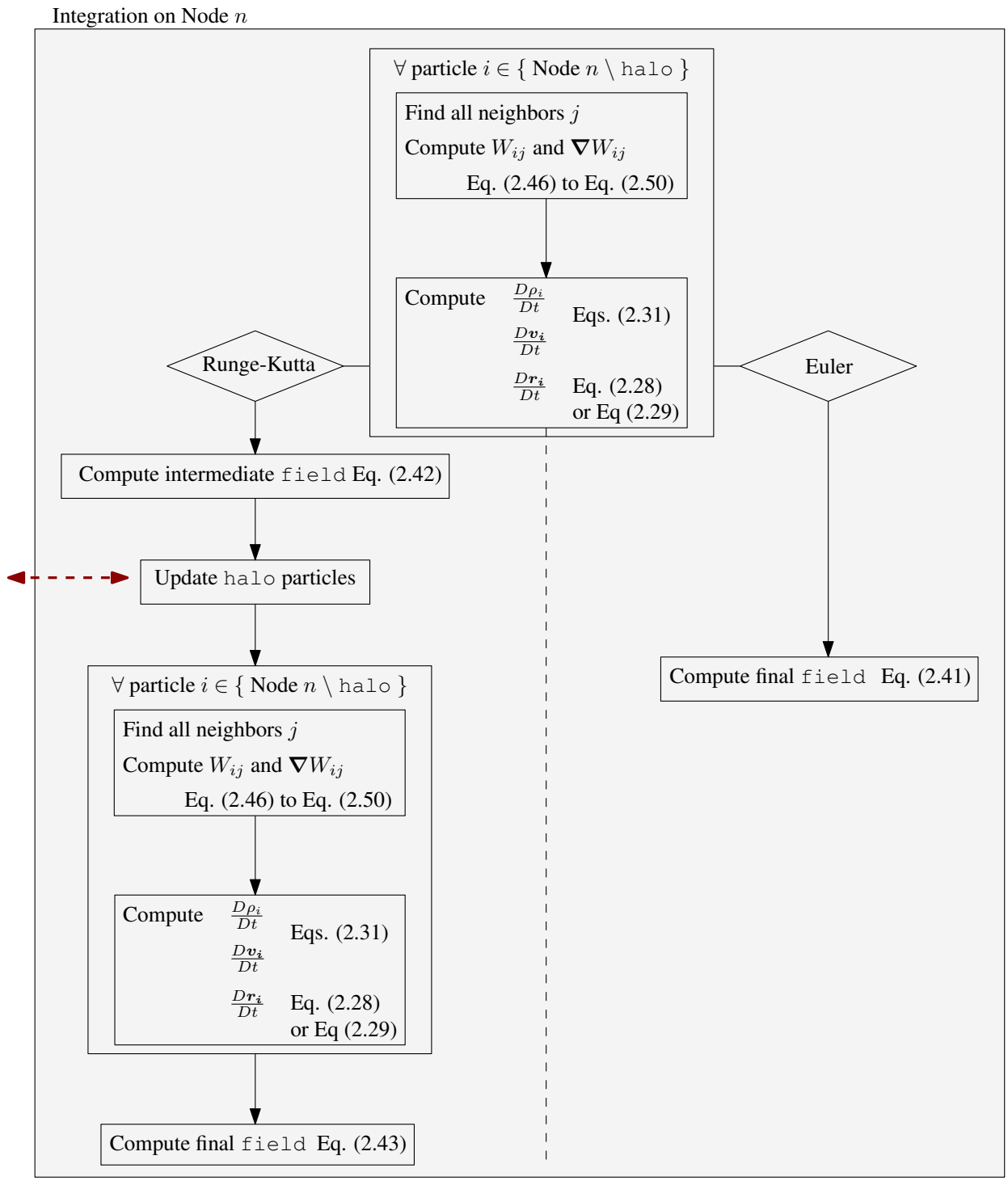


Figure 3.2: Detailed steps of the integration procedure.

Considering only one node n, for all particles i belonging to the interior of this node (and not to the halo) the first step is to find their neighbors j and to compute the kernel values W_{ij} and the kernel gradients ∇W_{ij} associated to each pair. As this neighbors finding is performed for each particles and at each time step, it is important to implement an efficient algorithm. The algorithm

that has been chosen as well as other possibilities are described in details in section 3.2.

The density, velocity and position time derivatives are then computed through Eqs. (2.31) and Eqs. (2.28) or (2.29) respectively. Knowing the time derivatives, a configuration at a later time can be withdrawn from the initial configuration. In the case of an Euler integration scheme, this configuration is obtained thanks to Eq. (2.41) and is the final configuration. In the case of a Runge-Kutta integration scheme, Eq. (2.42) is used and an intermediate configuration is found. This intermediate configuration is, however, incomplete as the boundaries (the `halo`) have not been updated yet. Consequently, a small exchange of information about the particles in the `halo` has to be done between the computer nodes. Finally, the time derivatives in the intermediate configuration are computed as for the initial configuration and the final configuration is given by Eq. (2.43).

During this procedure, each node is nearly independent, indeed, only a minor update of particles information in the boundaries as to be done at the intermediate configuration if a two-step scheme is used. Moreover, iterations of the loop on the particles are independent. Thus, to optimize the performances, shared memory parallelization is used on each node for the particle loop. This second level of parallelization is described in section 3.4.

Process update The process update contains all the exchanges between the nodes needed by the distributed memory parallelization. Indeed, once the final configuration on each node is obtained, it is possible that some particles do not lie in the appropriate domain anymore. Particles may then be transferred from one node to another according to their physical position along the x axis. These exchanges are much more substantial than the ones of the integration step as the number of particles and thus the size of the `field` vectors might change.

The minimal time step requirement computed according to Eq. 2.44 is also communicated to each nodes through this procedure. A detailed explanation of the distributed memory parallelization is given in section 3.3.

Results writing Finally, if the elapsed simulation time since the last results writing is bigger than a user defined write interval time, a global field, gathering all the particles, is created on the master node and an output file is generated. The available output formats are explained in appendix A.4.

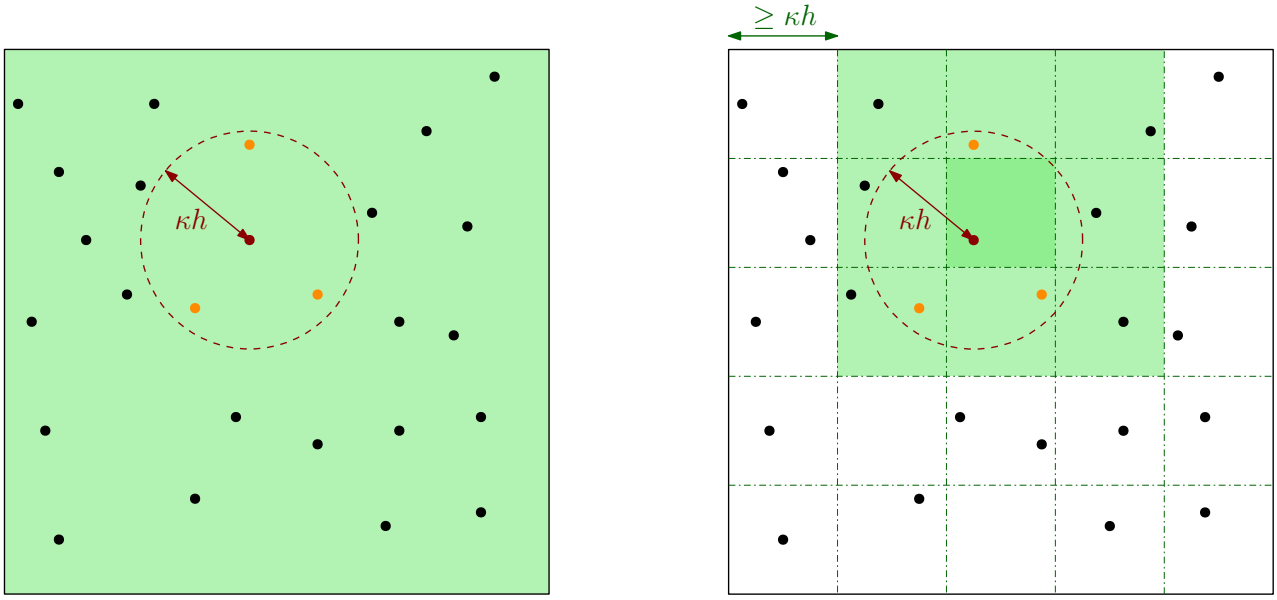
3.2 Neighbor search

The method for determining the neighbors of each particle is of crucial importance for the code to be efficient as it is performed at each time step. Different search algorithms are possible and will be described in the following sections.

3.2.1 Existing algorithms

All-pair search - Naive implementation This algorithm evaluates the distance corresponding to every pair of particles. The distances are compared to the smoothing length and the neighbor particles are stored. This method is very expensive in terms of computation time and resources. For each

of the N particles, $N-1$ distance computations are performed, leading to a $\mathcal{O}(N^2)$ complexity for the algorithm. This complexity reflects its inefficiency considering that the number of particles could rapidly go up to several millions. A schematic two-dimensional representation of this algorithm can be found in Figure 3.3a.



(a) All-pair search. (b) Linked-list algorithm.

Figure 3.3: Schematic representation of neighbor search algorithms.

Linked-list search - Constant smoothing length A much more efficient method consists in subdivising the search domain into identical cubes of side at least equal to the support radius of the kernel κh. With this subdivision, each particle necessarily finds its neighbours in the adjacent cubes. Therefore, this method reduces greatly the spatial searching interval, especially for large domains. A schematic two-dimensional representation of this algorithm can be found in Figure 3.3b.

The complexity of this algorithm is $\mathcal{O}(N)$ if the average number of particles per cube is sufficiently small [9]. Moreover, such an algorithm can be parallelized in a very efficient way as each neighbor search over a cell group can be performed in an independent way.

For an optimal algorithm performance, the box side should be equal to κh. However, there are some good reasons to choose it a bit larger. This is discussed in section 3.3.3 when considering a two-step integration scheme.

A drawback of this method lies in the fact that varying smoothing lengths can not be handled with this type of grid subdivision. If the smoothing length is not constant, other algorithms are necessary.

Tree search - Variable smoothing length In order to cope with variable smoothing lengths, a tree-search algorithm may be used. This method consists in the creation of a hierarchy tree by subsequently splitting the domain into octants containing the particles. This division is performed recursively until only one particle is contained in the current octant. The recursive subdivision of

the domain can be traced into a hierarchy tree, each node representing a subdivision of the previous octant.

Once the tree is defined, the algorithm performs a neighbor search by considering an initial particle and its smoothing length. It is used to define a volume surrounding the first considered particle. The algorithm then walks the tree, checking if the previously defined volume overlaps the volume associated to each node of the tree. The algorithm can therefore be defined by two main steps:

1. Creation of an adaptive hierarchy tree based on the position of the particles. At each node of the tree, the domain is subdivided into smaller domains until the branches lead to a leaf containing only one particle.

2. Neighbor search by evaluating for a given particle i an enclosing cube of side κh_i. At each level, the enclosing volume V_i is compared to the volume occupied by the current node in the tree:

   - If the enclosing volume overlaps the current node volume, the next level is evaluated until a single-particle level is reached. If this single particle is enclosed into the V_i volume, the particle is recorded as being a neighbor.
   - Otherwise, the descent is not performed on the actual path.

The complexity of this algorithm is increased to $\mathcal{O}(N \log(N))$ [9] but this search method allows to use different smoothing lengths κh_i for each particle. A schematic two-dimensional representation of this algorithm can be found in Figure 3.4.

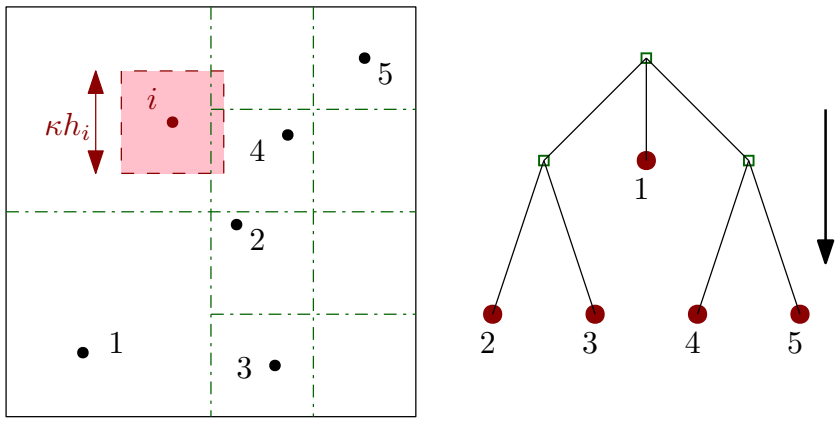


Figure 3.4: Schematic representation of the tree search algorithm. With the domain on the left and the associated hierarchy tree on the right.

3.2.2 Implemented algorithm

Because spatially constant smoothing length will be considered, the chosen neighbor search algorithm is the linked-list algorithm. Remind that the motivation for choosing a spatially constant smoothing length is that for many fluid mechanics applications, all particles have similar physical

interaction with each other. By contrast, for astrophysical models, the best choice is to consider different smoothing lengths due to the gravitational mass which is different for each astrophysical body.

The search function consists mainly in two steps. First, the particles are sorted in cubic cells (or boxes) that cover the numerical domain. The size of these cubes is discussed further in the report (see section 3.3.3). Then, when looping over the particles during the time integration, the neighbors are identified.

Particle sorting After partitioning the numerical domain into small boxes, a list is associated to each of them. This list contains the identifiers of the particles that are contained in the corresponding box. A loop is performed on all the particles to associate them with the cubes. When a particle is just at a boundary between two boxes, it is associated to the box with the highest identification number. The identification number increases with increasing z, y and x as illustrated in Figure 3.5.

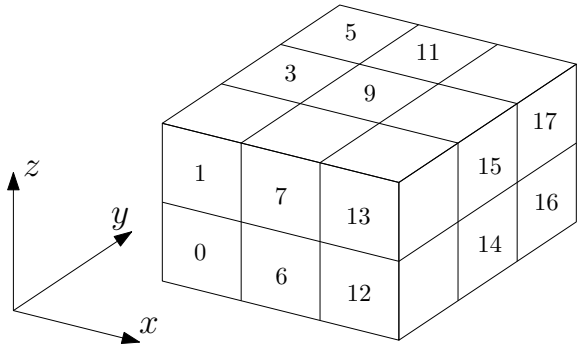


Figure 3.5: Box numbering convention.

Neighbor search and storage For each particle i, the search procedure consists in two nested loops. The first loop spans the boxes that are adjacent to the box of the considered particle and the second loop selects a particle j in the considered surrounding box. This procedure selects a pair of particles (i, j) belonging to adjacent or identical boxes at each iteration. The distance between these two particles is computed and the pair is said to be a neighbor pair if this distance is smaller than κh. The remaining operation is the storage of this neighboring relation. Several possibilities have been considered.

Possibility 1 The neighboring relation is reciprocal. The global situation can then be fully described by a symmetric incidence matrix $[\boldsymbol{M}]$ such that $M_{ij} = 0$ if the distance between particles i and j is greater than kh and $M_{ij} = 1$ otherwise, $\forall i, j \in \{0, \ldots, N-1\}$. Any nonzero element in the matrix indicates a couple of neighbor particles.

The incidence matrix is sparse. An efficient storage could consist in 3 vectors. The first one containing the nonzero values of the matrix and the next two vectors the corresponding row number and column number (starting from 0). The symmetry property of this matrix is taken into account by storing only the upper diagonal part of the matrix. Sparse BLAS level 2 and level 3 routines are available to handle operations with this incidence matrix in an efficient way[1].

[1]For example, the function `mkl_dcoosymv` computes the matricial product between a sparse symmetrical matrix and a vector.

Possibility 2 Each particle has its own neighbor list that is computed just before the force computation and the time integration. As a result, this method does not allow to take advantage of the reciprocity of the neighbor relations.

The first possibility is twice more efficient in terms of number of operations while the second requires much less information storage as the neighbor vector is overwritten for each particle. The second possibility also allows to gather the neighbor search procedure and the particle properties derivatives computation into one single loop over the particles. This is desirable in view of a shared memory parallelization. For this latter reason, the second possibility has been implemented.

During the time integration of the semi-discrete equations, the distance associated to each pair of neighbor particles is needed to know the value of the kernel or of its gradient. All these distances have already been computed during the neighbor search algorithm. To take advantage of this work, together with the list of neighbors, the neighbor search function returns the associated values of the kernel and of its gradient.

3.3 Distributed memory parallelization

This first level of parallelization shares the work among several computers that all have their own memory. Each computer is referred to as a *node*. The communication between the nodes is carried out by MPI (Message Passing Interface). This level of parallelization is highly intrusive and requires many interconnections between the nodes during the simulation.

3.3.1 Strategy

The main ideas for sharing the physical domain among several processors as well as the terminology for the different subdomain parts follow what is presented in [24], [25] and [26].

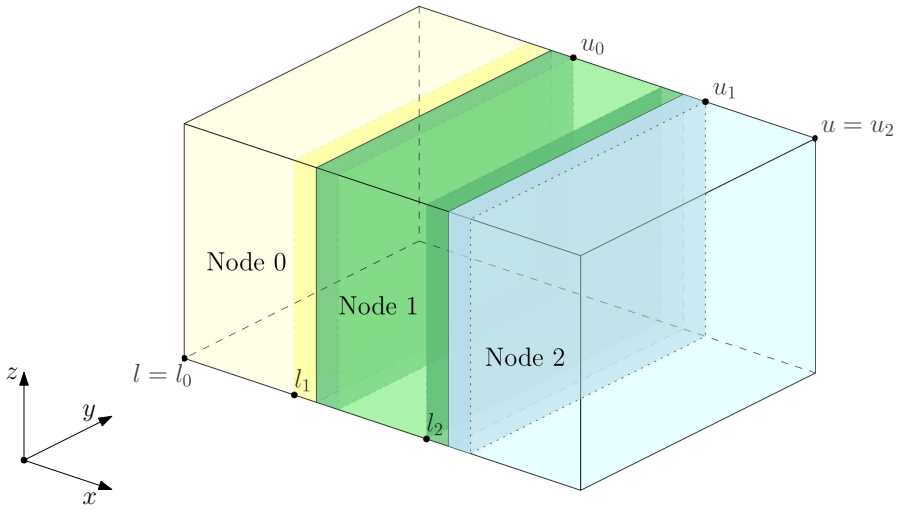


Figure 3.6: Physical domain sharing (example for 3 nodes).

The whole physical domain is delimited by the corners l and u. It is cut along the x axis and shared among the processors. Therefore, each node has only a fraction of the domain to handle. Since there are interactions between the particles over a finite distance, some overlap is necessary in the domain sharing. The local lower and upper corners l_i and u_i of node i are represented in Figure 3.6 for the case of 3 nodes. For a general case, a 2-D projection is shown in Figure 3.7. This figure is described in details in the following paragraphs.

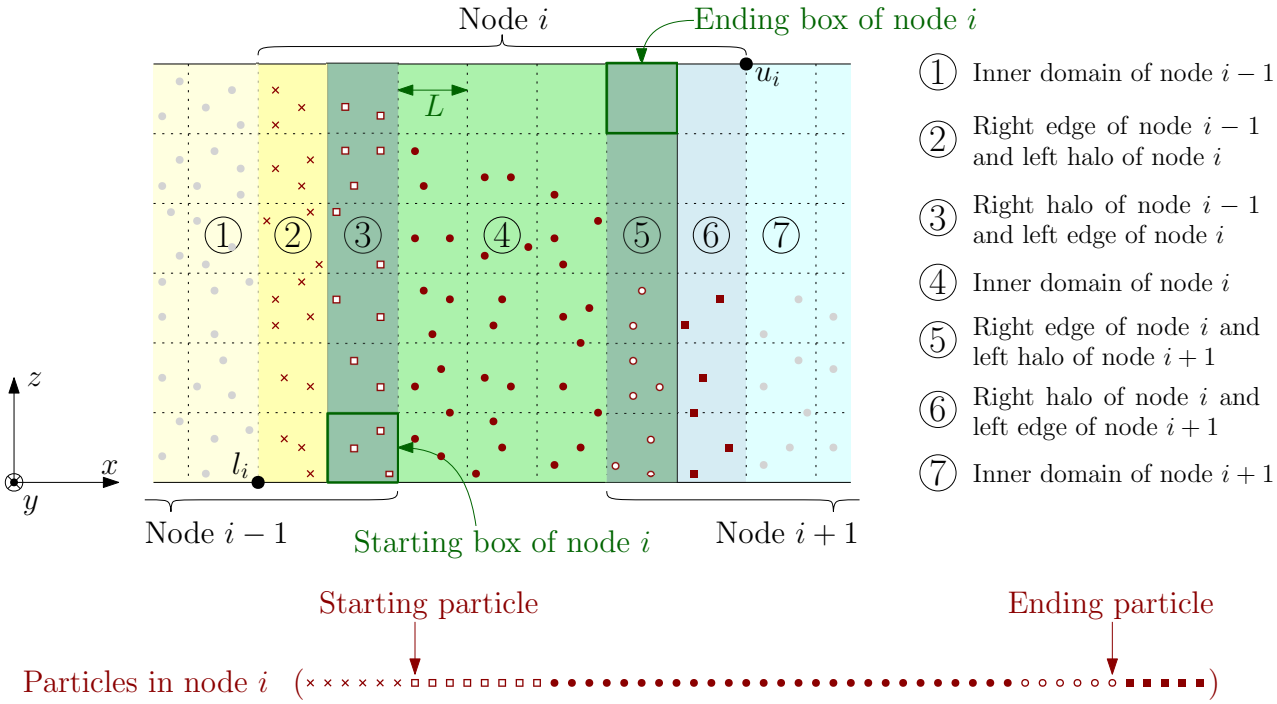


Figure 3.7: Subdomains definition and particle organization.

The linked-list algorithm sorts the particles in boxes to allow efficient neighbor search. The boxes are cubic cells of side L at least equal to the interaction distance κh between the particles. The box boundaries are represented in dotted lines in the figure. The chosen box size L is an important parameter and is discussed in section 3.3.3. Here, it is considered to be known.

All the domain sharing is based on the boxes. They mark out the range of influence of the particles. Each node has a given domain to solve, which is represented by parts number 3, 4 and 5 for node i in Figure 3.7. Parts number 3 and 5 constitute the *edges* of the domain and are notably influenced by some particles that belong to a slice of width L in the adjacent domains, denoted 2 and 6 in the figure. Consequently, each node has to have access to information about these slices. The two slices of width L forming the closest edges of the surrounding domains are referred to as the *halo* of node i. Node i stores their content in its own memory. The halo of a node serves as a "read only" information, the node does not update its content but only reads it[2]. As a consequence of this extra information storage, when looping over the particles for time integration, each process needs to distinguish the particles to update from the others. This is achieved by sorting the particles according

[2]The update of the corresponding particles is performed by the adjacent nodes that send new information at each time step.

to their position, as illustrated in Figure 3.7. The identifiers of the first and last particles to update is known by the node. Similarly, when a loop is first performed over the boxes, the nodes have to know where to start and where to end. This is the purpose of the starting and ending boxes identifiers.

The major difficulty resulting from the Lagrangian nature of the method is that particles can move from one domain to another. Therefore, after each time step, particles have to be resorted. This is done in two steps. Consider the situation of node i. First, particles that have migrated from the set of parts 3, 4 and 5 (see Figure 3.7) to the left or to the right are sent to node $i-1$ or $i+1$ respectively and removed from node i. Node i receives in the same time particles coming from nodes $i-1$ and $i+1$. Second, particles are sorted according to the part they belong to (3, 4 or 5) and particles of the left and right edges (parts 3 and 5) are then sent to nodes $i-1$ and $i+1$ to constitute their right and left halo. In the same time, particles of the right and left edge of nodes $i-1$ and $i+1$ are sent to node i to constitute its left and right halo (part 2 and 6). The starting and ending particles are also updated.

The main ideas of the parallelization have been described. In the next sections, details are given on some implementation choices.

3.3.2 Particle sorting and process communication

The particle sorting is an important procedure. Indeed, it is performed twice on each processor at each time step. The `C++` standard library provides built-in algorithms allowing vectors to be sorted in an efficient way, such as the `std::sort` function. The sorting algorithm used in the `C++11` standard is a hybrid of quicksort and heap sort called introsort. This algorithm is of complexity $\mathcal{O}(N \log N)$, N being the size of the vector to sort. In this case, N is the number of particles in each domain.

The sorting is conducted by using a *key* vector. It contains *key values* associated to each particle. The values depend on the position of the particle, in a way that is specific to each sorting. The sorting algorithm will gather in the vector all the particles that have the same key value. Since many vectors have to be reordered (position, speed, pressure,...), a specific sorting strategy is adopted. A vector of *pairs* is created. Each pair is related to a given particle. It contains the position of this particle in the unsorted vector and the key value associated to it. This vector of pairs is then sorted and particles with the same key value are gathered. Once the pairs are sorted, the properties of each particle can be reordered successively following the established order. A single temporary vector (that is allocated only once) is filled with the reordered values and is then swapped with the vectors. This minimizes the required memory resource.

After each time step, every processor evaluates the particles contained in its domain in order to determine the particles that have migrated to another domain and the particles that can have an influence on adjacent domains. The whole update procedure consists in three main steps. Figure 3.8 illustrates the different steps. They are described below.

Step 1 The first operation is to delete particles that constituted the halo of the node. The vectors are cut and only particles between the starting and ending particles identifiers are kept. Since each node performs this operation, every particle is represented only once in the cluster after this step.

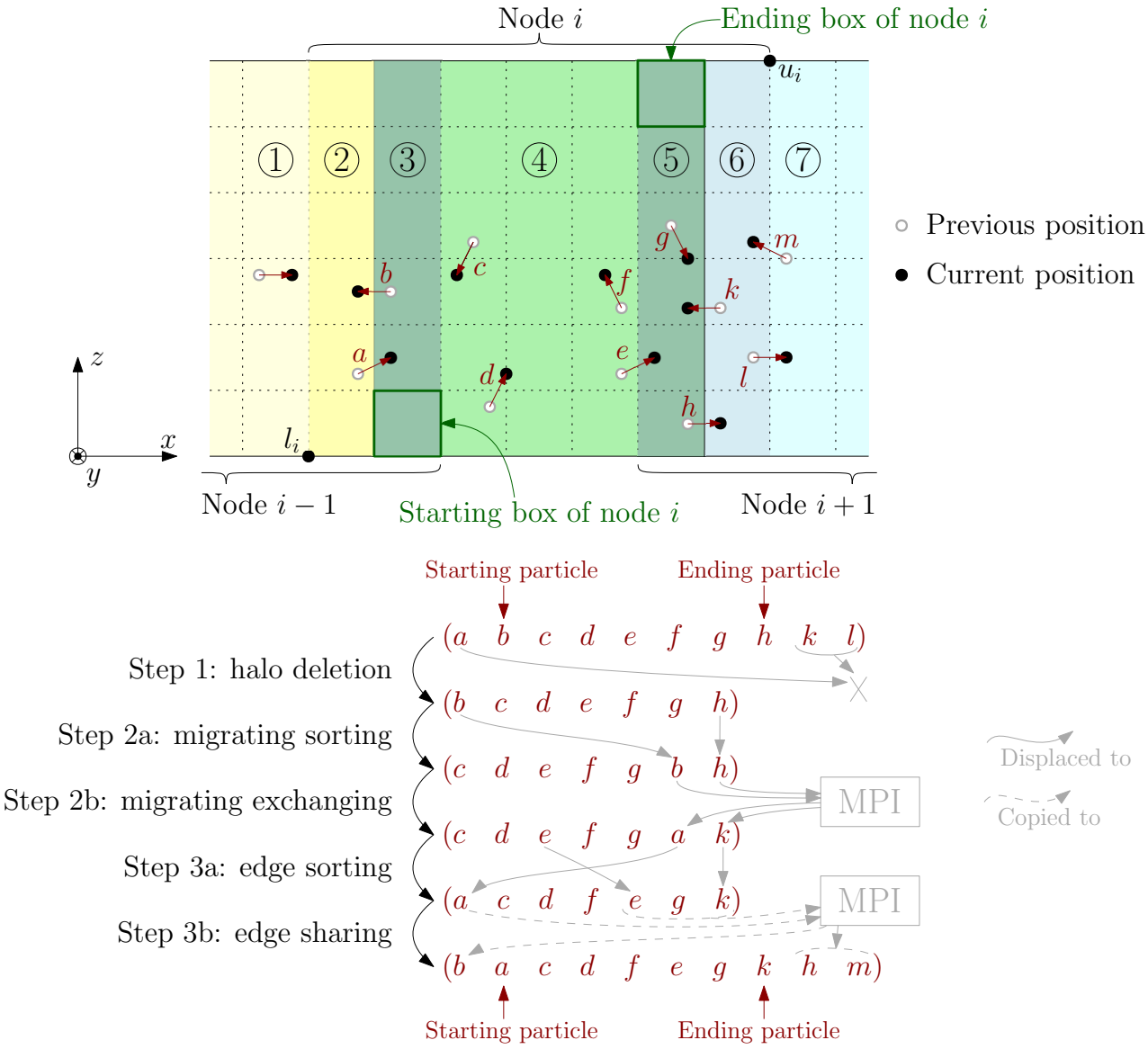


Figure 3.8: Three-step update procedure.

Step 2 The second step is to determine which particles have migrated. This operation is performed by sorting the particles of each node with respect to a sorting key giving information on their location in the domain. Typically, the key tells if a given particle is still inside the domain (parts 3, 4 or 5 of Figure 3.8), has migrated to the left node domain (part 2) or has migrated to the right one (part 6). Once the particles have been identified, they need to be sorted to allow efficient communication between processors. It is indeed more efficient to exchange data stored in successive memory addresses. When the migrating particles are identified and pushed at the end of the vector, their number and their properties are sent to the relevant adjacent processor through *MPI* communication. The send and receive operations are performed according to the scheme described in Figure 3.9, where the even and odd processors send and receive data alternatively. Four operations are performed. This ensures to avoid deadlock issues. Received particles are stored in a buffer waiting for the node to delete those that it had sent. They are then put at the end of the vectors.

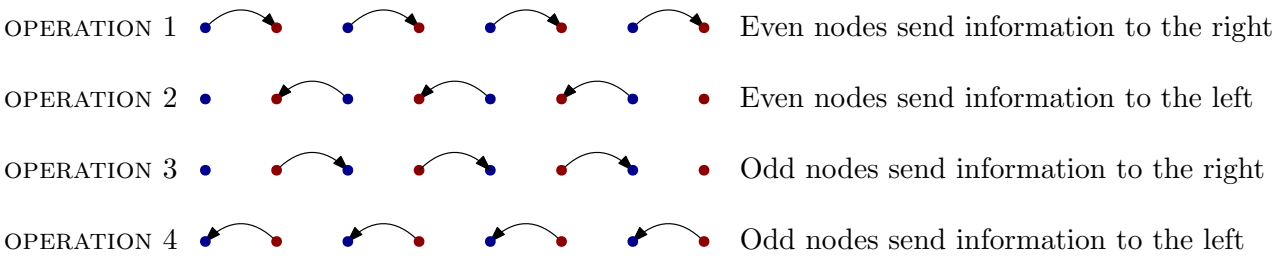


Figure 3.9: Communication scheme of the *MPI* processes.

Step 3 After the second step, each node has an updated list of its particles. A third step is necessary to share their edges that will constitute the "read-only" halo of surrounding domains. The sorting and exchange operations of these overlap particles is very similar to the previous step. Only the key values are modified. Now, they tell if a particle is in part 3, 4 or 5 (see Figure 3.8).

3.3.3 Box size

The concept of box is introduced by the linked-list neighbor search algorithm and is significantly exploited in the distributed memory parallelization implementation. As mentioned previously, the box side L should be at least equal to the interaction distance κh between the particles. If it is chosen larger, the neighbor search procedure is less efficient. However, some restrictions appear when considering a two-step integration scheme.

For the two-step Runge-Kutta integration scheme, two particle updates are performed within a single time step. For the scheme to be consistent, the identifiers of the particles should remain identical between the two steps. Since the MPI particle exchange does not preserve the particle identifiers, it has to be avoided. Then, all the pairs that have the possibility to become neighbors during one of the two steps must be in adjacent boxes already at the beginning of the time iteration.

This is achieved by choosing first a box side a bit larger than needed, say $1.1\kappa h$. And second, by imposing the time step to be sufficiently small such that the fastest particle cannot travel more than $0.05\kappa h$ during *one half* of the time step. Equivalently, it should not be able to travel more than $0.1\kappa h$ during *one* time step[3]. The situation is illustrated in Figure 3.10.

In the MPI parallelization context, this modification is of crucial importance. In the sequential version, since it avoids to sort the particles twice within a single time step, this modification slightly improves the global performance of the code.

When using the explicit Euler integration method, no intermediate state is involved and the minimum box size κh can be chosen. Note that it could still be chosen larger to save particle sorting across successive time steps. However, this would require to lose code readability for a negligible time saving.

[3]This reasoning is valid only if $\theta = 1$ *i.e* when the intermediate configuration is in the middle of the previous and the next configurations. If θ is arbitrary, the particles should not travel more than $0.05\kappa h$ during $k/2\theta$.

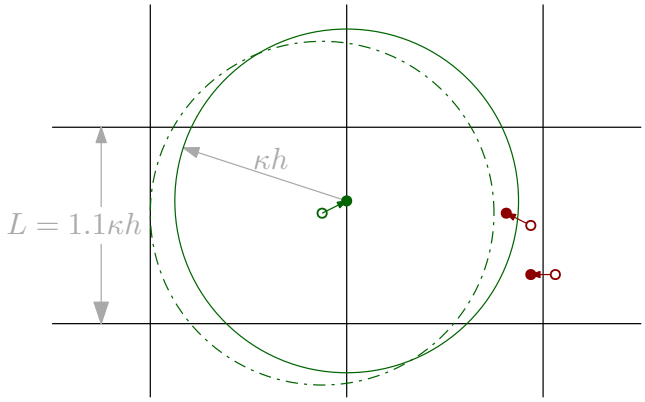
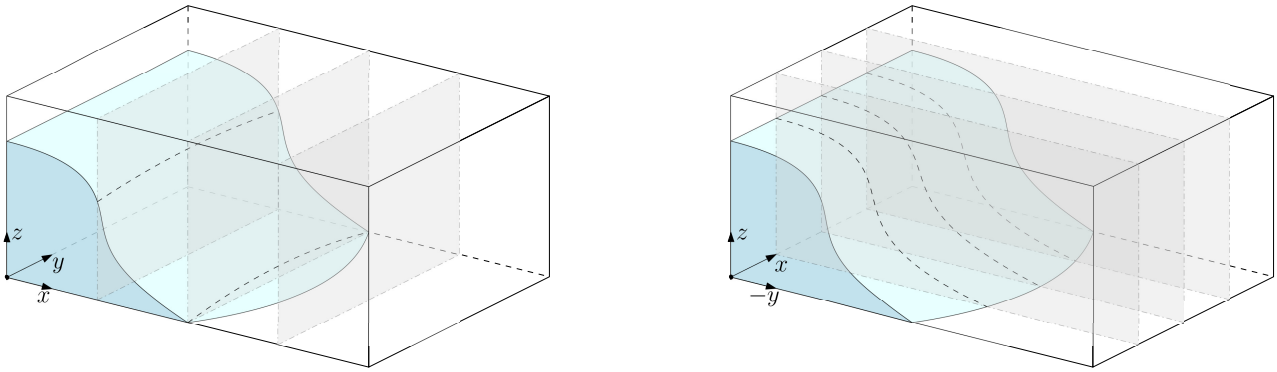


Figure 3.10: Box size for Runge-Kutta integration scheme.

3.3.4 Implementation details

Halo identification As said in the previous sections, for each node to know where to start loops on particles or on boxes, some identifiers are defined. This information is stored in a structure `SubdomainInfo` that contains all the relevant information for the MPI parallelization.

Domain partition direction In the current implementation, the domain is always partitioned along the x direction. In some cases, that leads to a very inefficient computation load sharing. This is illustrated in Figure 3.11a where the first node has much more fluid particle to process than the others. A better solution is to define the geometry in another direction, such that the x axis corresponds to the longitudinal dam direction as in Figure 3.11b where the fluid volume is shared evenly.



(a) Inefficient domain sharing.

(b) Efficient domain sharing.

Figure 3.11: Two possibilities for domain sharing.

Note that for better performances, it would be possible to adapt dynamically the size of the domains to react to the fluid movement. That can constitute a crucial point for performance optimization in some geometries and that would involve major changes in the implementation. However, the aim of this project is to model wave propagation and big load changes are not expected. This improvement has not been implemented.

Minimum domain width The domain partition is based on the concept of boxes. The width of each subdomain is an integer multiple of the box size. For the parallelization strategy to work

correctly, the domains must be at least two-box-wide such that they contain at least one slice for the left edge and one for the right edge. Of course, this is not efficient and having an inner domain is preferred. Strong scaling analysis will illustrate this remark.

Output file writing The writing of output files for post-processing is time consuming and cannot easily be parallelized. Each time an output file has to be written, particles are gathered in the master node with all their properties. The file is then written sequentially. This operation is expensive and can have an important influence on the speed-up.

3.4 Shared memory parallelization

The second level of parallelization exploits the multiplicity of CPU inside one node by means of OpenMP (Open Multi-Processing) instructions. These instructions take advantage of the shared memory between the CPU. This level of parallelization is barely intrusive and consists in some instructions only.

3.4.1 Strategy

This step of parallelization consists in splitting the various iterations of a loop among the CPUs. In order to achieve such a distribution, independence of the iterations has to be ensured. An important aspect of the shared memory parallelization is the definition of a scheduling type. If the amount of work performed by each loop iteration is almost the same, a static scheduling allows a faster execution of the instructions as no load balancing has to be performed. However, if the amount of work is not the same on each loop iteration, a dynamic scheduling should be promoted even though a certain overhead is required.

3.4.2 Implementation

OpenMP instructions over the main loops in the time integration functions allow a distribution of the iterations over the particles and boxes in order to decrease the computation time. As the box content can vary, a dynamic scheduling is proposed when looping over the boxes. The same instruction is provided when looping over the particles as their type (free or boundary) make the computation work vary.

Note that all loops are not parallelized. In particular, when sorting the particles in the boxes, `push_back` operations are used. This would require the declaration of a *critical* section to avoid any data race. The resulting parallel loop is much less efficient than the sequential one. A possibility would be to get rid of the `push_back` operations. However, the corresponding loop is not time consuming and it has been chosen to let it sequential.

Part 4

Performance analysis

This part is devoted to the analysis of the implementation performance. At first, the sequential code is tested with an increasing number of particles. It will highlight the necessity for using a parallel implementation, whose performance is then analyzed. The parallel implementation analysis is conducted in two separate ways. A strong scaling analysis provides information about the communication cost between process while a weak scaling analysis gives an idea about the limits of the code in terms of number of particles.

4.1 Sequential Performance

Throughout this section, the sequential version of the implemented SPH method will be tested and its performance will be evaluated. The performance analysis of the initialization step, the Runge-Kutta integration step and the writing step will be carried in two main parts. First, the influence of the total number of particles on the execution time is evaluated. Then, the smoothing length is varied on a fixed geometry and its influence is also quantified.

4.1.1 Increasing number of particles

In order for the number of particles to be the only changing parameter, all the parameters except the size of the analyzed geometry are kept unchanged. The considered geometry is represented in Figure 4.1. Pools with different values of W_{fluid} were used, with a constant spacing of 5 [m] on each side of the fluid. Therefore, a controlled increase of the number of particles could be achieved.

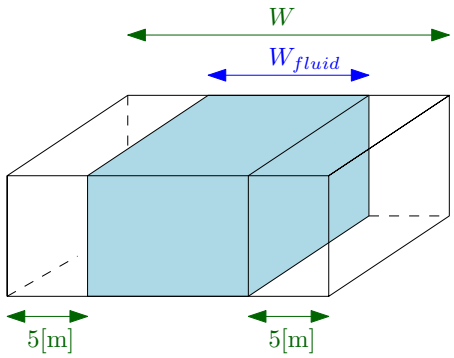


Figure 4.1: Schematic representation of the geometry used for the sequential tests.

A total of 10^4 time steps were computed, with a writing interval every 10^3 time step. The particle spacing was $s = 0.5$ [m] and the smoothing length $\kappa h = 0.6$ [m]. The obtained timings can be found in Figure 4.2. These same results but distinguishing the initialization, computation and writing times can be observed in a logarithmic scale in Figure 4.3. It is directly possible to see that even though they are increasing with the complexity of the problem, the writing and initialization delays stay always negligible compared to the computation time. The latter, as can be seen in Figure 4.2 and 4.3, exhibits nearly a linear behaviour.

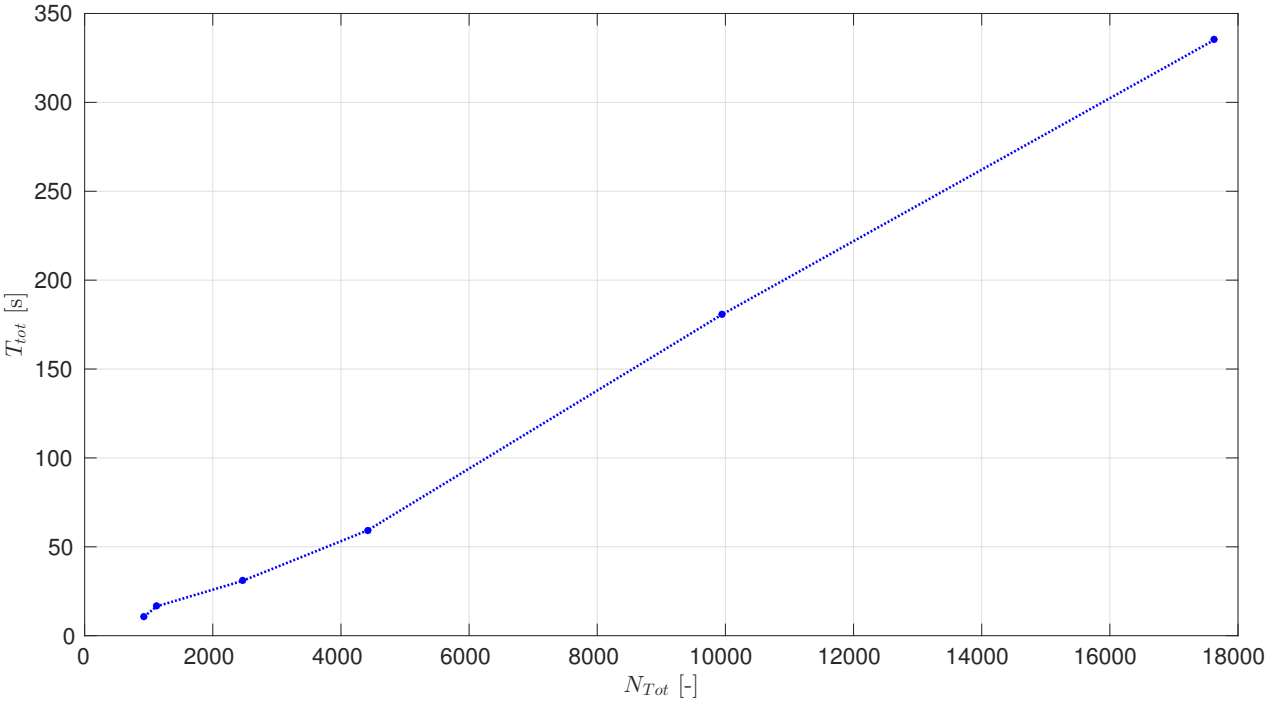


Figure 4.2: Influence of the total number of particles on the execution time.

4.1.2 Increasing smoothing length

In this section, the effect of an increase in the smoothing length is observed. One of the geometries used in the previous section has been used with fixed dimensions. The same parameters have been

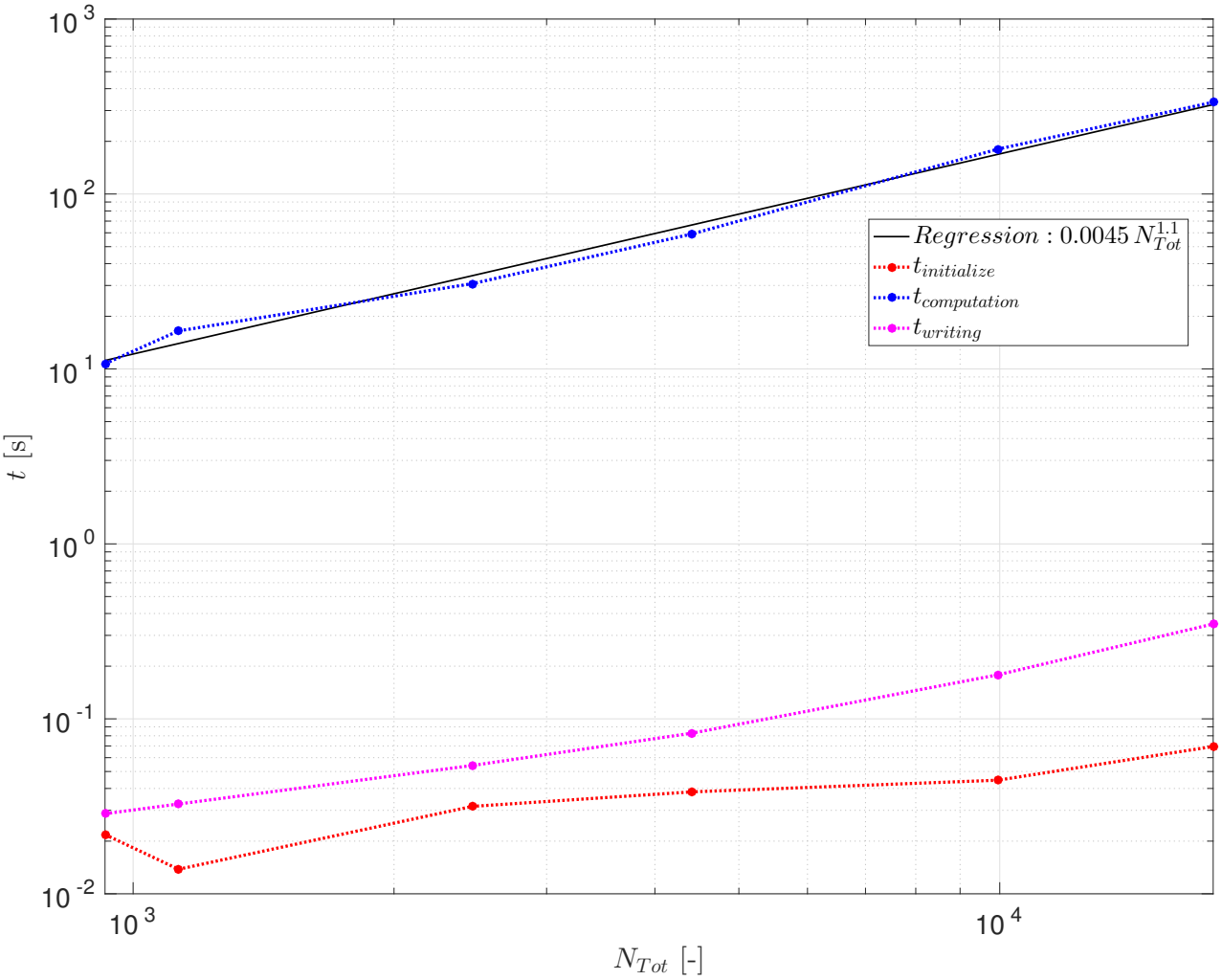


Figure 4.3: Influence of the total number of particles on the initialization, computation and writing times.

used, except the smoothing length that has been varied in the range $\kappa h \in [0.6\ ,\ 2.5]$ [m] with 0.5 [m] increments. This increase in the smoothing length increases the range of influence of each single particle, resulting in an increase of the number of neighbours. The results are reported in terms of the mean number of neighbours, computed over all the particles at the first timestep. The relationship (globally a cubic dependence in 3D) between the smoothing length and the mean number of neighbours can be found in Table 4.1.

κh [m]	0.6	1.0	1.5	2.0	2.5
$\bar{N}_{\text{neighbours}}$ [-]	5.4	23.2	84.2	181.6	327.2

Table 4.1: Mean number of neighbours compared to the smoothing length

The obtained timings as a function of the initial number of neighbours $\bar{N}_{\text{neighbours}}$ can be found in Figure 4.4. As one can see, there is no global trend in the variation of the initialization and the writing time.

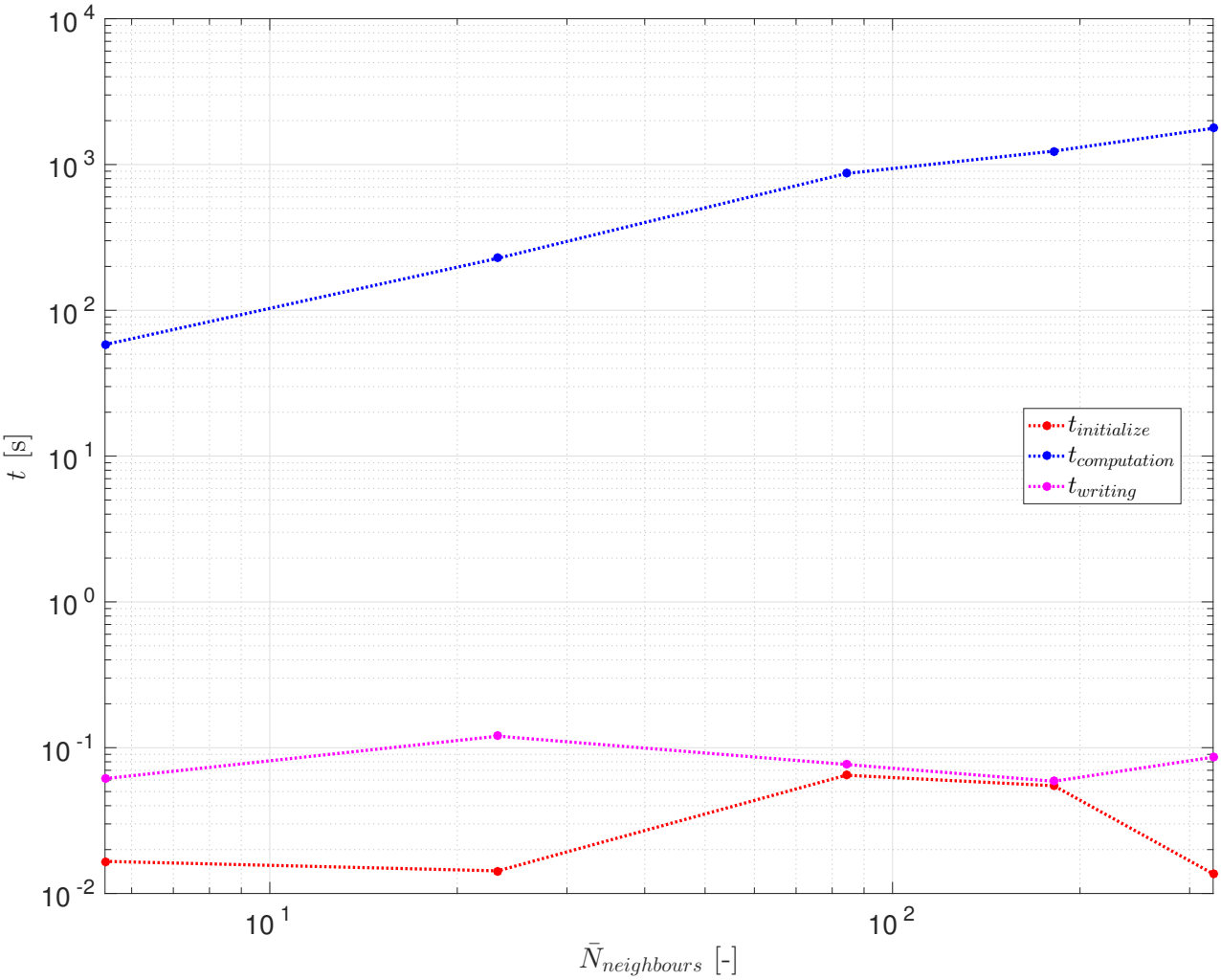


Figure 4.4: Influence of the smoothing length (through the mean number of neighbours) on the initialization, computation and writing times.

The number of considered neighbouring particles, however, has a large influence on the computation time. Therefore, the smoothing length should be chosen wisely, not too small to ensure reliable physical results but not too large to keep bearable execution times.

4.2 Strong scaling

4.2.1 Distributed memory parallelization

Since the distributed memory parallellization involves a fixed domain sharing, its performance is expected to be geometry dependent. This has already been mentioned in section 3.3.4. To estimate the importance of this remark, both an ideal and a non-ideal cases have been considered.

Ideal case definition A usual geometry for SPH model validation is the dam break case. It corresponds to a rectangular volume of fluid enclosed in a larger rectangular box. The fluid is initially

out of static equilibrium. The geometry defined for the simulations is represented in Figure 4.5. The x-axis is chosen in order to optimize the load balancing between the process (the domain sharing is performed along x). The length along x is chosen quite large to allow the use of many process for the scaling study (each domain slice has to be at least two-box-wide).

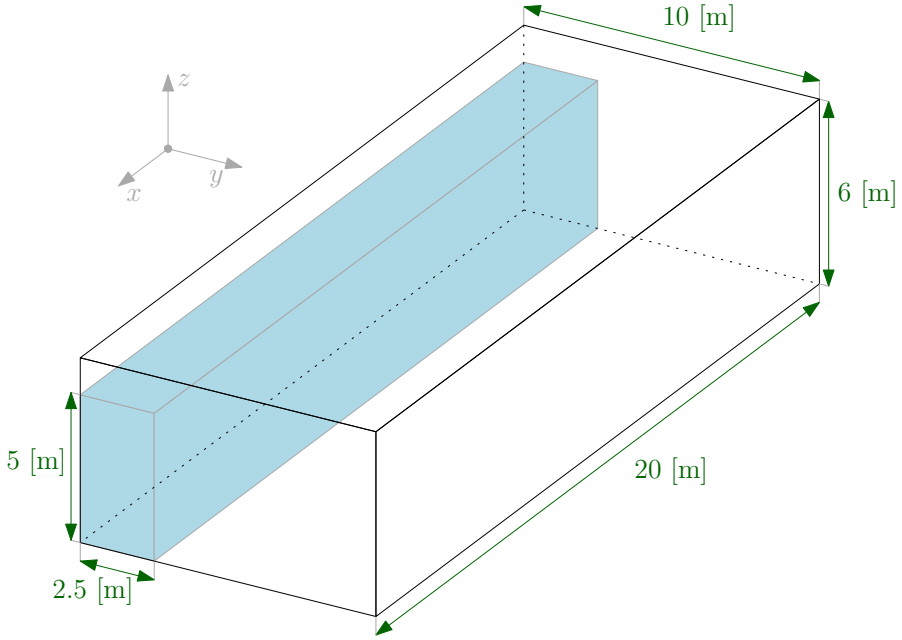


Figure 4.5: Geometry of the dam break case.

The computation load for each process is indeed stable during the simulation as the problem is nearly x-independent as suggested by Figure 4.6. Note that a detailed study of the physical results is conducted in section 5.4.

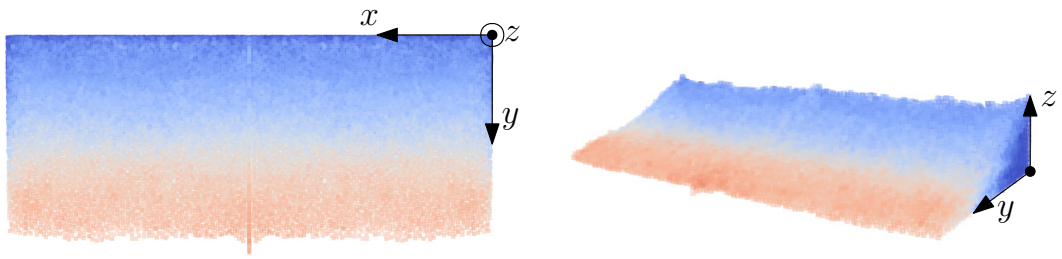


Figure 4.6: Top and perspective views of the results at $t = 1.2$ [s].

Non-ideal case definition As an example of a non-optimized case for the domain sharing, the geometry of figure 4.7 is considered.

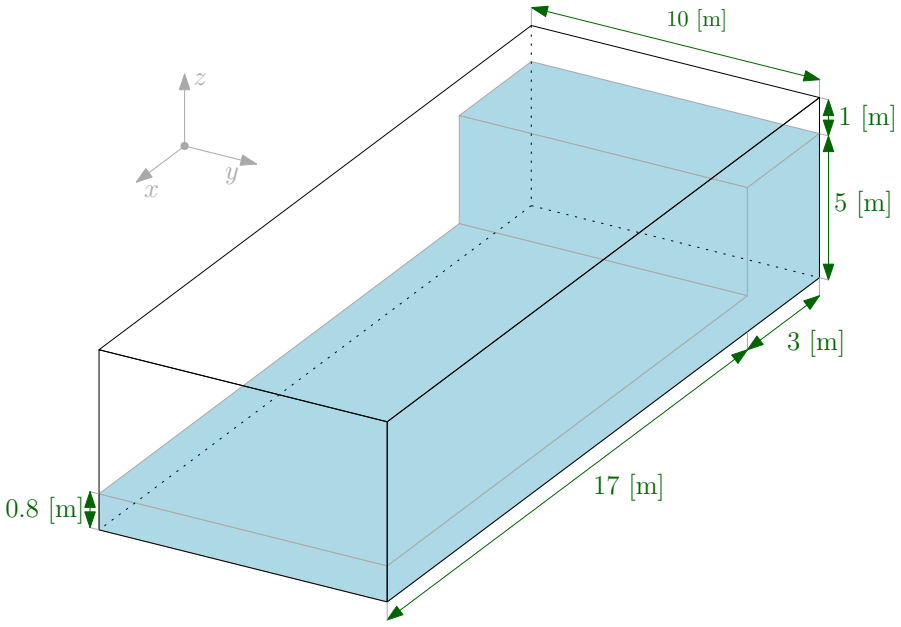


Figure 4.7: Geometry for the non-ideal test.

Scaling The two simulations have been run on variable numbers of processors using only the MPI parallelization. The same number of particles was considered for all simulations (57,700 for the ideal case and 62,350 for the other one), this corresponds to strong scaling. The obtained speedups are given in Figure 4.8.

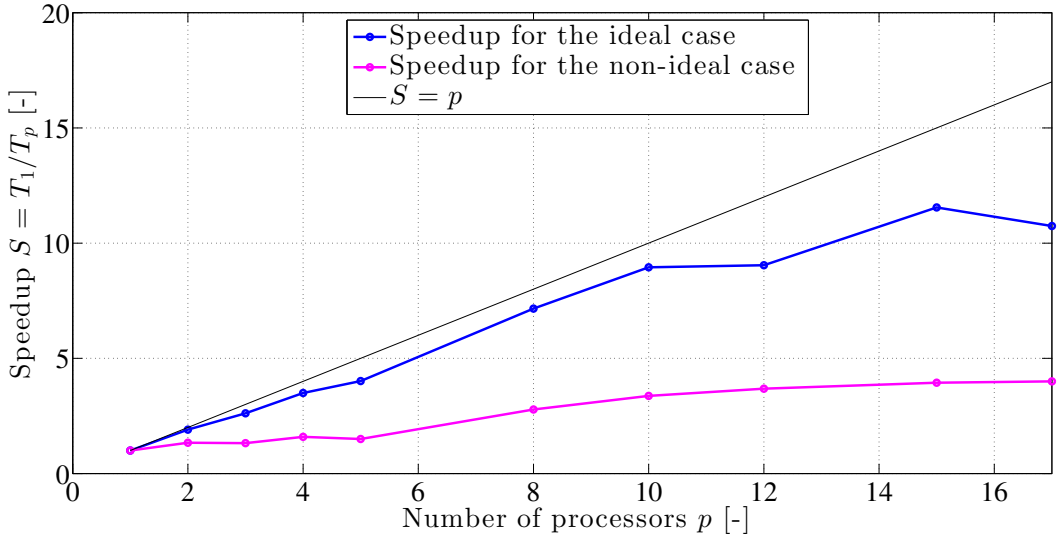


Figure 4.8: Speedup evolution for two test cases with only MPI parallelization.

In the ideal case, the speedup evolution remains close to the line $S = p$. However, a slight drift is observed when the number of processors increases. This is due to the increased proportion of

message exchange.

The speedup evolution is not perfectly regular. This can explained by the discrete nature of the MPI parallelization. Each process is forced to handle a domain slice whose width is an integer number of boxes. Because the total number of boxes along the width of the domain is not always a multiple of the number of process, the physical size of the process domain can vary from one to another. Moreover, the particles are not necessarily evenly shared among the boxes. These two observations lead to the conclusion that even for the dam break geometry, some process can have to handle more particles than others. This phenomenon is negligible when the width of the domain is large but becomes important when it decreases. For example, when using 12 process, one node is initialized with 10% of the particles while the others have between 8% and 8.5% particles. With 10 process, the distribution is more uniform and each node has 10% of the particles. As a result, there is absolutely no difference between the two simulations because all process have to wait for the slowest one, which is identical in both cases.

The speedups obtained for 15 and 17 nodes illustrate the variability of the simulation time. Two identical simulations do not always take exactly the same simulation time. The times can for example depend on the distribution of the nodes in the cluster. Average values could be computed to have more accurate speedup evaluations.

As expected, the non-ideal geometry leads to very low speedup values. This constitutes a motivation for implementing a dynamical domain sharing. As already discussed, this has not been considered in this project.

As a final remark, note that the writing operation is expensive (not represented in the figure). The data in Figure 4.8 were obtained when asking for only 4 output files during the whole 2 second simulation. When more output files are asked, the speedup decreases importantly because the writing operation is performed in a completely sequential manner.

4.2.2 Shared memory parallelization

To estimate the efficiency of the shared memory parallelization, the dam break geometry (Figure 4.5, 56,700 particles) has been run on several numbers of CPU per node. Two and five nodes have been considered. The obtained speed-up are plotted in Figure 4.9.

The performance of the OpenMP parallelization is nearly ideal when a small number of CPU is used. A deviation from the ideal speed-up is observed when the number of CPU increases or when more nodes are considered. This can be explained by the fact that all the computation work is not parallelized. The remaining sequential parts take more importance when the parallel sections are more distributed.

Moreover, other simulations suggest that a sufficient amount of work per CPU is necessary for ensuring a reasonable speed-up.

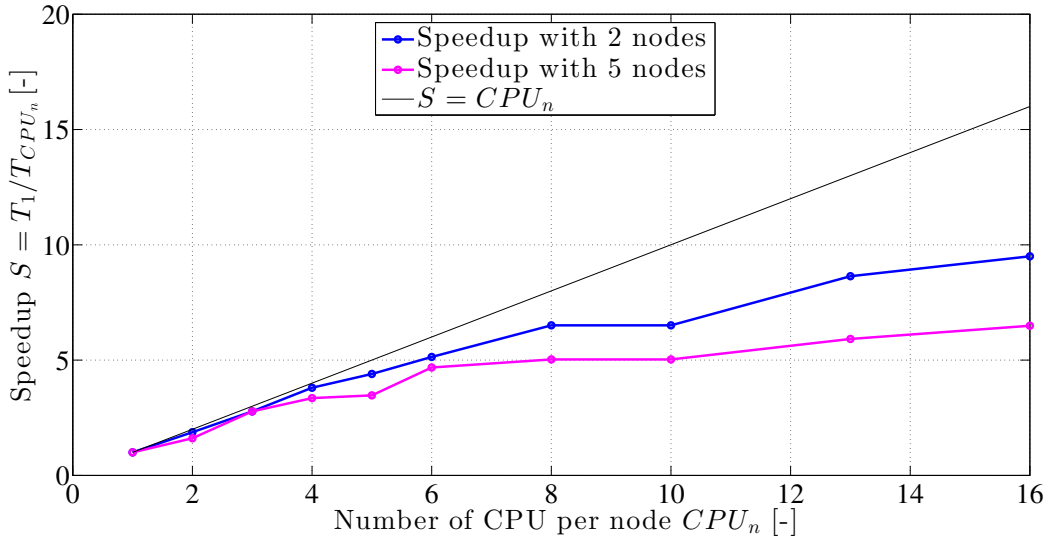


Figure 4.9: Speedup evolution for the dam break case (56,700 particles) with OpenMP parallelization on 2 and 5 nodes.

4.3 Weak scaling

Since the computation work is not fully parallel, a weak scaling study can be performed in order to compare the proportion that sequential parts takes. The test consists in increasing the total work and the number of computer nodes together to keep a constant the amount of work per processor.

The test case considered here is the dam break. It allows to keep the amount of work per processor constant by simply increasing the length along the x-axis proportionally to the number of processor. The used parameters are given in Table 4.2.

Parameter	Value	Units	Parameter	Value	Units
κh	0.2684	[m]	B	175,000	[kg/m^3]
c	35	[m/s]	T	2	[s]
α	0.1	[-]	Kernel	Bell shaped	/
β	0	[-]	k	0.0005	[s]
s	0.2237	[m]	Nb. of particles/CPU	43,000	[-]
Number of time steps	4000	/	Writing interval	0.1	[s]

Table 4.2: Simulation parameters for the dam break case for the weak scaling test.

The main sequential parts of the code are the writing and the initialization of the variables. This is performed on all the particles by the master node only. The initialization part of the code is not represented in Figure 4.10 because it is negligible compare to the other computation time. The complexity of the writing part is observed to be $\mathcal{O}(N)$ as suggested by the linear increase in Figure 4.10 (the number of processor being proportional to the number of particles). The communication time between the nodes is constant because the load that is exchanged between two processors is constant whatever the total size of the domain is the considered case.

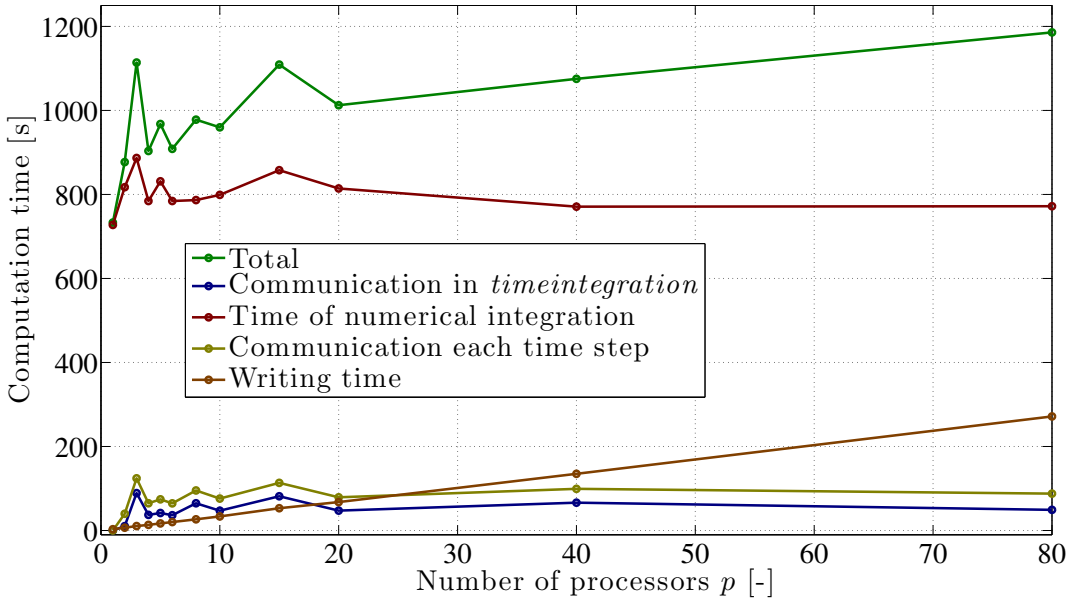


Figure 4.10: Weak scaling results. Computation time for the different parts of the code with respect to the number of processors.

Part 5

Test cases and applications

The implemented SPH code has been run on simple geometries for physical validation. The first test consists of a fluid cube subjected to gravity. This test has been compared to the analytical solution of a free falling mass. In the second geometry, a fluid cube is initially at rest inside a cavity and the hydrostatic pressure should set up. Then, the first test case is extended to a free falling cube impacting a rigid plane. The influence of the fluid compressibility on the results has been studied. In section 5.4, the dam break case numerical results are compared to experimental measurements and the influence of several physical and numerical parameters is analyzed. In section 5.5, the ability of the SPH method to reproduce gravity wave propagation is studied by considering some theoretical results.

5.1 Free falling cube

The geometry of the test consists of a single cube of fluid subjected to gravity. The numerical results are compared to the exact solution, which is, for each particle i,

$$\begin{cases} x_i = x_{0i} \\ y_i = y_{0i} \\ z_i = z_{0i} - \dfrac{gt^2}{2} \end{cases} \tag{5.1}$$

where (x_{0i}, y_{0i}, z_{0i}) is the initial position of particle i and g is the gravitational acceleration.

Simulation shows that the displacement, velocity, pressure and density fields are perfectly homogeneous throughout the particles. This is due to the relative velocity between each neighbor pair which is identically zero, yielding no density change and thus no pressure increase and no spatial velocity gradient creation.

This simple test case has been simulated with both Euler and Runge-Kutta integration schemes in order to compare them. When using Euler numerical scheme, one can see in Figure 5.1 that the solution is less accurate than with a Runge-Kutta scheme. All results are above the analytical curve which is in good agreement with the fact that the Euler method keeps a constant velocity between two time steps, while the speed should be constantly modified accordingly to the constant acceleration.

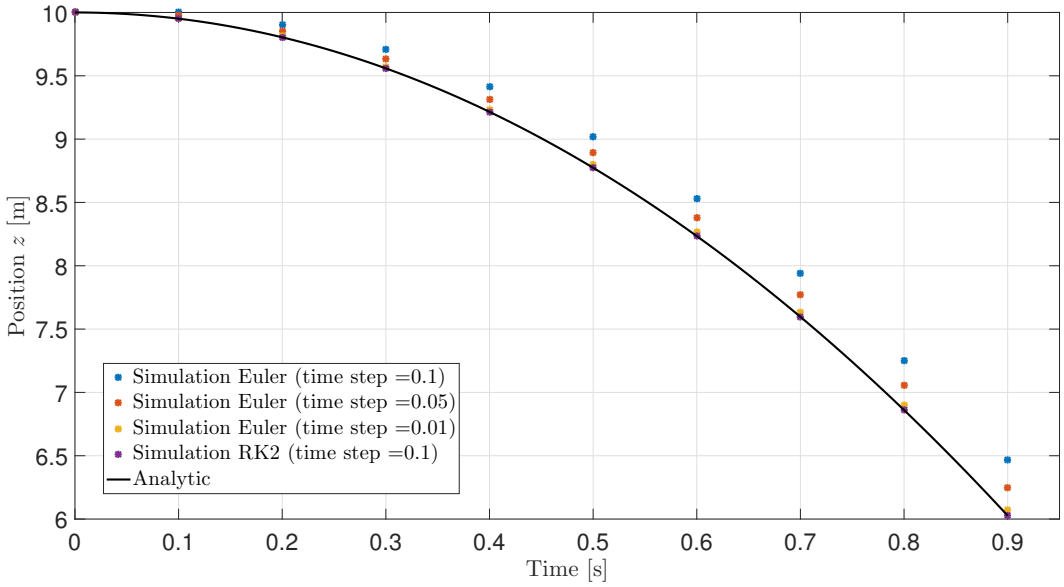


Figure 5.1: Numerical scheme comparison on a Free Falling Cube test case.

When using the Kunge-Kutta numerical scheme on this specific test case, one can see that the analytical solution is exactly reached. Therefore, a constant acceleration is exactly represented by a second order Runge-Kutta method. This is obvious by the fact that a method of order two can perfectly represent a quadratic behaviour.

5.2 Still fluid

The second test has been designed to analyze the hydrostatic pressure establishment. A cube of water is put into a motionless container. When particles are generated, they are fixed to a specific position on a grid with an equal spacing s from each other. This initial configuration does not represent the equilibrium configuration and a new one takes place. The bottom particles are squeezed due to the mass of the above particles accordingly to the compressibility parameter B.

This particular test is compared to the simple analytical solution of hydrostatic pressure which is given by $p = \rho g H$, where p is the hydrostatic pressure, ρ is the density of the fluid and H the depth from the free surface. This test is represented in Figure 5.2 with the initial parameters $B = 1,820,000$, $\alpha = 0.5$ and a small smoothing length $\kappa h = 1.2s$.

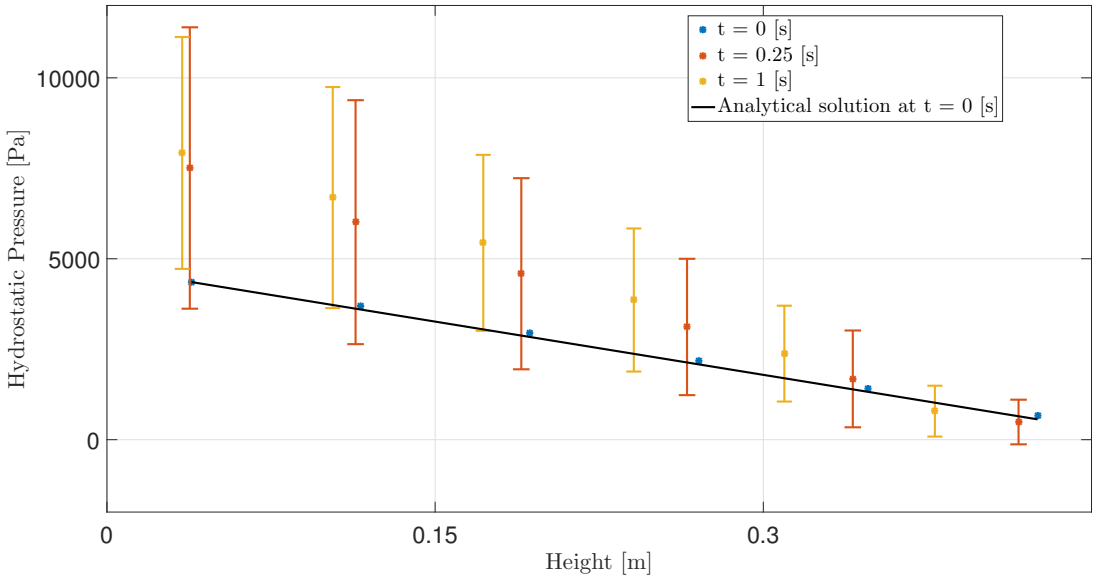


Figure 5.2: Hydrostatic pressure.

As can be seen, there is a huge dispersion of the pressure, especially at the bottom of the pool. This is due to a too small smoothing length. It has to be increased to $\kappa h = 1.2\kappa s$ as suggested in the literature, with κ depending on the kernel and equal to 2 here.

After this modification, the results are much more in agreement with the analytical solution as shown in Figures 5.3 and 5.4 by using the same parameters. The comparison is done between the simulation with an initialized hydrostatic pressure and the one without the initialized hydrostatic pressure. For both cases, the hydrostatic pressure stabilize quite fast in the same amount of time. Only 0.25 [s] is required to reach the equilibrium configuration. The initial pressure set by the SPH code is close to the one reached in the equilibrium configuration and the dispersion is reduced by using a larger smoothing length.

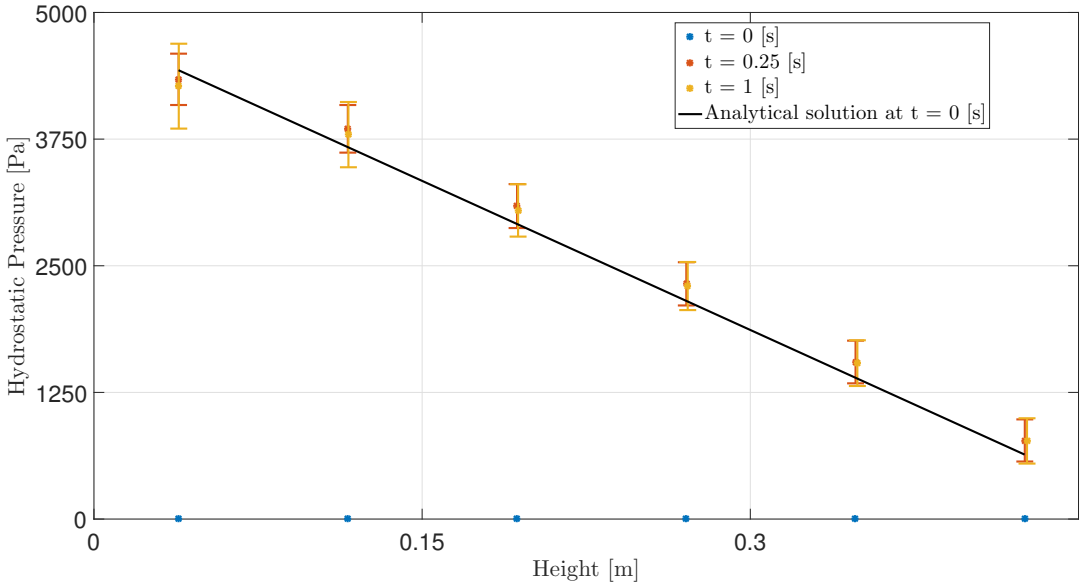


Figure 5.3: Hydrostatic pressure without initialized hydrostatic pressure.

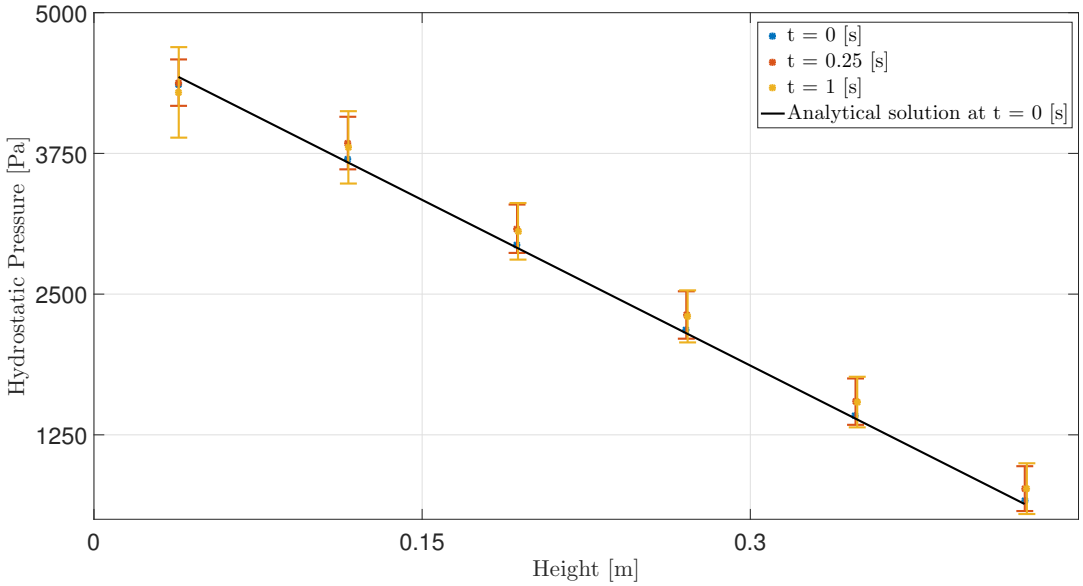


Figure 5.4: Hydrostatic pressure with initialized hydrostatic pressure.

In Figure 5.5, the hydrostatic pressure is computed at the middle of the depth as a function of time. This graph highlights the oscillatory behaviour of the stationary fluid test case to reach an equilibrium configuration. Several parameters were tested in order to validate their influence on the system. Accordingly to the results, the frequency and the damping effect are given in Table 5.1 for all simulations computed in Figure 5.5.

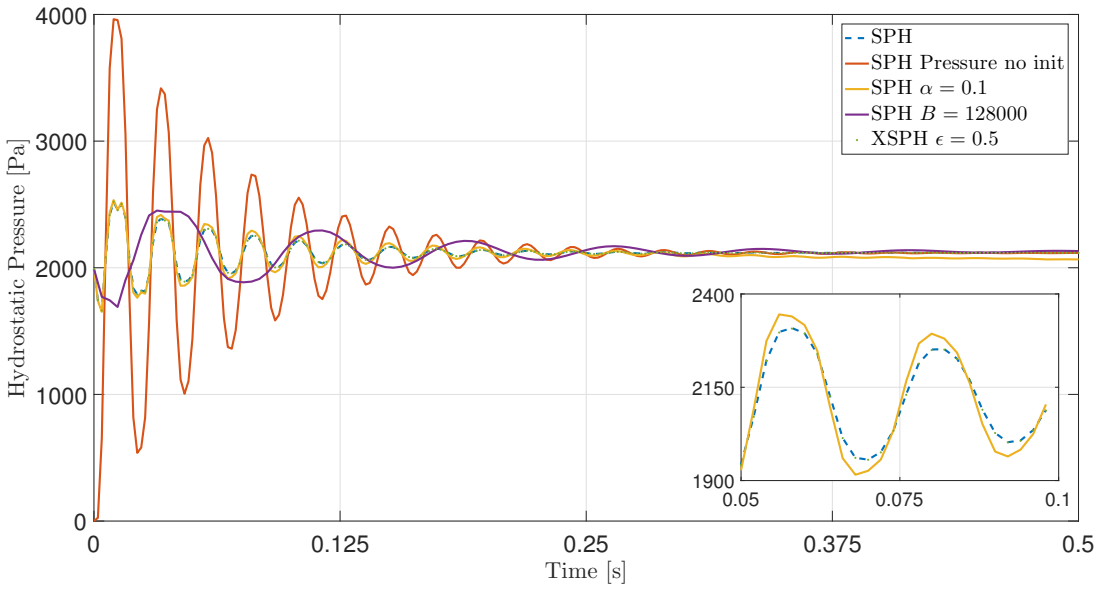


Figure 5.5: Hydrostatic pressure at middle depth for different change in parameter (with a zoom in).

The situation can be modelled by a damped oscillator. The behaviour captured in Figure 5.5 of the damping effect is a typical logarithmic decay Δ that can be computed as follows:

$$\Delta = \log \frac{P_n}{P_{n+1}} = \frac{2\pi\epsilon}{\sqrt{1-\epsilon^2}}, \tag{5.2}$$

where P_n is a local maximum (peak) in pressure, P_{n+1} the following local maximum and ϵ the damping coefficient. The time interval Δt between those two successive peaks is simply given by:

$$\Delta t = \frac{2\pi}{\omega_d} = \frac{2\pi}{\omega_0\sqrt{1-\epsilon^2}}, \tag{5.3}$$

where, ω_d is the natural frequency of the damping oscillator, ω_0 is the natural frequency of the non-damping oscillator and ϵ the damping coefficient. From equations 5.2 and 5.3 one can directly express the damping coefficient ϵ of the simulation:

$$\epsilon = \sqrt{\frac{\Delta^2}{\Delta^2 + 4\pi^2}}. \tag{5.4}$$

	SPH	SPH No Init	$\alpha = 0.1$	$B = 182000$	XSPH
$\omega_d/(2\pi)$ [Hz]	45	43	46	13	45
ϵ [-]	0.0013	0.0039	0.0014	0.0022	0.0013

Table 5.1: Frequency and damping coefficient of the oscillatory movement (Figure 5.5).

As explained before, the time it takes to the system to reach the equilibrium configuration is globally unchanged when the pressure is not initialized at the theoretical value, *i.e.*, when it is initialized to zero everywhere. However, the amplitude of the oscillation are much bigger when the

pressure starts from zero.

The parameter α is linked to the viscous interactions between particles. When α is diminished to 0.1, the viscous dissipation effect is reduced. The velocity of particles are a little bit higher in magnitude and so the amplitude of the oscillations in the pressure are a little bit higher.

When the compressibility parameter is reduced the frequency of oscillations is also reduced. This is expected because it plays the role of a restoring force.

Finally, the XSPH method has been tested on this basic case study. The parameter ϵ was set to 0.25 and the other parameters were the same as the initial ones. One can see that the XSPH method does not affect the results when the fluid is stationary and when particles velocity are very small.

To sup up, the different physical phenomena involved in this test are quite well represented. A change in some parameters can influence the dynamics of the system and therefore can be tuned to reach the desired behaviour.

5.3 Crashed cube

The test is made up of a free fluid cube above a rigid plate. The free cube falls and spreads over the plate. Different values of the compressibility B have been tested. Some snapshots of the solution for $B = 12.8 \times 10^6$ [kg/ms^2] are presented in Figure 5.6. The fluid particles do not go through the rigid plate. The chosen value of the compressibility B has the same order of magnitude of that of water. If B is chosen bigger, a smaller time step should be imposed as the wave propagation is faster inside the fluid. On the contrary, when B is smaller, a larger time step can be used.

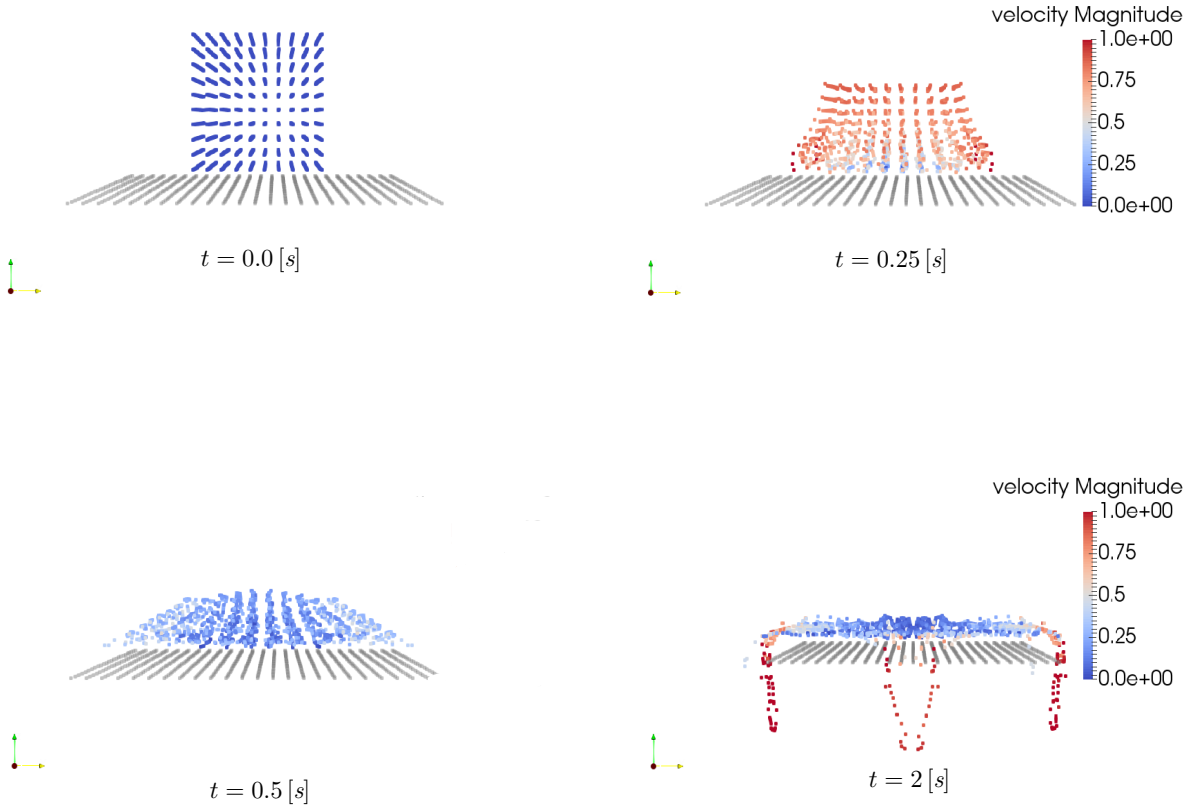


Figure 5.6: Fluid cube spreading of a rigid plate (xz-plane projection).

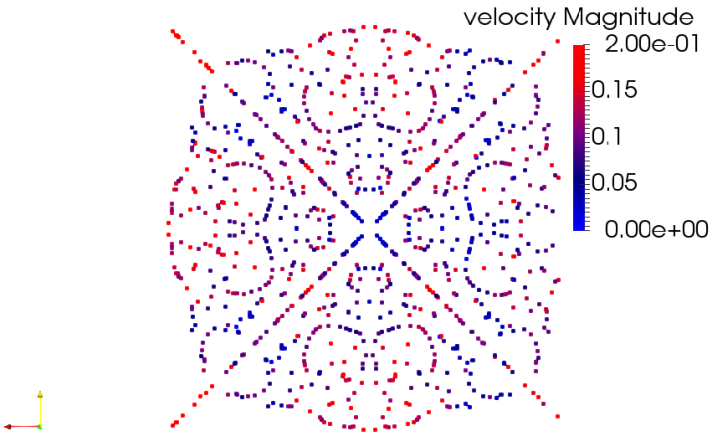


Figure 5.7: Symmetry of the numerical solution (xy-plane projection) at $T = 1$ [s].

As can be noticed in Figure 5.7, the numerical solution is perfectly symmetric. When the position

of each particle in the initial configuration is moved randomly (of 2% of s maximum), the solution becomes more realistic and the artificial symmetry is broken as can be seen in Figure 5.8. One can observe that the dispersion of particles is more homogeneous and more diffuse. Thus, one can observe more easily the evolution of the velocity with respect to the distance with the center of the cube.

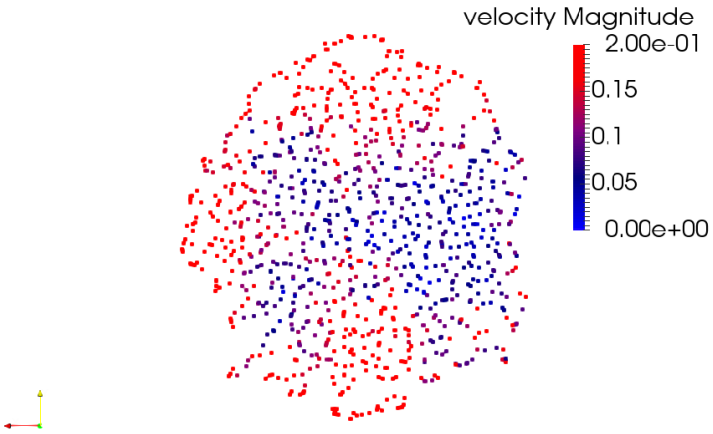
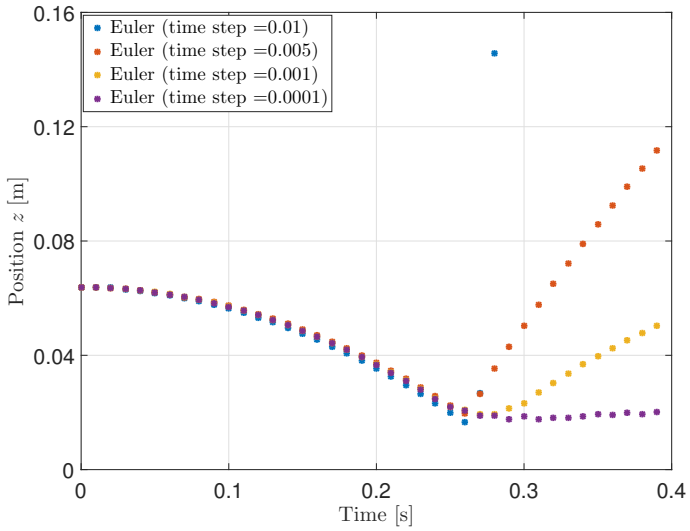


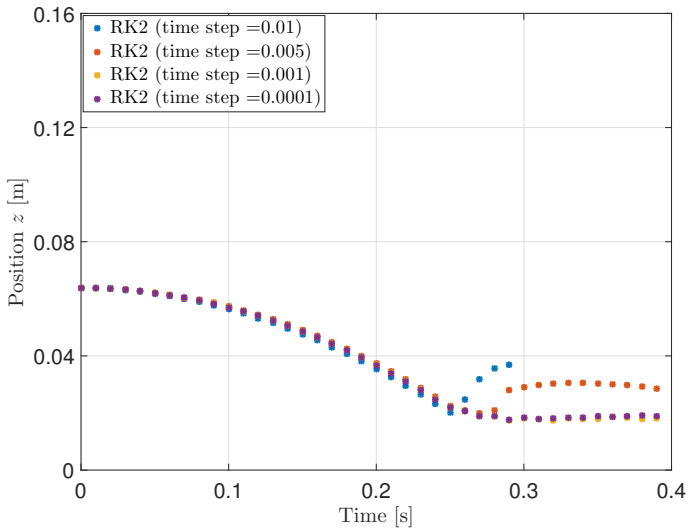
Figure 5.8: Symmetry of the numerical solution (xy-plane projection) at $T = 1$ [s] with $2\%s$ randomness for initial positions.

This crashed cube test case is a well chosen case to compare the two integration schemes. The crashed cube test case represents the worst case, *i.e* when the speed of the fluid particles is instantaneously set to zero. Indeed, when the particles enter in collision with the fixed plane, their speed along the z axis must be equal to zero. Moreover, their z position stays the same if rebounds are avoided by setting the α coefficient to a nonzero value. These observations were found by setting a very small step time. Due to the consistency of the method, this accurate simulation is taken as a reference.

In order to simplify the problem and focus on the speed of particles when the crash happens, the falling cube is replaced by only four particles fairly spaced in the same plane (xy-plane). In the following graphs, the speed is averaged in those four particles.



(a) Euler.



(b) Runge-Kutta 2.

Figure 5.9: Numerical scheme comparison on a crash cube test case.

One can see in Figure 5.9 that the correct solution is reached with a larger step time when using the Runge-Kutta numerical scheme compared to Euler that tends to be unstable when using larger step time (the method diverged for $k = 0.01$). However, for a given step time, the CPU time is not the same for both integrated scheme. In fact, when using Runge-Kutta integration, the continuity and momentum equation are computed twice which rises the computation time.

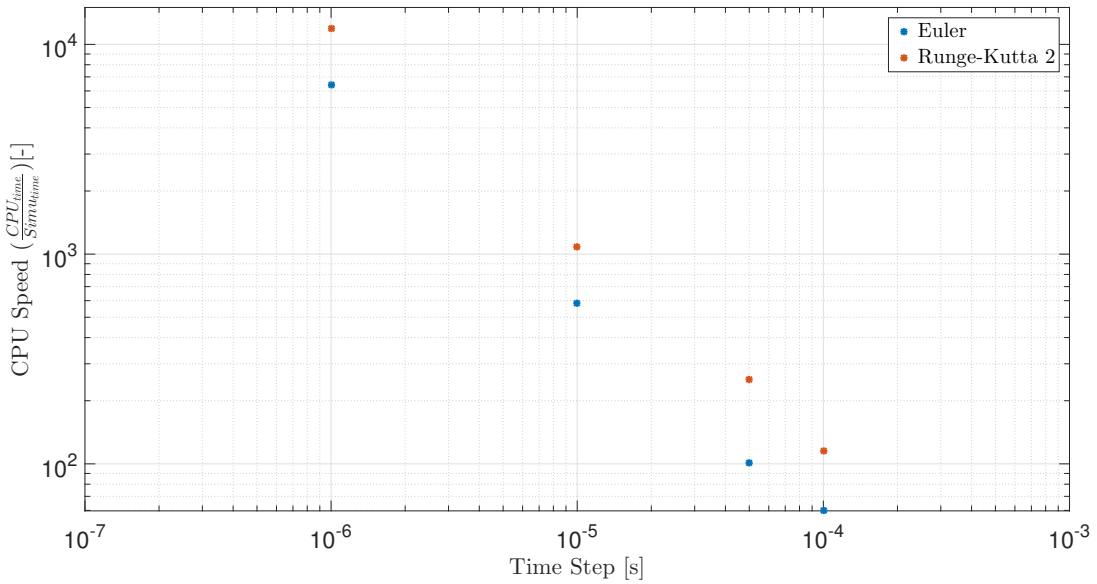


Figure 5.10: CPU Time comparison between Euler and Runge-Kutta numerical scheme.

As can be seen in Figure 5.10, the computation time when using Runge-Kutta scheme is nearly doubled. For example, when using a step time $k = 10^{-6}$, the CPU takes 3h20' to compute one second of simulation when using Runge-kutta integration method. Only 1h45' is required for the Euler scheme to compute the solution with the same time step. This corresponds to an increase of 90% CPU time.

Finally, it can be concluded that an order-two Runge-Kutta numerical scheme is a good choice in this type of application. Indeed, the gravity effect on free falling particles is well represented and the time step can be increased while keeping an excellent accuracy on results.

5.4 Dam break

A usual geometry for SPH models validation is the dam break case. It has already been introduced in section 4.2. The considered geometry for the simulations is the same as the one in Figure 4.5.

A first simulation has been run on NIC4 with a total number of particles of 56,721. The important parameters used for the simulation are given in table 5.2. Some pictures of the simulation results are given in Figure B.10 in appendix B.

The time evolution $Y(t)$ of the wave front in the y-direction is compared with experimental measurements (taken from [29]) for several values of the physical parameters. The position of the wave front which is computed is the one of the particle which has the maximal position's y-component. The time evolution of the y-component of the speed of the wave-front is also analyzed. The considered speed is the mean y-component of the velocity for all particles within a frontal part of the fluid. If L is the extent of the fluid volume along the y-axis, the frontal part contains all the particles whose

Parameter	Value	Units	Simulation information	
κh	0.48	[m]	Free particles	32,175
c	35	[m/s]	Fixed particles	24,546
α	0.5	[-]	Number of nodes	10
β	0	[-]	CPU per node	1
B	175,000	[kg/m^3]	Run time	28'11"
T	6	[s]		
k	0.0005	[s]		
Kernel	Quintic			

Table 5.2: Simulation parameters and information for the dam break case.

y-component position is larger than $0.9L$.

It could be interesting to look at the influence of different physical parameters: the viscosity α, the speed of sound c_0 (through the compressibility B for example) and the smoothing length h.

As can be seen in Figures 5.11 and 5.12, the results are slightly different when the viscosity varies. The speed of the wave front increases when the viscosity decreases. This can easily be explained by the fact that lower viscosity induces less friction so that the wave front can propagates with a higher speed. The wave front collides the end of the enclosing box after about 1 [s], which explains the sharp decrease of its velocity. In all cases, Figure 5.11 demonstrates that the numerical results of the simulation are in a good agreement with the experiment.

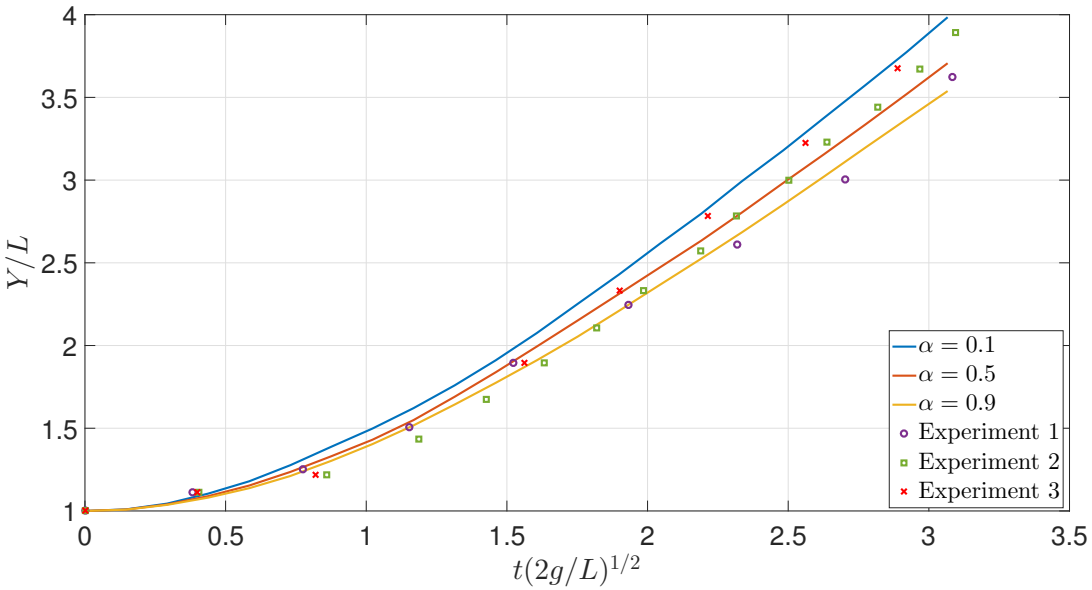


Figure 5.11: Time evolution of the wave-front for an increasing viscosity. L is the initial length of the fluid volume along the y-axis.

The speed of sound does not influence importantly the results as can be seen in Figure 5.13.

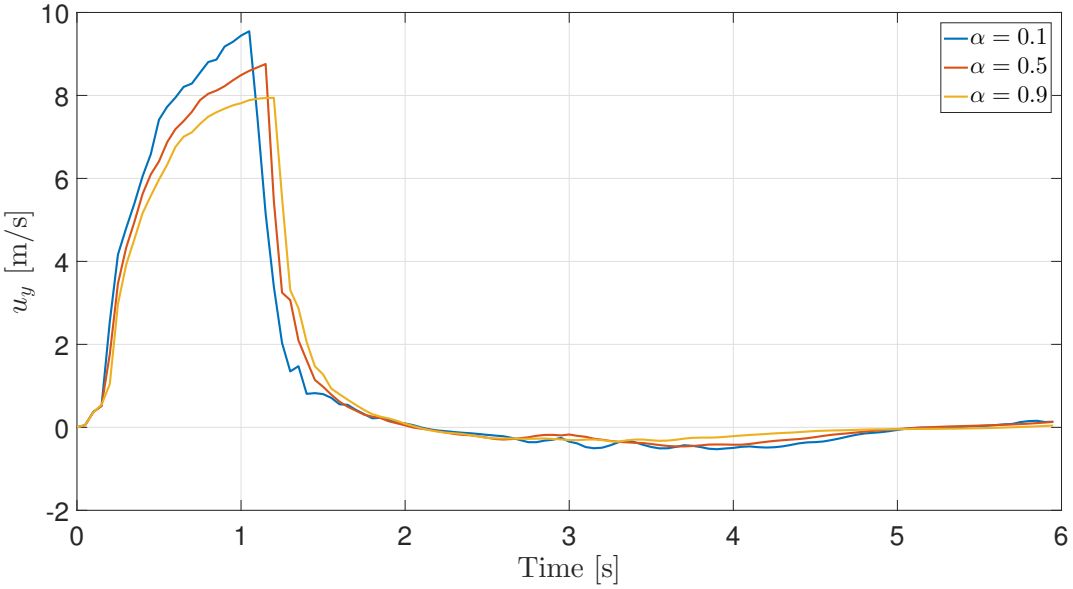


Figure 5.12: Time evolution of the y-component of the speed of the wave-front for an increasing viscosity.

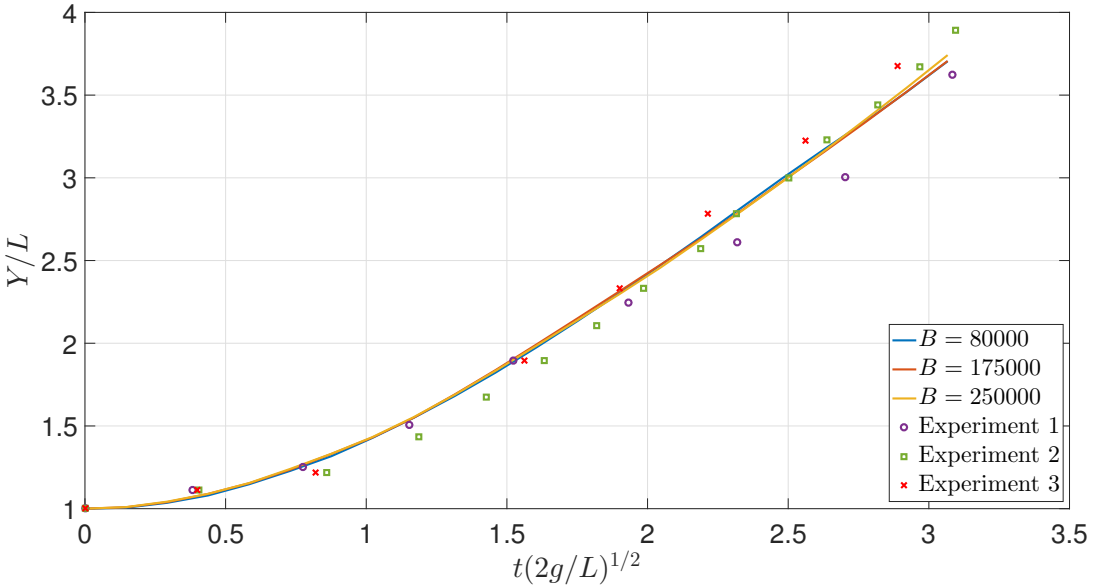


Figure 5.13: Evolution of the wave-front over the time for different values of the speed of sound. L is the initial length.

As already mentioned, the smoothing length h is a crucial parameter when problems are modelled with SPH. The smoothing length will determine the range of interaction between the particles. Imposing a higher smoothing length means that the influence of further particles is taken into ac-

count. A minimum smoothing length is required to obtain physical results. However, the higher the smoothing length, the higher the computational cost, this is illustrated by table 5.3.

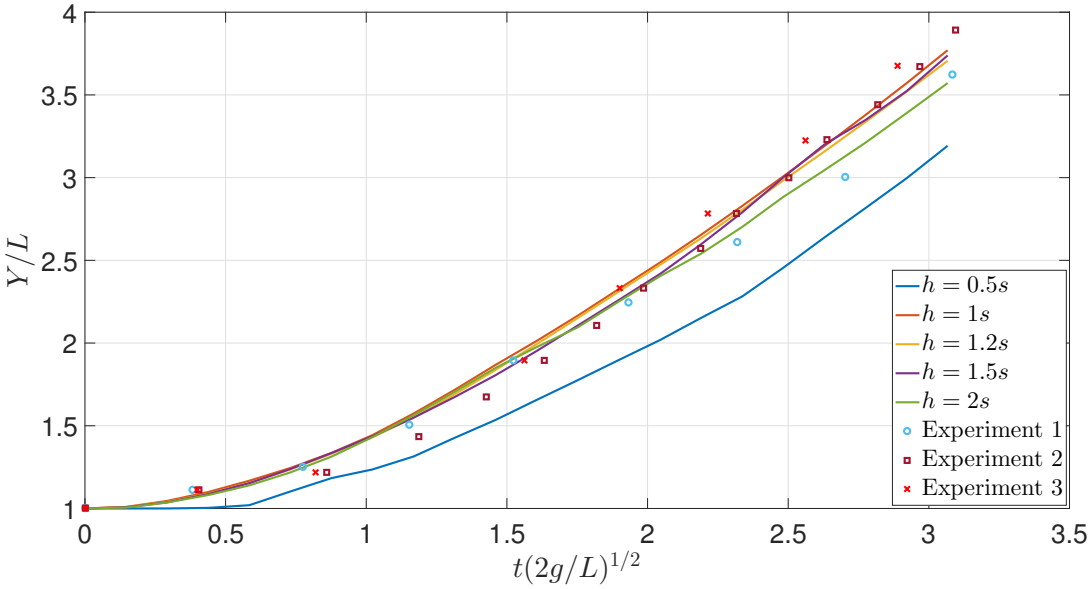


Figure 5.14: Evolution of the wave front over the time for different values of the smoothing length. L is the initial length.

Smoothing length	Run time
$h = 0.5s$	6'36'
$h = 1s$	18'15"
$h = 1.2s$	28'35'
$h = 1.5s$	1h20'19"
$h = 2s$	2h12'43"

Table 5.3: Run time for several smoothing length.

5.5 Wave propagation

5.5.1 Waves in a constant depth profile

Theory of linear inviscid gravity waves predicts a phase propagation speed

$$c = \sqrt{\frac{g\lambda}{2\pi} \tanh \frac{2\pi H}{\lambda}}, \tag{5.5}$$

where λ is the wavelength of the surface wave and H is the depth of the water. In the shallow water case, *i.e.*, when $\lambda \gg 2\pi H$, the propagation is not dispersive and the speed is given by $c = \sqrt{gH}$. Otherwise, the wavelength influences the speed.

To validate the SPH model and estimate its ability to reproduce accurately gravity waves, several simulations have been run for various parameters and water depth. The general geometry is depicted in Figure 5.15. A moving plate excites the fluid with a period $T = 5$ [s] and an amplitude of 2 [m]. Four values of H have been considered, with a viscosity parameter $\alpha = 0.01$ and $\alpha = 0.1$. The chosen speed of sound was $c_0 = 10\sqrt{gH}$. The number of particles ranges from 78,000 to 223,000. The wavelength is measured on the results and the wave speed is calculated by looking at the wave maximum height x-position evolution, once a stable flow is obtained. Snapshots of the results and illustration of the speed measurement are represented in Figures 5.16 and 5.16. A comparison between the simulation speeds c_{simu} and the theoretical predictions c_{th} is given in table 5.4.

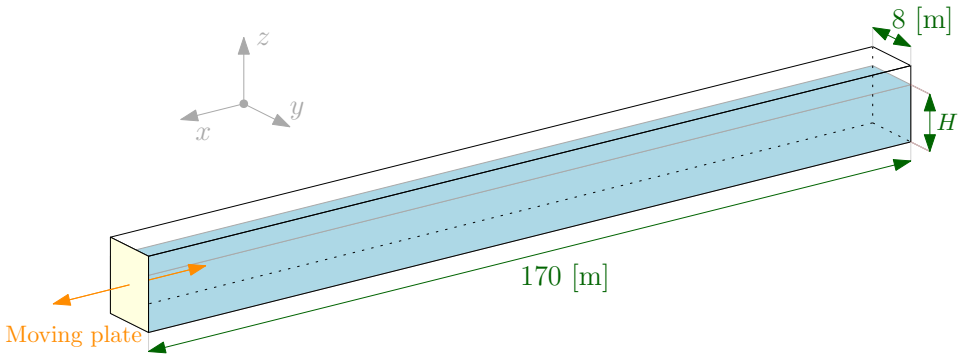


Figure 5.15: Considered geometry for the wave propagation study.

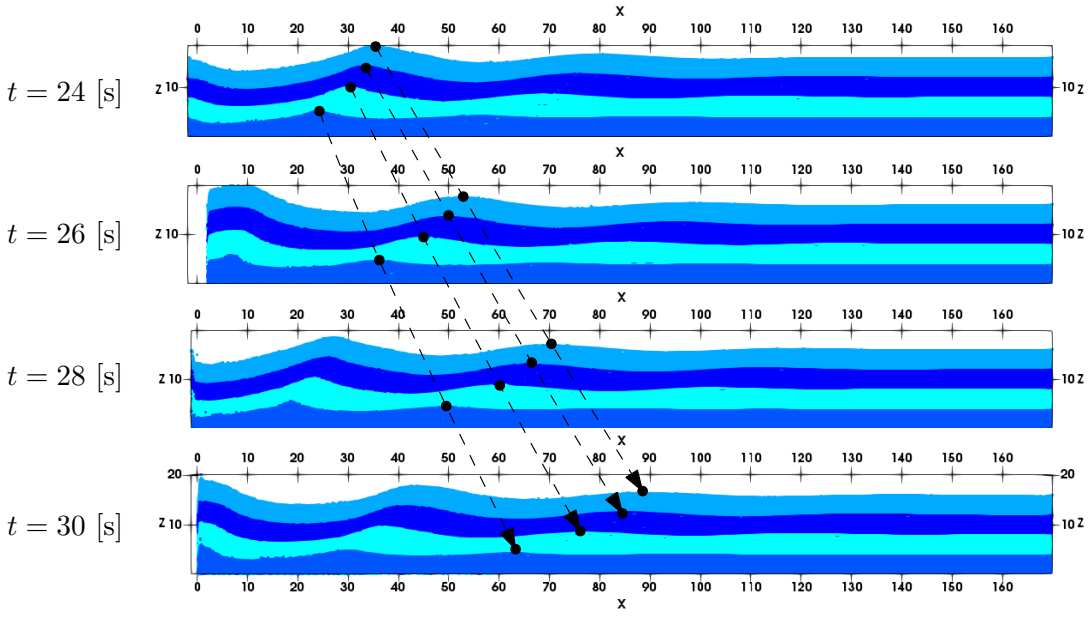


Figure 5.16: Results for $\alpha = 0.1$ [-] (xz-plane projection). Colors correspond to simulations with different depth H.

The wave speed is always slightly larger than predicted, with less than 10% of relative difference. The measured speeds for the two values of the viscosity coefficient α are close to each other.

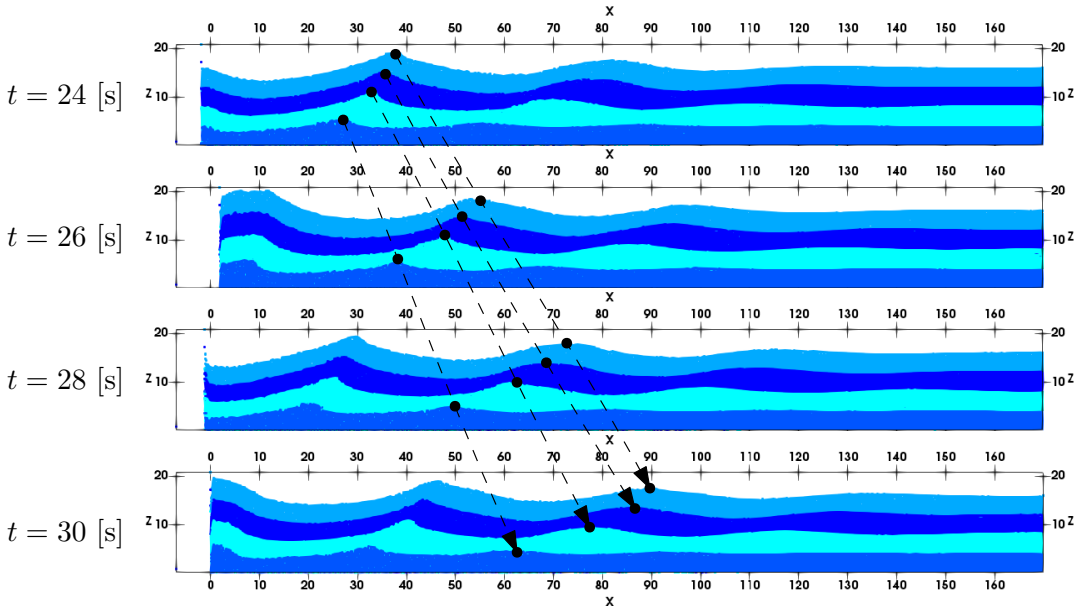


Figure 5.17: Results for $\alpha = 0.01$ [-] (xz-plane projection). Colors correspond to simulations with different depth H.

H [m]	λ [m]	c_{simu} [m/s] ($\alpha = 0.01$)	c_{simu} [m/s] ($\alpha = 0.1$)	c_{th} [m/s]
4	32	6.17	6.33	5.72
8	36	7.67	7.67	7.05
12	42	8.50	8.67	7.84
16	43	8.67	8.83	8.11

Table 5.4: Wave speed comparison.

The difference between the two simulations lies in the dissipation, the wave amplitude decays much faster for larger values of the viscosity, which is expected.

The XSPH method has also been tested on the same geometry in order to estimate its influence. A comparison is given in Figure 5.18. As expected, the averaging for the position update results in a smoothing of the free surface. However, the effect is limited. Both methods provide similar global results.

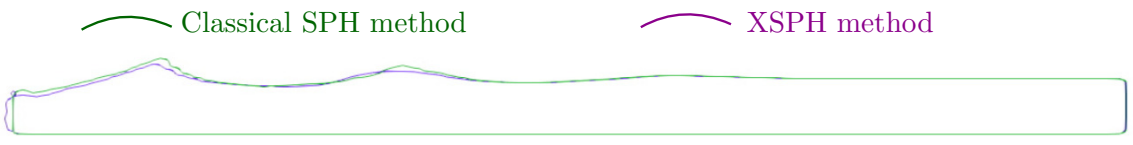


Figure 5.18: Comparison between the classical SPH results and the XSPH results ($\varepsilon = 0.5$) for $\alpha = 0.1$. Contour of the free surface.

Different values of the speed of sound have been tested. In agreement to what has already been

observed in Figure 5.13, its value does not have a large influence. Provided that the speed of sound is large enough, the wave propagation speed remains unchanged.

5.5.2 Waves in a variable depth profile

The tests of the previous section were basically two-dimensional tests. The flow was nearly y-independent. To exploit fully the possibilities of the code, a 3D geometry can be build. To model a variable depth profile, the geometry of Figure 5.19 is considered. The depth distribution is constant and equal to 8 [m] in rectangle $abcd$ and a bilinear interpolation is performed inside rectangle $cdef$ where c, d and e are 8 [m] below the water level and f is 2 [m] above it. A moving plate excites sinusoidally the fluid near the constant depth region with a period $T = 8$ [s] and an amplitude of 2 [m].

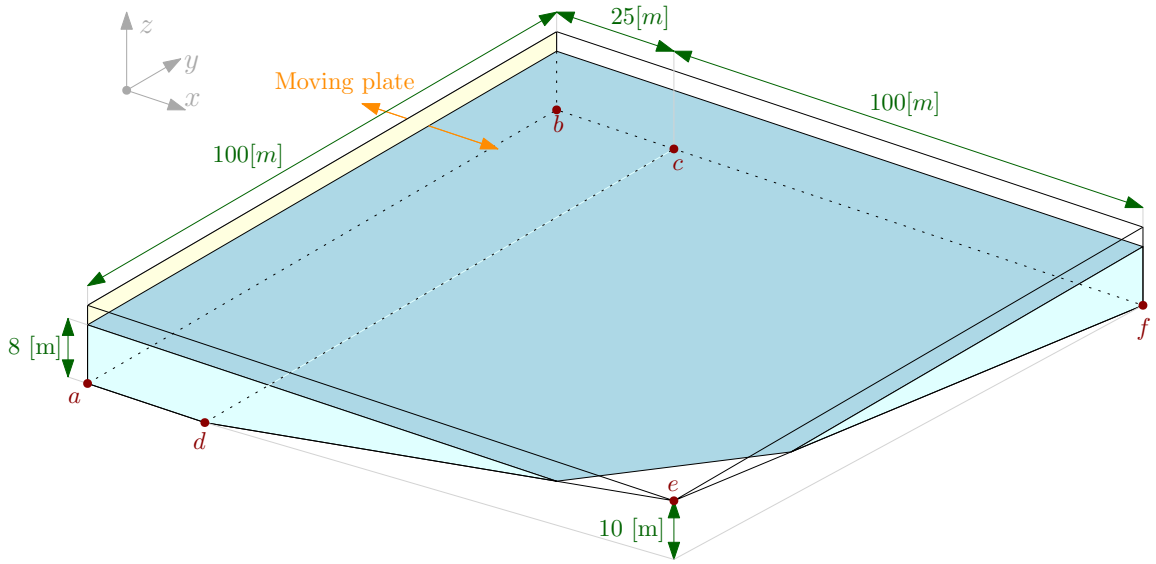


Figure 5.19: Variable depth profile geometry.

Information about the simulation are given in table 5.5. Snapshots of the results are presented in Figure B.11 in appendix B. The wave breaks when the depth decreases.

Parameter	Value	Units	Simulation information	
κh	1	[m]	Free particles	700,564
c	35	[m/s]	Fixed particles	130,320
α	0.01	[-]	Moving particles	11,600
β	0	[-]	Number of nodes	40
B	$175,000$	[kg/m^3]	CPU per node	2
T	50	[s]	Run time	6h03'
k	0.001	[s]		
Kernel	Quintic			

Table 5.5: Simulation parameters and information for the wave break test.

5.5.3 Waves in a realistic bathymetric profile

In the scope of coastal protection, it is interesting to be able to simulate underwater landslides, the propagation of the resulting waves, and their impact when breaking on terrestrial or offshore infrastructures.

However, the only SPH formulation that has been implemented in this project is not sufficient for several reasons. On the one hand, it is shown in section 5.5.1 and 5.5.2 that the dissipation is too high to enable to propagation of waves over long distance and so impossible over thousands of kilometers. To decrease the dissipation, one could decrease the value of the α parameter (section 2.2) but it would lead to unphysicial fluid instabilities. On the other hand, wave propagation in the ocean is a problem with two very different scales, the spatial extent of the ocean and the height of fluid. The former being of the order of thousands of kilometers while the latter is only a few kilometers. The size of the particles is however dictated by the smallest dimension and that thus would lead to an excessively large number of particles to have a realistic simulation. For example, considering the simulation of a tsunami striking the Sri Lanka, the typical domain size [27] is given by 1200 [km] $\times$ 2000 [km] $= 2.4 \times 10^6$ [km^2]. Considering a depth of 1 [km] and particles of 10 [m^3], which is already much too large to have an accurate description of the impact of infrastructure, the simulation would contain 2.4×10^{14} particles. This number is clearly not realistic.

Consequently, other approaches have to be considered. Altomare *et. al.* [35] proposed to use an hybrid model coupling SPH and a finite difference method. The former is used to model the breaking waves on the shore while the latter is used to simulate the wave from the far ocean to the shore. Using the algorithm from the *High performance scientific computing* project [27], a similar approach could have been implemented. This is however beyond the scope of this project.

Conclusion and further work

In this project, the Smoothed-Particle Hydrodynamics method, a lagrangian, meshfree method has been studied and implemented in order to simulate problems involving free surface flows. The code allows the definition of arbitrary geometries with fixed or moving boundaries. An important aspect of the implementation is that it is fully parallel. Both shared and distributed memory parallelization strategies have been exploited.

First, the characteristics of the SPH method as well as the general context in which this method was born have been introduced. A few applications in different fields involving SPH and other numerical techniques already existing have been presented. Then, the method has been introduced in a mathematical point of view. The Navier-Stokes equations have been discretized in the framework of the SPH method. Two integration schemes have been implemented and compared: the explicit Euler scheme and a two-step Runge-Kutta integration scheme. Other physical and numerical considerations have been described (artificial viscosity, adaptive time step, smoothing length, etc.). The third part of the project consisted in describing the implementation strategy. A sequential version of the algorithm has first been written and then parallelized using both MPI and OpenMP. The very first part of the implementation consisted in the neighbor search. For this work, the linked-list algorithm has been chosen. The next part was devoted to the performance analysis of sequential and parallel runs. For the sequential runs, it has been observed that the running time increases almost linearly. The performance of the parallel code has been studied in a strong and a weak scaling analysis. The speed up is quite close to the ideal curve for a low number of processors but tends to drift when the number of processors increases. In the last part, several test cases have been designed in order to validate the implemented algorithm. As a conclusion, the results were in a good agreement with what would be expected both visually and physically. The main important conclusions that arise from these test cases are the followings. The Runge-Kutta integration scheme provides better results and allows to use a larger time step despite the increased computational cost of the scheme compared to the explicit Euler scheme. The choice of the smoothing length is a crucial element. A compromise between proper physical results and reasonable computation cost should be found out. Due to the large dissipation, the simulation of the wave propagation over a very large domain remains an issue and a solution has been proposed.

Some challenges are still present and should be handled in some future work. In order to enhance the scaling performance in non homogeneous fluid distributions, a dynamic domain sharing should be considered.

The implementation offers a wide range of possibilities thanks to the moving boundaries features. However, these could still be extended. Fluid sources and sinks could be interesting to model continuous flows. Moreover, additional physics could be added to the current formulation. For ex-

ample, surface tension effects could be investigated by implementing a free surface detection method (detection through the number of neighbors that is lower than in the bulk) and a local curvature calculation for determining the local Laplace pressure. Moreover, physical viscosity models could be considered based on (non-)Newtonian constitutive laws. Fluid-structure interactions are also possible to model with the SPH method.

Appendix A

Code use

A.1 Compilation and running

A.1.1 Compilation

Windows A compiler (g++ MinGw) must be installed or an IDE with a build-in compiler can be used (*e.g.* Visual Studio). Also, the Cmake software must be installed. Compilation on Windows can be different, depending on the used IDE. Here follows some command lines for compilation:

```
mkdir build
cd ./build/
set PATH=%PATH:D:/Programy/Git/bin=%
cmake -G "MinGW Makefiles" ..
mingw32-make
```

Linux or Mac OS The Cmake software must be installed and a g++ compiler must be available.

```
mkdir build
cd build
cmake ..
make
```

On Clusters The code can be launched on clusters to benefit from the MPI implementation of the code and speed up the computation time of the simulations. All simulations in this project, that was composed of a large number of particles, were launched on *Nic4* (a cluster hosted at the University of Liège (SEGI facility)). It features 128 computer nodes with two 8-cores Intel E5-2650 processors at 2.0 GHz and 64 GB of RAM (4GB/core). Some modules should be loaded before compiling:

```
module load slurm
module load openmpi
module load cmake
module load gcc

mkdir build
cd build
```

```
cmake ..
make
```

The following command could also be needed to set the g++ compiler by default:

```
export CXX=g++
```

A.1.2 Start a simulation

Create a new directory to store data

```
cd build
mkdir Results
mkdir ./Results/<myName>
```

The field <myName> should be replaced by an user chosen name for the simulation.

Launch a simulation (command line)

```
mpirun -np <nbProc> ./sph <pathToParameterFile>
        <pathToGeometryFile> <name>
```

The fields <nbProc> and <name> should be replaced by the desired number of processors and the desired name for the output files while <pathToParameterFile> and <pathToGeometryFile> should be replaced by the path to the parameter and the geometry file.

Launch a new simulation (bash script) An example file of a bash script is given here below

```
#!/bin/bash
#
#SBATCH --job-name=wetFloor
#SBATCH --mail-user=karl.ajumide@student.ulg.ac.be
#SBATCH --mail-type=ALL
#SBATCH --output=wetFloor.txt
#
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=4
#SBATCH --time=100:00
#SBATCH --mem-per-cpu=400

Para="../Playgrounds/wetFloor.para"
Geom="../Playgrounds/wetFloor.geom"
TestName="wetFloor"


module load openmpi/1.6.4/gcc-4.9.2
module load cmake/3.5.2
export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK
mpirun sph \$Para \$Geom \$TestName
```

The number of processors allocated to the job are fixed with `ntasks`. The number of threads allocated to the job are specified with `cpus-per-task`. The name of the input files as well as the output file name also have to be entered. After saving the file, it can be launched with

```
sbatch scriptTest.sh
```

A.2 Input parameter file

A parameter input file must contain the following information:

- a parameter file identifier '#`param`',
- the specification of each parameter given in Table A.1 in the same order and in the form '`variable name = value`',
- a parameter file end identifier '#`END_F`'.

A.3 Input geometry file

A geometry input file is composed of at least three parts. First, a geometry file identifier

- #`GEOM1`,

followed by the definition of the domain $\Omega = [l_x; u_x] \times [l_y; u_y] \times [l_z; u_z]$ as

- 'u_α = value', $\alpha = x, y, z$,
- 'l_α = value', $\alpha = x, y, z$,

then the definition of the particles through different building blocks *i.e* rectangular parallelepiped and/or bathymetries, which are detailled in the following sections and finally, the geometry file end identifier

- '# `END_G`'.

A.3.1 Rectangular parallelepiped

A rectangular parallelepiped block of particles can be created after the definition of the domain writing

- the parallelepiped identifier '#`brick`',
- the specification of each parameter given in Table A.2 in the same order and in the form '`variable name = value`'.

The brick orientation θ_x, θ_y and θ_z corresponds to the Euler angle of an extrinsic rotation of axis $\hat{x}$, $\hat{y}$ and $\hat{z}$.
The particles in the brick are spaced from each other of a distance s and from the faces of the box of a distance $s/2$ in each direction.
Parameters after the second horizontal line in Table A.2 are only relevant when declaring a moving block. Their aim is to control the position $\boldsymbol{r}(t)$ (and thus the velocity $\dot{\boldsymbol{r}}$) of the moving particles. A description of the evolution of $\boldsymbol{r}(t)$ according to the chosen law is given below.

Parameter	Symbol	Typical value	Units	Reference
Kernel type	/	$0 \rightarrow$ Gaussian $1 \rightarrow$ Bell shaped $2 \rightarrow$ Cubic spline $3 \rightarrow$ Quadratic $4 \rightarrow$ Quintic $5 \rightarrow$ Quintic spline	/	Section 2.5.1.
Smoothing length	κh	0.06	[m]	Section 2.5.2
Integration method	/	$0 \rightarrow$ Euler $1 \rightarrow$ Runge-Kutta	/	Section 2.4.
R.-K. parameter	θ	0.5	/	Section 2.4.
Adaptive time step	/	$0 \rightarrow$ enabled $1 \rightarrow$ disabled	/	Section 2.4.3.
Initial time step	k	10^{-5} (Euler)	[s]	Section 5.3
Simulation duration	T	1	[s]	/
Reference density	ρ_0	10^3	[kg/m^3]	/
Density initialization	/	$0 \rightarrow$ hydrostatic $1 \rightarrow$ homogeneous	/	Section 2.3.
Compressibility parameter	B	$c_0^2\rho_0/\gamma$	[kg/ms^2]	Section 2.3.
Weakly compressible parameter	γ	7	[-]	Section 2.3.
Gravity	g	9.81	[m/s^2]	/.
Speed of sound	c_0	$10\sqrt{gH}$	[m/s]	Section 2.3.
Viscosity model	/	$0 \rightarrow$ artificial viscosity	/	Section 2.2.
Viscosity parameter	α	0.1	[-]	Section 2.2.
Shock viscosity parameter	β	0	[-]	Section 2.2.
XSPH parameter	ε	0.25	[-]	Section 2.1.4.
Equation of state	/	$0 \rightarrow$ weakly compressible $1 \rightarrow$ perfect gas	/	Section 2.3.
Molar mass	M	18.01 (H_2O)	[gr/mol]	/
Temperature	T	293.15	[K]	/
Mass initialization	/	$0 \rightarrow$ Violeau	/	Section 2.1.2.
Output file interval	/	10^{-3}	[s]	/
Matlab output file	/	$0 \rightarrow$ disabled $1 \rightarrow$ enabled	/	Section A.4.
ParaView output file	/	$0 \rightarrow$ disabled $1 \rightarrow$ all (1 file) $2 \rightarrow$ only free (1 file) $3 \rightarrow$ only bound. (1 file) $4 \rightarrow$ all (2 files)	/	Section A.4.

Table A.1: Input parameter list.

Uniform rectilinear motion (linear)

$$\boldsymbol{r}(t) = \boldsymbol{r}(0) + A\boldsymbol{m}\, t. \tag{A.1}$$

Parameter	Symbol	Value	Units	Reference
Particle type	/	$0 \rightarrow$ Free $1 \rightarrow$ Fixed $2 \rightarrow$ Moving	/	Section 2.6.
Particle spacing	s	/	[m]	Section 2.1.2.
Brick origin	x, y, z	/	[m]	/.
Brick dimensions	L, W, H	/	[m]	/.
Brick orientation	θ_x, θ_y, θ_z	/	[$\circ$]	Section A.3.
Position law	/	$0 \rightarrow$ linear $1 \rightarrow$ sine $2 \rightarrow$ exponential $3 \rightarrow$ rotating	/	Section A.3.
Angle law	/	$0 \rightarrow$ linear $1 \rightarrow$ sine $2 \rightarrow$ exponential	/	Section A.3.
Moving period	τ	/	[s]	Section A.3.
Moving direction	$\boldsymbol{m} = (m_x, m_y, m_z)$	/	[-]	Section A.3.
Rotation center	$\boldsymbol{b} = (b_x, b_y, b_z)$	/	[m]	Section A.3.
Moving amplitude	A	/	[m]	Section A.3.

Table A.2: Input parameter list.

Sinusoidal motion (sine)

$$\boldsymbol{r}(t) = \boldsymbol{r}(0) + A\,\boldsymbol{m}\,\sin\left(\frac{2\pi}{\tau}t\right). \tag{A.2}$$

Exponential motion (exponential)

$$\boldsymbol{r}(t) = \boldsymbol{r}(0) + A\,\boldsymbol{m}\left(1 - \exp\left(-\frac{t}{\tau}\right)\right). \tag{A.3}$$

Rotational motions (rotation)

$$\boldsymbol{r}(t) = \boldsymbol{R}(\theta(t), \boldsymbol{m})\,(\boldsymbol{r}(0) - \boldsymbol{b}) + \boldsymbol{b}. \tag{A.4}$$

This movement actually corresponds to a rigid rotation of an angle θ around the straight line defined by the point $\boldsymbol{b}$ and the direction $\boldsymbol{m}$ as depicted in Figure A.1.

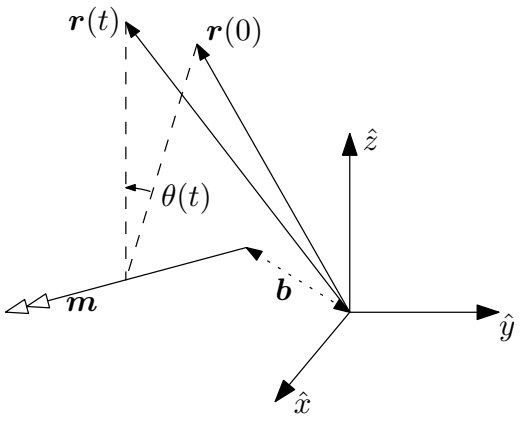


Figure A.1: Rotation of moving boundaries

The evolution of $\theta(t)$ can be chosen through the angle law and the possibilites are described here after.

Uniform circular motion (linear)

$$\theta(t) = \frac{2\pi}{\tau}t. \tag{A.5}$$

Oscillating motion (sine)

$$\theta(t) = A\sin\left(\frac{2\pi}{\tau}t\right). \tag{A.6}$$

Exponential rotation (exponential)

$$\theta(t) = A\left(1 - \exp\left(-\frac{t}{\tau}\right)\right). \tag{A.7}$$

A.3.2 Bathymetry

In the scope of this project, a bathymetry is defined as a rectangular surface that has the following parametric representation:

$$\boldsymbol{r}(x, y) = x\boldsymbol{e}_x + y\boldsymbol{e}_y + h(x, y)\boldsymbol{e}_z \quad \forall x, y \in [x_a, x_b] \times [y_a, y_b],$$

$\boldsymbol{r}$ being the position vector and x_a, x_b, y_a, y_b being the coordinates of the bathymetry corners.

A declaration of a bathymetry in the geometry file must contain the following informations:

- the bathymetric brick identifier '`#bathy`',
- the path to the bathymetric map file that stores the data relative to the profile to interpolate (the structure of this file is detailed later),
- the initial particle spacing s,

- an integer corresponding to the number of particle layers the bottom fixed profile has to be composed of,
- a reference height to which the height from the bathymetric map file are added,
- the height of the free surface (in the frame of reference of the simulation).

Bathymetric map file The bathymetric map file must be a binary file that has the following structure[1]:

x_a	x_b	y_a	y_b	N_x	N_y
$h_{(0,0)}$	$h_{(0,1)}$	$\dots$	$h_{(0,N_y)}$		
$h_{(1,0)}$	$h_{(1,1)}$	$\dots$	$h_{(1,N_y)}$		
$\vdots$	$\vdots$	$\ddots$			
$h_{(N_x,0)}$	$\dots$	$\dots$	$h_{(N_x,N_y)}$		

where x_a, x_b, y_a, and y_b are the bathymetry's corner in double precision, N_x and N_y are integers so that $N_x + 1$ and $N_y + 1$ represent the number of sampling points respectively along the x and y directions and $h_{i,j}$ is the depth in double precision such that

$$h_{(i,j)} = h(x_a + i\delta x, y_a + j\delta y),$$

with

$$\delta x = \frac{x_b - x_a}{N_x} \quad \text{and} \quad \delta x = \frac{y_b - y_a}{N_y}.$$

Interpolation Particles are spaced from one to another by a distance as close as possible to s. Consequently, they are generated at positions

$$\begin{aligned} &\boldsymbol{r} = (x_s, y_s, h(x_s, y_s)), \\ &\quad \text{with } x_s = x_a + s(1/2 + k) \quad \text{and} \quad y_s = y_a + s(1/2 + l), \\ &\qquad \forall k = 0, ..., n_x, \quad l = 0, ..., n_y, \\ &\qquad\quad \text{with } n_x = \lfloor (x_b - x_a)/s - 1 \rfloor \quad \text{and} \quad n_y = \lfloor (y_b - y_a)/s - 1 \rfloor. \end{aligned} \tag{A.8}$$

As these points do not necessarily match those given by the bathymetric map file, an interpolation has to be performed. As in [27], a bi-linear interpolation has been chosen. For each $\boldsymbol{r}$ given by Eq. (A.8), the coordinates x_{up}, x_{low}, y_{up}, y_{low} of the surrounding points in the bathymetric map file are found. Then $h(x_s(k), y_s(l))$ is approximated by

$$\begin{aligned} h(x_s(k), y_s(l)) \approx [&h(x_{low}, y_{low})(x_{up} - x_s)(y_{up} - y_s) \\ &+ h(x_{up}, y_{low})(x_s - x_{low})(y_{up} - y_s) \\ &+ h(x_{low}, y_{up})(x_{up} - x_s)(y_s - y_{low}) \\ &+ h(x_{up}, y_{up})(x_s - x_{low})(y_s - y_{low})]/(\delta x \delta y). \end{aligned}$$

Finally, free particles are generated until the free surface is reached.

[1]The structure has been taken from [27] for comparison purposes.

A.4 Output files

There are two types of output files that can be generated. The first one is built to be used with the software ParaView and the second one to be used with the software Matlab. Three parameters are available in the input file to select the output format:

- **writeInterval** It corresponds to the time interval at which an output file will be generated (the same parameter for both Matlab and ParaView output files).
- **matlab**
  - **0**: Disables output file for Matlab (.txt) analysis;
  - **1**: Enables output file for Matlab (.txt) analysis.
- **paraview**
  - **0**: Disables output file for ParaView (.vtp) visualization;
  - **1**: Enables output file for all particles for ParaView (.vtp) visualization.
  - **2**: Enables output file only for *free* particles for ParaView (.vtp) visualization.
  - **3**: Enables output file only for *moving* and *fixed* particles for ParaView (.vtp) visualization.
  - **4**: Enables output file for all particles for ParaView (.vtp) visualization and separate in two different files the *free* particles and the *moving+fixed* particles.

A.4.1 ParaView output file

The output files are built to work with the software ParaView which is a open-source, multi-platform data analysis and visualization application. The format `.vtp` is a compressed format to reduce the memory consumption and accelerate the transfer of results. ParaView was used in this project to analyze the behaviour of the fluid and generate all pictures of simulations available in this report.

A.4.2 Matlab output file

Another output file can be generated in a `.txt` format. This file is more readable and can be used to debug overflows, *NaN*, bad results, etc. However, they are not compressed and they can take a lot of memory especially when dealing with big simulations with a large number of particles. This file is composed of several parts:

- Information about the name of the simulation *i.e* input files used, computer ID, the date of the simulation, etc,
- Parameter used for the simulation,
- Data corresponding to all particles, ordered by types. Each row corresponds to one particle, and each column correspond to a physical field (scalar or component).

When using Matlab, this file can simply be loaded using the command

- `Data = importdata('mySimulation.txt').`

This will generate a structure. To access automatically to the data of this structure, the field `data` can be used. To access the text or variables above data, the field `textdata` can be used.

Appendix B

Additional tests and figures

B.1 Fluid mixing

To illustrate the possibilities of the implemented moving boundaries feature, the geometry of Figure B.1 has been considered. It consists of a rectangular fixed box, a central part that rotates counter-clockwise and a fluid volume that fills the box up to a height of 12 [m].

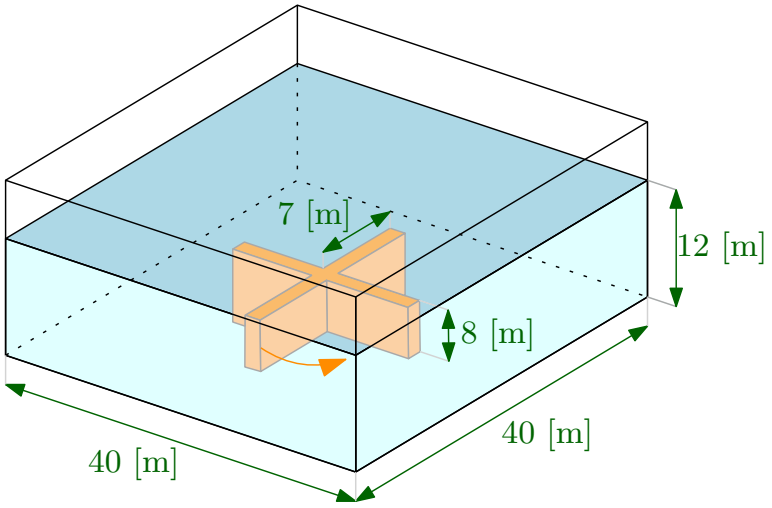


Figure B.1: Fluid mixing geometry.

A 80 [s] simulation (80,000 time steps) has been launched with 250,000 particles on 16 computer nodes with 4 CPU per node. The computation time was 2h20'. A bottom view of the results is presented in Figure B.2. Some streamlines are given in Figure B.3. The apparent crossings of the lines is due to the projection on the xy-plane, the flow is fully tridimensional and recirculations are observed. Another set of streamlines is represented in Figure B.4. The free surface is not flat, as shown in Figure B.5. It could be interesting to compare its shape to the analytical solution of a vortex.

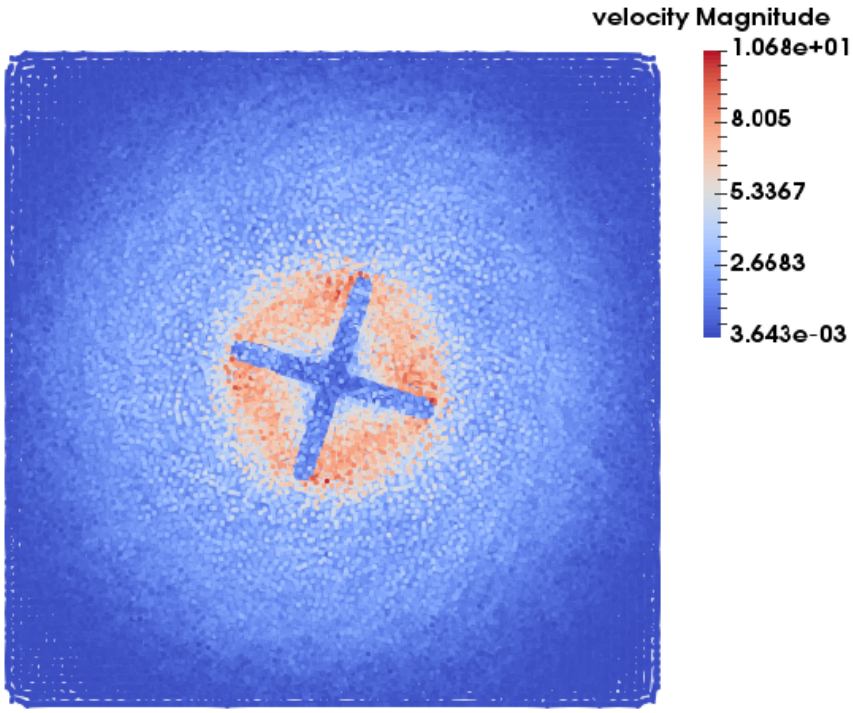


Figure B.2: Bottom view of the vortex (after 40 [s] for stabilization). Projection on the xy-plane.

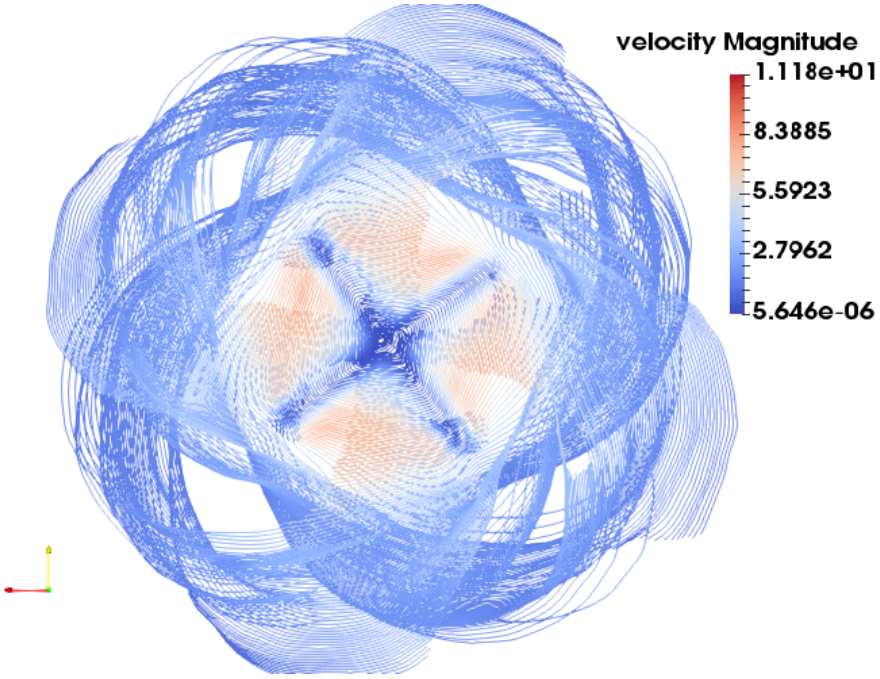


Figure B.3: Several streamlines (after 40 [s] for stabilization). Projection on the xy-plane.

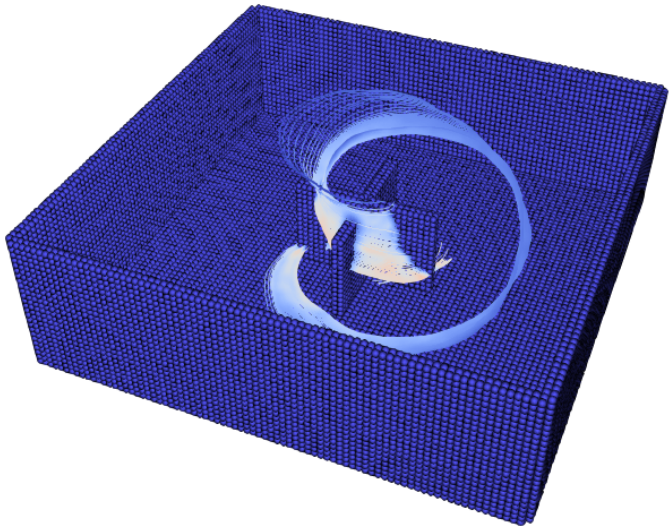


Figure B.4: Perspective view of a set of streamlines. They suggest the recirculations that are encountered in the vortex.

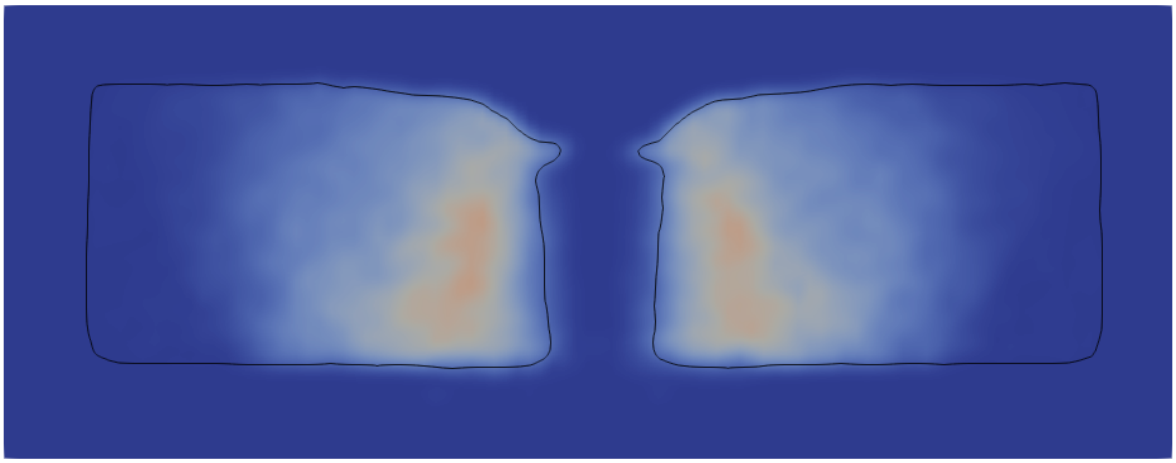
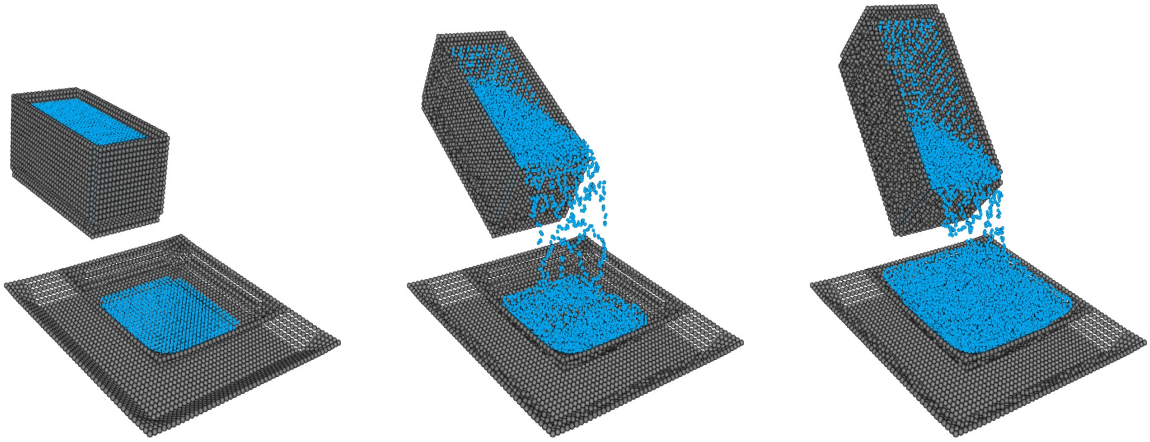


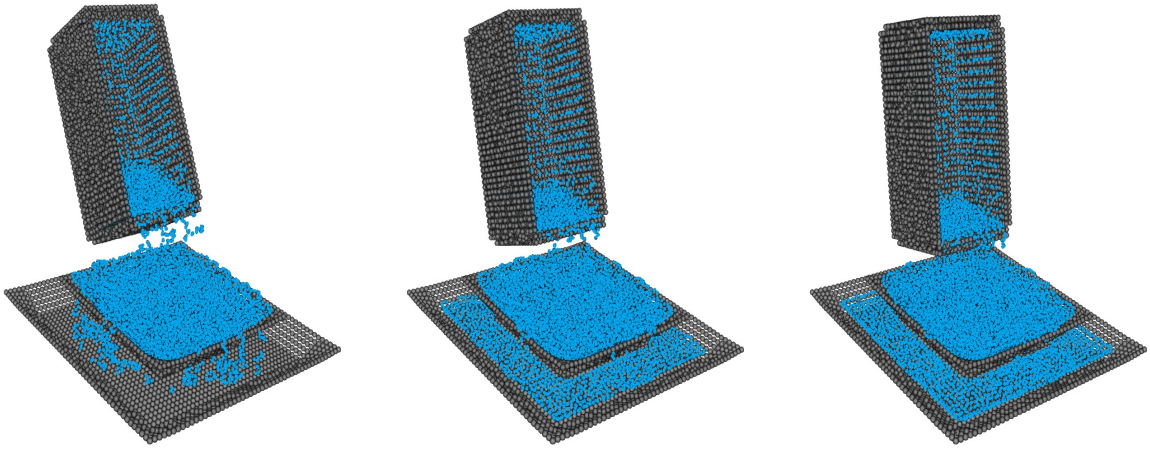
Figure B.5: Vertical cut in the middle of the domain. The black contour delimits the fluid volume and colors are related to the velocity magnitude.

B.2 Fluid pouring

To demonstrate the capabilities of the algorithm to create non trivial geometries, an example of exponential rotating moving boundaries and two bathymetries is shown in Figure B.6. The goal is to simulate the pouring of some liquid into a bowl on a plate.



(a) $t = 0$ [s] (b) $t = 0.9$ [s] (c) $t = 1.8$ [s]



(d) $t = 2.7$ [s] (e) $t = 3.6$ [s] (f) $t = 4.5$ [s]

Figure B.6: Fluid pouring geometry.

B.3 Simple gravity model

The code takes gravity into account by adding to the speed derivative a constant term $\boldsymbol{g} = -9.81\boldsymbol{e}_z$ [m/s^2]. This can be changed to model other simple physical phenomena. For example, a large gravitational mass centered at $\boldsymbol{x} = \boldsymbol{0}$ generates a gravitational field that attracts objects towards it. If the attracted objects have a negligible mass compared to the large one, the center of mass of the latter will not experience large displacement. For any particle external to the big mass, the gravitational pull can thus be assumed to be always pointing towards $\boldsymbol{x} = \boldsymbol{0}$, with a magnitude that is proportional to $1/r^2$, r being the distance between the particle and the point $\boldsymbol{x} = \boldsymbol{0}$. This is illustrated in figure B.7.

If this simple model is implemented[1], interesting results can be obtained. They could model a

[1]The magnitude of the gravitational pull is bounded to avoid divergence when $r \to 0$. A more physical model would be to consider a linear evolution of the force magnitude inside the large mass, which corresponds to the analytical solution of Newton's spherical gravitational objects.

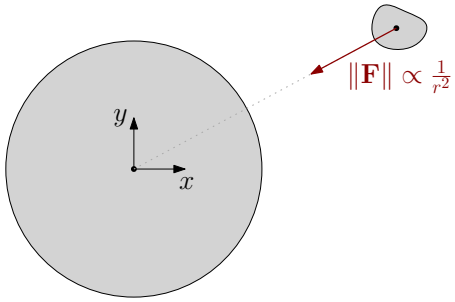


Figure B.7: Simple gravity model.

liquid planet subjected to its own gravitational field. Figures B.8 and B.9 show the results of two simple simulations. Note that the considered model is oversimplified and cannot be used to extract physical predictions. It only illustrates the possible extensions for the implementation.

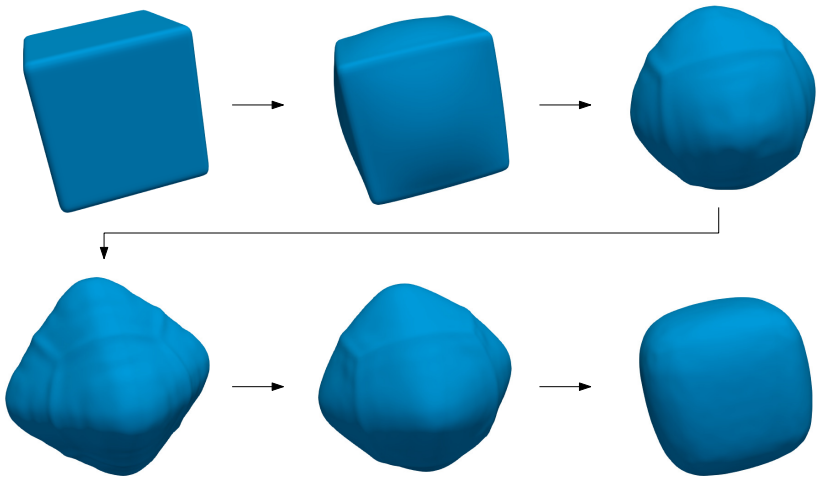


Figure B.8: Case of a single initial cubic mass under its own gravitational field. Eventually, a spherical shape is obtained.

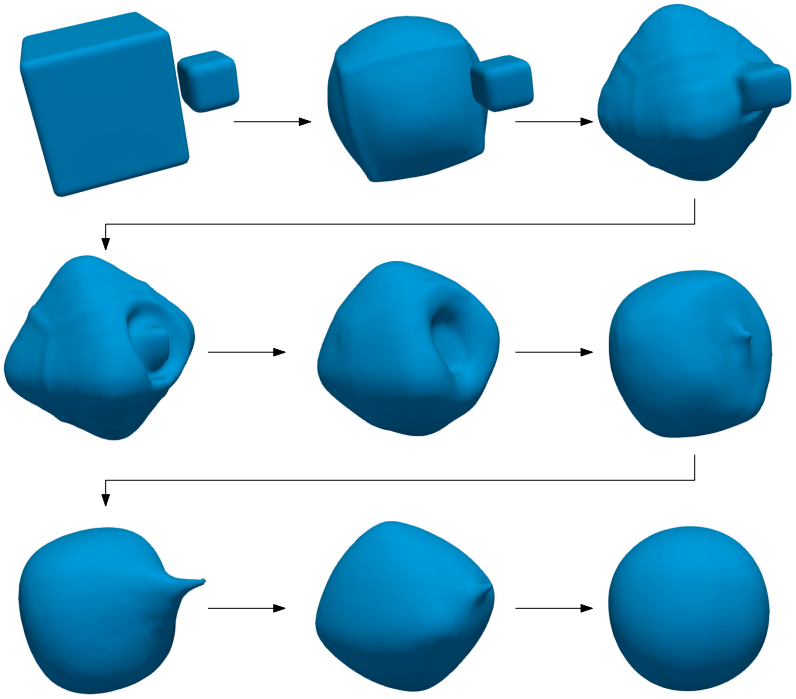


Figure B.9: Cubic object impacting a larger fluid gravitational mass.

B.4 Figures referenced in the text

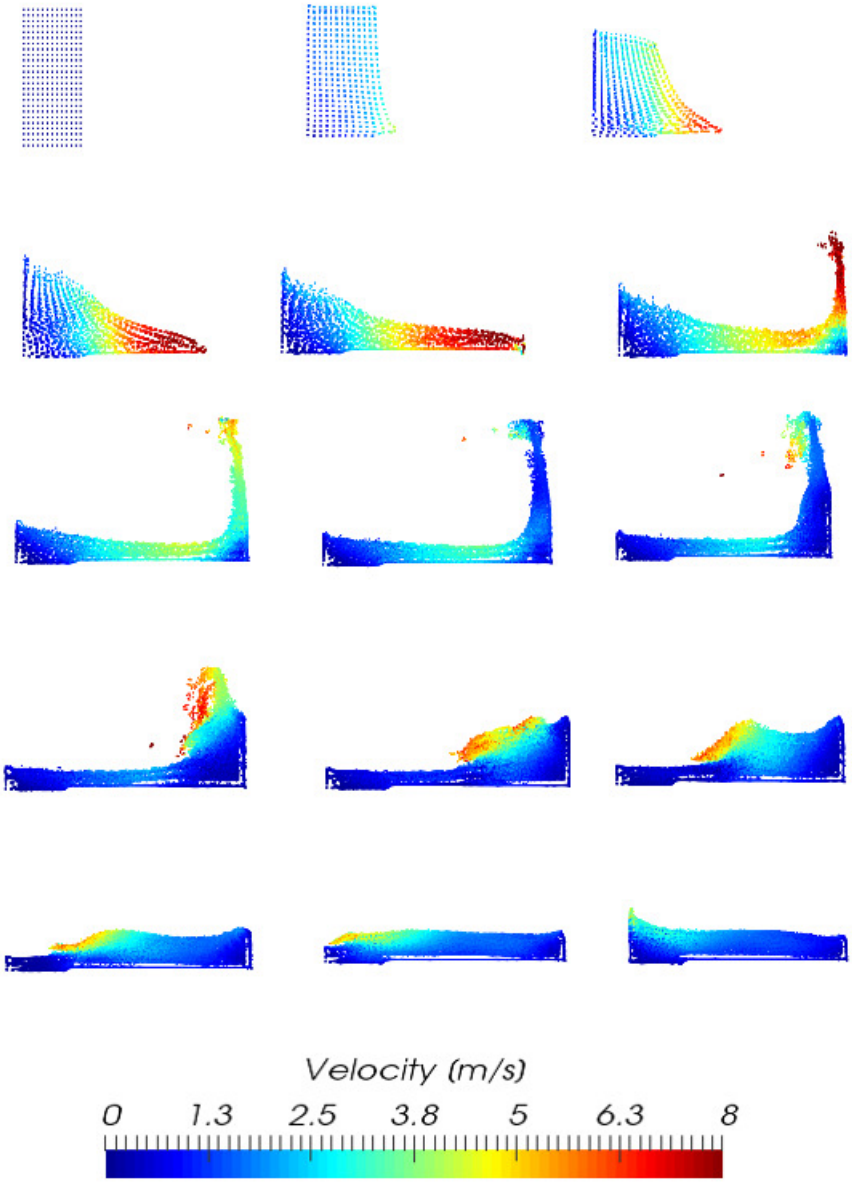


Figure B.10: Snapshots of the dam break simulation results every 0.3 [s] (from $t = 0$ [s] to $t = 4.2$ [s]). The colors are related to the velocity magnitude. See section 5.4.

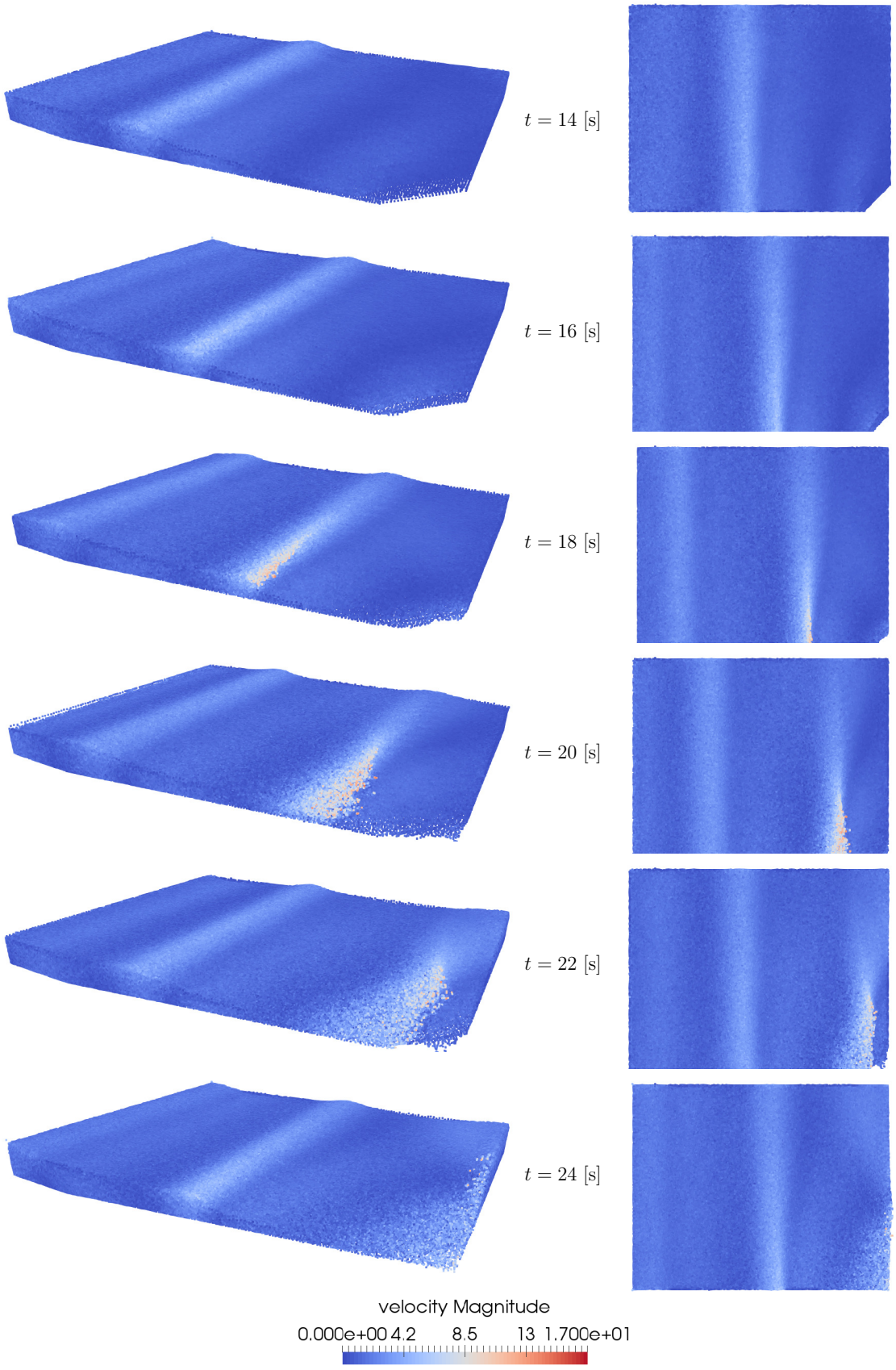


Figure B.11: Snapshots of the results of the breaking wave simulation (velocity in [m/s]) (830,000 particles). See section 5.5.2.

Bibliography

[1] Goffin L., *Development of a didactic SPH model*, 2013.

[2] Lucy L.B. and B.L., *A numerical approach to the testing of the fission hypothesis*, 1977.

[3] Libersky L.D., Petscheck A.G., Carney T.C., Hipp J.R. and Allahdadi F.A., *High strain lagrangian hydrodynamics*. Journal of computational physics.

[4] Nayroles B., Touzot G. and Villon P., *Generalizing the finite element method: Diffuse approximation and diffuse elements*. Computational mechanics (1992), 307-318.

[5] Ma Q.W., *Meshless local Petrov-Galerkin method for two-dimensional nonlinear water wave problems*. Journal of computational physics (2005), 611-625.

[6] Onate E., Idelsohn S.R., *The particle finite element method. An overview.*. International journal of computational method, 2004.

[7] Gomez-Gesteira M., Rogers B.D., Crespo A.J.C., Dalrymple R.A., Narayanaswamy M. and Dominguez J.M., *SPHysics - development of a free-surface fluid solver- Part 1: Theory and Formulations*, 2012.

[8] Gomez-Gesteira M., Rogers B.D., Crespo A.J.C., Dalrymple R.A., Narayanaswamy M. and Dominguez J.M., *SPHysics - development of a free-surface fluid solver - Part 2: Efficiency and test cases*, 2012.

[9] Liu G.R. and Liu M.B., *Smoothed Particle Hydrodynamics*, 2003.

[10] Nelson, Richard P. and Papaloizou, John C.B., *Variable smoothing lengths and energy conservation in smoothed particle hydrodynamics*, 1994.

[11] Gingold, R. A., Monaghan, J.J. *Smoothed particle hydrodynamics - Theory and application to non-spherical stars*, 1977.

[12] Gingold R.A., Monaghan J.J., *The collapse of a rotating non-axisymmetric isothermal cloud*, 1981.

[13] Monaghan, J.J. and Gingold, R. A., *Shock simulation by the particle method SPH*, 1983.

[14] Monaghan, J.J.; Lattanzio, J. C. *A refined particle method for astrophysical problems*, 1985.

[15] Monaghan J.J., *On the problem of penetration in particle methods*, p.8, 1989.

[16] Monaghan J.J., *Smoothed particle hydrodynamics*, 1992.

[17] Monaghan, J.J., *Simulating Free Surface Flows with SPH*, 1994.

[18] Monaghan J.J., *Smoothed particle hydrodynamics*, pp.1738-1740, 2005.

[19] Viggo H. Hansteen, *Numerical modelling of complex turbulent free-surface flows with the SPH method*, 2012.

[20] Benz W., Thielemann F.-K., Hills J.G. *Three-dimensional hydrodynamical simulations of stellar collisions. II - White dwarfs*, 1989.

[21] Antuono M., Marrone S., Colagrossi A., Lugni C., *Gravity wave propagation solved through an enhanced SPH method*, 2010.

[22] Ozbulut M., Yildiz M., Goren O., *A numerical investigation into the correction algorithms for SPH method in modeling violent free surface flows*, 2013.

[23] Onderik J., Durikovic R., *Efficient Neighbor Search for Particle-based Fluids*, 2007.

[24] Balderas D.V.,Dominguez J.M. *et. al*., *Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters*, 2012.

[25] Dominguez J.M., Crespo A.J.C. *et. al*., *New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters*, 2013.

[26] Dominguez J.M., Crespo A.J.C. *et. al*., *DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH)*, 2015.

[27] Geuzaine C., *INFO0939-1 - High performance scientific computing: Project 3*, 2016.

[28] Violeau D. and Issa R., *Numerical modelling of complex turbulent free-surface flows with the SPH method*, 2006.

[29] Koshizuka S. and Oka Y., *Moving-particle semi-implicit method for fragmentation of incompressible fluid.*, Nuclear science and engineering, 123(3): 421-434, 1996.

[30] Pelfrene J., *Study of the SPH method for simulation of regular and breaking waves*, 2011.

[31] Shao S., Lo E.Y.M., *Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface*, 2003.

[32] Dalrymple R.A.,Rogers B.D., *Numerical modeling of water waves with the SPH method*, 2005.

[33] Violeau D., *Fluid Mechanics and the SPH Method: Theory and Applications*, 2012.

[34] Crespo A.J.C., Gómez-Gesteira M., *Boundary conditions generated by dynamic particles in SPH methods*, 2007.

[35] Altomare C., Dominguez J.M. , Crespo A.J.C., *Hybridization of the Wave Propagation Model SWASH and the Meshfree Particle Method SPH for Real Coastal Applications*, 2015.

[36] Ahrens J., Geveci B., Law C., *ParaView: An End-User Tool for Large Data Visualization*, 2005.

[37] The MathWorks Inc., *Matlab*, 2000.

[38] Cheong O., *IPE: an extensible drawing editor*, 1993-2016.