Federal University of Pernambuco

Computer Engineering

Informatics Center

A GPU-accelerated enhanced MPS method for fluid simulation

Author:
André Luiz Buarque Vieira e Silva

Supervisor:
Veronica Teichrieb

Co-Supervisor:
Mozart William Santos Almeida

January 26, 2016

Universidade Federal de Pernambuco

Acknowledgments

É com grande satisfação que concluo o curso de Engenharia da Computação do Centro de Informática da UFPE. Aos que acompanharam minha jornada de 6 anos até aqui sabem do começo inseguro, das dúvidas que tive e o quanto eu relutei até me achar no meu próprio curso, algo que só aconteceu com a ajuda das conversas e reflexões com as diversas pessoas que já conheci e (quem sabe) por algum amadurecimento.

Bem, se achar no seu próprio curso pode ser considerado muita sorte, mas a partir daí ainda havia um longo caminho, basicamente correndo atrás do prejuízo...

Gostaria de agradecer a minha orientadora Veronica Teichrieb o voto de confiança e a oportunidade de trabalhar nesse grupo incrível que é o Voxar Labs com tantos colaboradores admiráveis. Agradecer também o meu coorientador, Mozart, o valioso conhecimento e dicas passadas.

Quero agradecer a minha família, principalmente meu pai, Luiz, minha mãe, Maria das Neves e meu irmão, Felipe, que estão comigo desde o começo, sempre me apoiando incontestavelmente durante minha vida, conversando, refletindo junto a mim, dando momentos de extrema felicidade, e de inúmeras outras formas; foram e são fundamentais!

Gostaria de agradecer o apoio também fundamental e os incentivos incessáveis de Mirella Sampaio, que acompanhou e acompanha com detalhes meu dia-a-dia sempre me deixando mais e mais feliz :)

Não posso esquecer dos guerreiros da minha turma, EC-2010.1, e da galera de tantos outros períodos que também me ajudaram muito a chegar até aqui! Saulete, Paco, Braynner, os magos Tiago e Moiseis, Renê, Thiaguinho, mestre Rennason... Quero mandar um abraço pra galera do Apoio, galera das bandas que eu já tive e tanta outras pessoas que já conheci e me fizeram pensar e refletir, por mais brevemente que fosse, sobre minhas escolhas.

Sinto orgulho desse trabalho, com certeza foi uma das coisas em que mais me empenhei.

Obrigado pessoal!!

Resumo

Métodos sem malha para simular fluxos de fluido têm evoluído cada vez mais com o passar dos anos já que eles são uma ótima alternativa para lidar com grandes deformações, que é onde os métodos baseados em malha não têm um desempenho eficiente. Um dos métodos sem malhas mais conhecidos é o método Moving Particle Semi-implicit (MPS), que foi feito para simular fluxos de fluido incompressíveis de superfície livre. Diversas mudanças e aprimoramentos para o método têm sido propostos desde sua criação e, devido a estes, ele se mostrou bastante útil numa ampla gama de problemas de engenharia. Entretanto, uma de suas desvantagens é a alta carga computacional e algumas funções com execução demorada. Unidades de processamento gráfico (GPU) fornecem recursos sem precedentes para cálculos científicos. A linguagem CUDA é um exemplo. Para promover a otimização por GPU, a solução do sistema de equações lineares na equação de pressão de Poisson (PPE) foi colocada em foco.

Este trabalho de graduação se beneficia de algumas das técnicas apresentadas nos trabalhos relacionados, onde aquelas utilizadas serão especificadas durante este trabalho, e também de CUDA a fim de conceber um método estável, preciso e acelerado por GPU baseado no MPS. Mostra-se que a versão em GPU do método desenvolvido possui um desempenho muito maior e com a mesma confiabilidade que os baseados em CPU.

Palavras-chave: MPS aprimorado, Otimizado por GPU, CUDA, Simulação de fluidos, Métodos sem malha

Abstract

Meshless methods to simulate fluid flows have been increasingly evolving through the years since they are a great alternative to deal with large deformations, which is where mesh-based methods fail to perform efficiently. One of the most well known meshless methods is the Moving Particle Semi-implicit (MPS) method, which was designed to simulate free-surface incompressible fluid flows. Many variations and refinements of the method have been proposed since its creation and, due to these, it has proved to be quite useful in a wide range of engineering problems. However, one of its drawbacks is high computational load and some very time-consuming functions. Graphics Processing Unit (GPU) provides unprecedented capabilities for scientific computations. The CUDA language is one example. To promote the GPU-acceleration, the solution of the linear system of equations in the Poisson Pressure equation (PPE) was brought into focus.

This graduation work benefits from some of the techniques presented in the related work, where those utilized will be specified throughout this work, and also from CUDA in order to get a stable, accurate and GPU-accelerated MPS-based method. It is shown that the GPU version of the method developed can perform much faster with the same reliability as the CPU-based ones.

Keywords: Improved MPS, GPU-optimization, CUDA, Fluid simulation, Meshless methods

Contents

List of Figures

List of Symbols

Symbol	Meaning
D_S	Number of space dimensions
$\mathbf{g}$	Gravitational acceleration
H	Initial height
L	Initial width
n	Particle number density
n_0	Particle density (constant)
n^*	The particle number density calculated in an intermediate step
n'	Correction value of particle number density
p	Pressure
r	Distance between two particles
r_e	Radius of interaction
r_0	Initial spacing distance
$\mathbf{r}_i$	Position in space of particle i
r_{ij}	Distance between particle i and j
t	Time
u_{ij}	Difference between the velocity of particle i and j in the x direction
$\mathbf{u}$	Fluid velocity vector
$\mathbf{u}^*$	The velocity field calculated in an intermediate step
v_{ij}	Difference between the velocity of particle i and j in the y direction
$w(r)$	Kernel function
w_{ij}	Difference between the velocity of particle i and j in the z direction
W	Kernel function
x_{ij}	Distance between particle i and j in the x direction
y_{ij}	Distance between particle i and j in the y direction
z_{ij}	Distance between particle i and j in the z direction
β	Free surface coefficient
δ	Dirac delta function
ν	Kinematic viscosity
ρ	Fluid density
ρ^*	The density calculated in an intermediate step
φ	Represents a physical quantity
Ω	Influence area

Listings

1 Introduction

Some of the most common problems in naval hydrodynamics involve the study of fluid flow. For this, it is necessary to deal with large deformations such as those presented in a good portion of computational mechanics problems [7].

Conventional methods, as the Finite Element Methods (FEM), Finite Difference Method (FDM) and other mesh-based methods (as shown in Figure 1), are considered well consolidated and accurate. However, they are relatively inefficient when dealing with certain problems where it is required the simulation of large deformations. The best approach considered to deal with these large deformations and the moving discontinuities caused by them is to constantly regenerate the mesh in order to keep the mesh discontinuities coincident through the simulation [8].

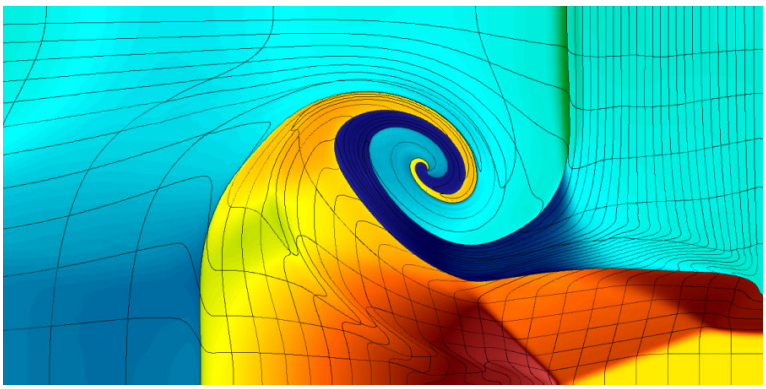


Figure 1: Fluid simulation using mesh-based methods

Clearly, this constant remeshing makes the process quite expensive in terms of computation, probably even causing accuracy degradation [9]. As an attempt to reduce those issues, methods that use meshes and discrete elements, called particles, were proposed. One example is the Particle Finite Elements Method (PFEM) [10]; another alternative, which has presented great potential over the years, are the entirely meshfree methods. They enable, mainly, that free-surface flow can be discretized and solved the Navier-Stokes equations without the need of a grid of any kind, such as [11], achieving flexibility in situations where the classic methods are too complex. Each particle carries a set of physics quantities and constitutive properties, such as mass, velocity and position, and they are responsible for characterizing the system state and its evolution through time. An interesting advantage of the meshless methods with Lagrangian characteristics, is that it allows an easy tracking of each particle's quantities in any moment of the simulation.

Some of the techniques fully free of meshes are the Moving Particle Semi-implicit method (MPS) and the well-known Smoothed Particle Hydrodynamics (SPH). The SPH was designed in the mid-1970s by Lucy [12] and Gingold and Monaghan [13] and intended to astrophysics applications. The first method mentioned, the MPS, was introduced in 1996 with the work of Koshizuka and Oka [14] and it was idealized to simulate the flows of incompressible fluids, which refers to a fluid in which the material density is constant within a fluid parcel. In many

scenarios the changes in temperature and pressure are so small that the density fluctuation is negligible; in such cases the flow may be modeled as incompressible. Its main difference from the original SPH method, which is considered a notable advantage for the MPS method, is that the calculations adopt a semi-implicit predictor-corrector model (which later has been similarly used in some incompressible SPH methods [15]).

1.1 Goals

That said, the MPS method was chosen in this work to be studied and implemented due to its appealing intrinsic incompressibility since this type of fluid flow presents environment importance [16] and appears in many industrial applications [17] [18]. The works of [19], [20] and [21] present systematic comparisons between the two MPS and SPH methods. The main issues of meshfree methods, in general, are in the modeling of solid boundary interaction, fluid flow and, in the specific case of the MPS method, spurious pressure oscillation of the particles [4] [22]. Therefore, several solutions have been proposed in the literature, such as local particle refinement and corrected formulations (as new continuous operators discretization models) [23] [24] [25] [26].

A lot of improvements and adaptations of the original method of both SPH and MPS techniques have been proposed in order to adequate them to the simulation of various kinds of physical phenomena or, more commonly, to get better stability and performance. In this work, more specifically in the MPS related chapter, some of the modifications and enhancements of the said method that were used in this work are addressed. A set of modifications and improvements of the MPS and the SPH method can be seen in the works of [27] [28].

A significant disadvantage of fluid simulation models that value numerical precision is time spent in the application execution, more specifically in the simulation generation [29] [30]. The challenge of dealing with this problem has been diminished through the use of computational platforms that provide Application Programming Interfaces (APIs) making it possible to benefit from the various processing cores of a Graphic Processing Unit (GPU). Created by NVIDIA, CUDA [6] is one of these platforms, which allows software development with CUDA-enabled GPUs for general purpose processing (GPGPU). This platform was explored and utilized in the development of this work.

1.2 Outline

This work intends to present the development and an analysis of a GPU-optimized improved MPS method. In the second chapter, related works, such as other optimized particle-based methods, are presented. Chapter 3 describes the MPS technique showing its governing equations, main application focus and the method's refinements used in this work. After that, chapter 4, shows the methodology adopted throughout the development of this work. Chapter 5 explains the modeling of the test case used as comparison basis between the implementations of the method and then the results obtained and an analysis of the techniques implemented using a general purpose processor (CPU) and utilizing general purpose programming on the graphics processor (GPGPU). At last, in chapter 6, the final considerations are discussed and the contributions of this work are exposed, together with future possibilities and enhancements.

2 State of the Art

Through the years, disasters involving natural phenomena have triggered several researches in many different areas on how to avoid them. Fluid simulation is one of these areas. One of the main reasons of simulating fluids in general has been on how to solve these kind of problems. The MPS method has also been providing great assistance in that field, since it was intentionally created for simulating incompressible flows. The work of Chen et al. is one of the main references of the area [31].

2.1 MPS and Its Variations

The MPS method and variations of it has already been used for various purposes and in various fields, such as nuclear engineering phenomena applied to molten core solidification behavior in nuclear power plant accidents and others [32] [33] [34] [35]. Another example is chemical engineering phenomena applied to eutectic reactions, as well as multiphase fluid simulation [36] [37].

As already has been stated, the MPS method was introduced focusing on the modeling of the behavior of incompressible fluids [14]. Other subsequent works apply the method to certain areas of research, such as coastal and mechanical engineering, among others. In 1998, [38] applied the method to wave breaking in a beach. The authors, in this same work presented an optimization in the neighborhood calculation (from $O(n^2)$ to $O(n^{1.5})$). The previous way (naive) provoked a higher computational load, since the algorithm required that each particle position had to be checked with all the others in the system in order to know which ones were its neighbors. Unfortunately, similarly to the other meshfree methods, the MPS technique suffers from instability problems. Some of these issues are related to numerical errors at the boundaries, i.e., at free-surfaces or when interacting with solid boundaries. The works of [39] and [1] describe why those instability problems arise from the MPS method.

In attempts to overcome these issues, some authors changed the method in order to improve it. Yoon et al. [40] proposed a particle-gridless hybrid method for the analysis of incompressible flows. The numerical scheme developed in order to serve as basis for the method consists of Lagrangian and Eulerian [41] phases in an arbitrary Lagrangian-Eulerian (ALE) method, where a new-time physical property in an arbitrary position is determined by introducing an artificial velocity. The method is applied to a set of sloshing problems and it is shown that the amplitude and period of sloshing are predicted accurately since it shows good agreement with the experimental data gathered by the authors.

Ataie-Ashtiani and Farhadi [42] used a meshless numerical approach to solve Euler's equation, which is the governing equation of the irrotational flow of ideal fluids. Since the time integration of the equations of inviscid flow (mass and momentum conservation) presents difficulties when dealing with incompressible, or nearly incompressible fluids, a fractional step method, which consists of splitting each time step in two, was proposed in order to facilitate solving the inviscid flow equations. Regarding the MPS method stability, various kernel functions were considered and applied to the method, and, as a result of this study, the most suitable kernel function was employed so that the method could increase its stability. The authors concluded that the developed method is quite useful for solving problems with irregular free-surface in hydraulic and coastal engineering when an accurate prediction of free water surface is required.

Lee et al. [1] stated that the MPS method, when it was initially proposed, had several defects including non-optimal source term of the Poisson Pressure equation (PPE), gradient and collision models, and search of free-surface particles, which led to less-accurate fluid motions. In that sense, the authors proposed step-by-step improvements in the processes referred above, originating what they called the PNU-MPS method. After analyzing the improvements using the dam break problem (shown in Figure 2) and the problem of liquid sloshing inside a rectangular tank, the authors concluded that the numerical results for violent free-surface motions and impact pressures are in good agreement with their respective experimental data.

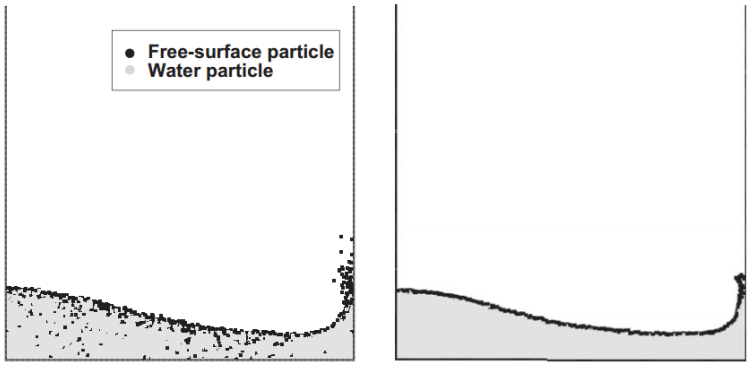


Figure 2: Identification of free-surface particles [1]

Duan and Chen [43] discussed the effects of setting up time step and space step on the stability and accuracy of the viscosity term in the MPS method, which is noted to be a very important property of fluids but not quite easy to simulate. In that work, using the MPS method, two conditions for the setup of time step and initial particle distance in a viscous shear flow simulation method are prescribed to be used specially for simulation flows where viscous forces are dominant. The authors concluded that the stability condition of the viscous term can provide a stable simulation. As for the accuracy condition of the viscous term, it is capable of producing the most accurate simulation for steady laminar flow, and can also provide a realistic and accurate simulation of the molecular viscosity term for unsteady turbulent flow at the expense of a high computational cost though.

One of the biggest issues with the MPS method is the spurious pressure oscillation. Various works already tried successfully to diminish this. In the work of Kondo et al. [39] an artificial pressure is adopted to stop gradual density change (one of the conditions to express incompressibility). The stabilization process consists in eliminate negative pressure after solving the Poisson pressure equation, setting the negative pressures to zero. This problem is due to the particle number densities near the surface being small, which causes the particles' pressure to be negative, thus causing instability in the system. With the scheme proposed, the authors claimed to obtain smoother pressure variations using a dam break test case for the analysis.

A set of papers by Khayyer and Gotoh presents valuable insights and improvements to this problem. Most of them proposed corrected differential operator models (laplacian and gradient). In one of their first attempts they proposed a Corrected MPS (CMPS) method [23]

for the accurate tracking of water surface in breaking waves. Modifications and corrections in gradient operator model used in the standard MPS method are made with the goal to achieve momentum conservation in the calculations of viscous incompressible free-surface flow. Qualitative and quantitative comparisons are employed to show the high capability of the CMPS method in simulating plunging breaking and post-breaking of solitary waves.

Then, in 2009, Khayyer and Gotoh [4] proposed new modifications to the MPS method in order to diminish spurious pressure fluctuation. The authors introduced a new formulation of the source term of the PPE, which was referred as a higher order source term (HS), thus creating the CMPS-HS method after combining this modification with their previous work. Another modification was allowing slight compressibility to the method, that being, adding part of an equation of state (EOS) to the right hand side of the PPE. The compressible term in the equation would have a stabilizing effect on the particle's pressure calculation. It was shown that the proposed methods are applicable for an approximate estimation of wave impact pressure on a coastal structure.

In 2010, Khayyer and Gotoh [24] focused on the Laplacian model used in the MPS method. They noticed that to further refine and stabilize the pressure calculation, a Higher order Laplacian model (HL) for discretization of the Laplacian operator should be derived. This model was applied in both Laplacian of pressure and the one corresponding to the viscous forces. By merging this new model with previous modifications proposed by the same authors, the CMPS-HS-HL was originated. In order to verify the performance of the modification proposed, hydrostatic pressure calculations with designed simple pressure oscillations, as well as exponentially excited sinusoidal pressure oscillations, were carried out. The authors remarked that, although the improvements enhanced pressure calculations, the numerical results still presented some unphysical numerical oscillation during the tests.

After that, in 2011, following the conclusion in their previous work, Khayyer and Gotoh [25] presented two new modifications in order to resolve the shortcomings that were present in the method proposed in their previous work. Another issue found that compromised the method's stability appears to be similar to the so-called tensile instability. The first improvement deals with unphysical numerical oscillation caused by the source term in the PPE, so, extra terms were added to it. These terms are referred by the authors as error compensating parts in the source term of the PPE (ECS) and, by combining with previous works, the CMPS-HS-HL-ECS was created. The second improvement is meant to deal with situations where tensile instability is present. It consists of a corrective matrix inserted in the pressure gradient calculations in order to achieve a more accurate approximation of the differential operator in question.

2.2 Optimizations to the MPS

Since the MPS is a fully meshless method, the particles are not connected explicitly by any edge, therefore it is possible to optimize some computational aspects of the simulation, such as by parallelization, using cluster technology or GPGPU techniques.

Tsukamoto [44] used shared memory parallelization as a way to accelerate the MPS method. His goal was to simulate floating bodies in highly nonlinear waves and he achieved significant performance gains compared with the sequential version of the simulation.

Ikari and Gotoh [45] compared two problem decomposition methods, one based on particles (particles decomposition) and the other on a domain decomposition. They verified that domain decomposition, in most cases, presents a smaller runtime to finish the calculations.

Gotoh [46] developed a MPS version to be executed parallelized, combining domain decomposition techniques with dynamic boundaries, periodically recalculating based on the center of mass of each subdomain, in order to enhance load balancing in the processors and also a process of preconditioner matrix restructuring for accomplishing the forward/backward process of the Conjugate Gradient in parallel. In that work, the performance results of two linear systems solving methods were compared: the Parallelized Incomplete Cholesky Conjugate Gradient with Renumbering Process (PICCG-RP) and the Scaled Conjugate Gradient (SCG). The authors concluded that the proposed method successfully simulated the studied models, but the parallelized model still needed further refinement, that being in precision and computational efficiency. This could be achieved through the development of more accurate and consistent numerical models of differential operators, such as time integration.

Iribe et al. [2] presented simulation results of the parallelized MPS for a PC cluster. The authors identified that the bottleneck of the iterative solver parallelization in shared memory is the computational cost of the communication between subdomains. To minimize this communication, the authors tested a sophisticated particle renumbering process based in packages and in a communication list. With these techniques, they were able to accelerate the communication process. A 237-hour simulation of a tsunami, pictured in Figure 3, with six million particles was generated. The authors concluded that the reordering process proposed can be used to elaborate an efficient scheme of unidimensional decomposition process.

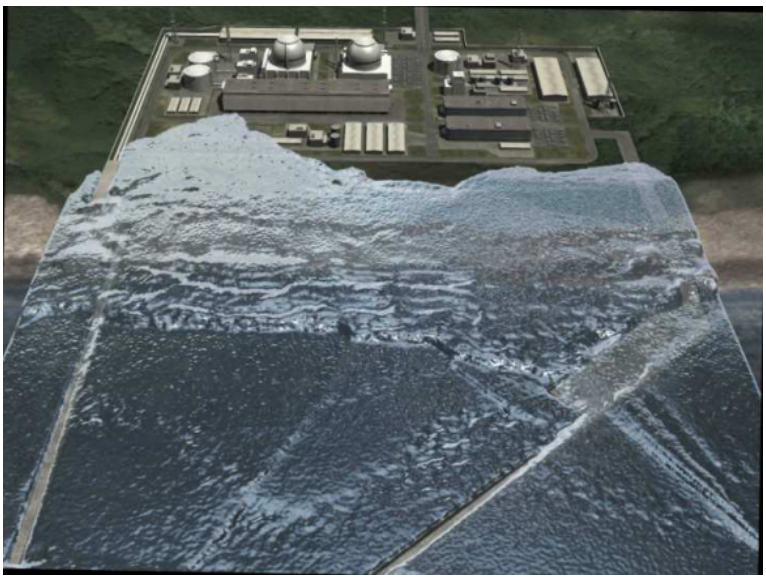


Figure 3: Tsunami simulation [2]

Hori et al. [47] developed a GPU-accelerated version of a MPS code using NVIDIA's CUDA. The authors focused on the search of neighboring particles and the iterative solution of the linear system generated by the Poisson Pressure equation. As the global memory accesses are performed on segments, memory accesses should be coalesced whenever possible, so, arrays of physical quantities are stored to be aligned in the global memory. The optimization of the

search for neighboring particles is achieved through structures called cells, in which each particle is stored in a specific cell according to the particle's position. The length of this structure is determined by the radius of influence of a particle. Neighboring particles of a particle i are searched by visiting the surrounding cells and, evidently, the cell which the particle is associated with. The main issue regarding the PPE is that the sparse matrix-vector (SpMV) multiplication it yields generates a large computational load, monopolizing almost entirely the computational time on a CPU, as well as on a GPU. The authors found a way to achieve coalesced global memory access on loading elements of the coefficient matrix, however, they found to be impossible to do coalesced access to following two arrays of N dimension (N being the total number of particles). So, these arrays were allocated in read-only cached texture memory in order to reduce the effect of the irregular and random access. In order to compare accuracy and performance between the CPU and GPU-based codes, 2-dimensional calculations of an elliptical drop evolution and a dam break flow have been carried out. Finally, the reported speedup achieved in that work is about 3 to 7 times.

Zhu et al. [30] developed a GPU-based MPS model using CUDA on a NVIDIA GTX 280. The authors also considered the implementations of the search of neighboring particles and solving the large sparse matrix equations (Poisson Pressure equation) to be very time-consuming. Two different methods were developed to find the neighbors for a specific particle i, the first one will access the memory in a high frequency with an order of $O(NP^2)$ where NP is the total particle number. The second method is an indirect method; it has a similar approach to [47], where background grids are employed in order to reduce significantly memory access, taking only $O(kNP)$ times. The authors built four different test cases in order to evaluate the GPU program optimization. All the test cases were based in the dam break scenario where each one of them only differed with respect to the total number of particles, raising it from test case 1 to 4. To solve the PPE, the Bi-Conjugate Gradient method (BiCG) is used and it is shown that the percentage of time used for solving the pressure equation decreases from 66% to 40% as the total number of particles raises. The authors concluded through a numerical analysis that the models based on CPU and GPU have the same precision and, through a comparison between the model's performance, a 26 times speedup can be obtained with the MPS-GPU in contrast to the MPS-CPU.

In Fernandes's PhD thesis [48], he developed a computational framework of hybrid parallelization of the MPS method. He, firstly notes the method's great applicability, such as in the influence of the movement of ships in waves, simulations of phenomena involving fragmentation, fluid dynamics in extreme conditions and dealing with large deformations. However, due to the high number of particles used in the simulation of complex systems by the MPS, the author focused on parallelizing a highly scalable solution making use of a computer cluster, which would be of easy maintenance and extensibility. He concluded that his work contributed to the consolidation of the MPS method as a practical tool to investigate complex engineering problems, since the method has its applicability extended to scenarios with tens of millions of particles.

3 The Moving Particle Semi-implicit Method

3.1 Motivation

Meshless methods were conceived as an alternative to grid-based methods since the process of remeshing to keep mesh lines consistent is time consuming and may lead to degradation of computational accuracy. The Moving Particle Semi-implicit (MPS) method was one of the first meshless methods proposed and initially created solely to simulate incompressible free-surface fluid flows, differently from other meshless methods as the Smoothed Particle Hydrodynamics (SPH), which was invented for modeling astrophysical phenomena and later adapted to solve fluid dynamics problems. The MPS method was created by [18] and is similar to the SPH method in that both methods employ simplified differential operators models. But, in the case of the MPS method, these models are based on a local weighted averaging process without taking the gradient of a kernel function. Another disparity between the two methods is that, while the standard SPH does not guarantee fluid incompressibility, the modeling of incompressibility is intrinsic to the MPS, an important characteristic in some fluid dynamics simulations [3]. This is due to the fact that the main goal of the SPH was not to simulate fluid flows. The key advantage of the MPS method is on how it can achieve fluid incompressibility through relatively simple formulations, since the method employs simplified differential operator models to approximate the Gradient and Laplacian, for example, of a field variable. These aspects contributed to the diverse applicability of the method. As the SPH method, the MPS uses models to approximate differential operators. Therefore, precision issues are associated. For this, several improvements and adaptations to the method were proposed. The ones which were utilized in this work will be addressed in the subsequent topics. Firstly, it will be shown the standard method including its governing equations and algorithm.

3.2 Standard Method & Governing Equations

This method models the fluid as an assembly of interacting particles, in which their motion is determined through the interaction with neighboring particles and according to the governing equations of fluid motion. To describe the motion of a viscous fluid flow, there is the continuity equation and Navier-Stokes equation as follows in Eq. (1) and Eq. (2).

$$\frac{1}{\rho}\frac{D\rho}{Dt} + \nabla.\mathbf{u} = 0 \tag{1}$$

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \nu\nabla^2\mathbf{u} \tag{2}$$

where $\mathbf{u}$ is the fluid velocity vector, t is the time, ρ is the fluid density, p is the pressure, $\mathbf{g}$ is the gravitational acceleration vector and ν is the laminar kinematic viscosity. To adapt these equations so that a fluid can be represented by discrete elements, some of these physical quantities become particles attributes; so $\mathbf{u}$ becomes the velocity vector of a particle, ρ now stands for the density of the particle and p, the pressure of a particle. The left hand side of the continuity equation (Eq. (1)) is represented, in the case of incompressible flow, by a simple volume continuity equation, as presented in Eq. (3) [5]:

$$\nabla.\mathbf{u} = 0 \tag{3}$$

A particle interacts with its neighbors through a kernel function $w(r)$, r being the distance between two particles. The most common form of kernel function employed in MPS, and used for the implementation in this work, is in (4):

$$w\left(|\mathbf{r}_j - \mathbf{r}_i|\right) = \begin{cases} \frac{r_e}{|\mathbf{r}_j - \mathbf{r}_i|} - 1 & , \quad 0 \leq r < r_e \\ 0, \quad r_e \leq r \end{cases} \tag{4}$$

where r_e is the radius of the interaction area. Clearly, a larger kernel size implies in an interaction with more particles, as seen in Figure 4.

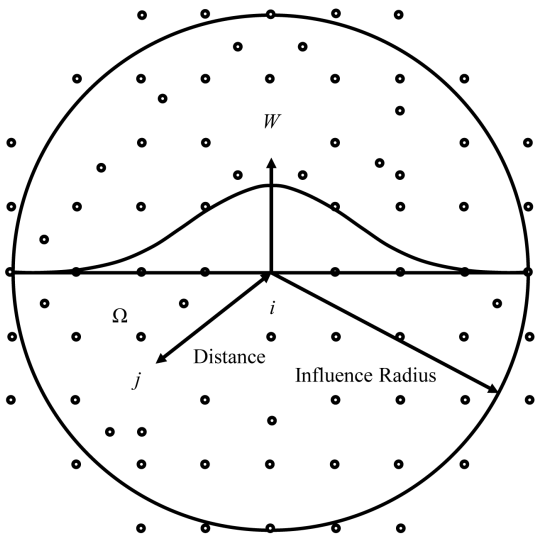


Figure 4: Radius of influence of a particle in a two-dimensional problem

There are also other types of kernel functions and a detailed analysis on the subject can be seen in [42].

To find all the neighboring particles j of each particle i in the simulation, there must be used, preferably, an efficient search strategy and a data structure to store the whole list. The simplest neighboring particle search algorithm is called "all-pair search algorithm", which can be readily implemented as neighbor search strategy of the MPS method. In this algorithm, for each target particle i, the distance from another particle j is checked to see if the position of the particle j is inside the target's radius of influence. If this condition is met, the particle j is a neighbor of particle i.

The particle number density n, which is proportional to the fluid density [37], at the position $\mathbf{r}_i$ of the particle i is defined as follows:

$$n_i = \sum_{j \neq i} \left(|\mathbf{r}_j - \mathbf{r}_i|\right) \tag{5}$$

Thus, the continuity equation is satisfied if the particle number density remains constant, and this constant value is denoted by n_0. As stated before, in the original MPS method the derivative of a kernel is not calculated, instead, the gradient or Laplacian are obtained by local weighted averaging of these operators calculated between a pair of particle i and a neighbor, particle j.

Below, it can be seen a graphical representation of the gradient model in 5 as its formulation used in MPS of a physical quantity φ:

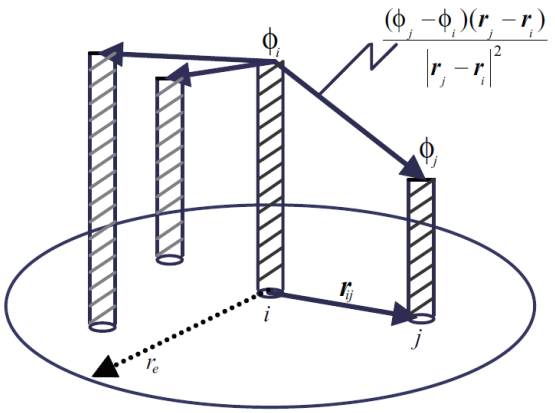


Figure 5: Gradient graphic representation [3]

$$\nabla\varphi_i = \frac{D_S}{n_0}\sum_{j\neq i}\frac{(\varphi_j - \varphi_i)}{|\mathbf{r}_j - \mathbf{r}_i|^2}(\mathbf{r}_j - \mathbf{r}_i)\, w(|\mathbf{r}_j - \mathbf{r}_i|) \tag{6}$$

In Eqs. (6) and (7), D_S is the number of space dimensions. This model is ultimately applied to the pressure gradient term. The Laplacian of φ, applied to the pressure and in the viscous stress calculation in this method, is represented by:

$$\nabla^2\varphi_i = \frac{2D_S}{n_0\lambda}\sum_{j\neq i}(\varphi_j - \varphi_i)\, w(|\mathbf{r}_j - \mathbf{r}_i|) \tag{7}$$

where λ is the weighted average of r_{ij}^2, as shown below, which implies the variance of the randomly distributed particle positions.

$$\lambda = \frac{\int_V w(r)r^2 dv}{\int_V w(r)dv} \tag{8}$$

Eq. (9) shows the discretized calculation of lambda that can be deducted from Eq. (8):

$$\lambda = \frac{\sum_{j\neq i} w(|\mathbf{r}_j - \mathbf{r}_i|)r_{ij}^2}{\sum_{j\neq i} w(|\mathbf{r}_j - \mathbf{r}_i|)} \tag{9}$$

To model incompressibility the satisfaction of the continuity equation is indispensable, so, the fluid density must remain constant. When the particle number density n^* calculated in an intermediate step is not equal to n_0, it is implicitly adjusted to n_0:

$$n^* + n' = n_0 \tag{10}$$

where n' is the correction value. Differently from many SPH-based calculations where the equations are solved explicitly, the pressure in MPS is implicitly calculated by solving a Poisson Pressure Equation (PPE). The other terms are approximated explicitly; this is why the MPS algorithm is semi-implicit, thus giving the name of the method. To solve the PPE it is necessary a two step prediction-correction process. In the first step there is the explicit integration in time, while, in the second step, the implicit computation of a divergence-free velocity field occurs. The calculation of the intermediate velocity field $\mathbf{u}^*$ is derived from the implicit pressure gradient term as:

$$\mathbf{u}_i^* = \mathbf{u}_i^k + \frac{\Delta t}{\rho_i^*} \nabla p^{k+1} \tag{11}$$

where k indicates the current time step in the simulation, ρ^* is the density calculated at time step k and p indicates the particle pressure. The velocity and particle densities in Eq. (11) satisfy the mass conservation law as follows:

$$\frac{1}{\rho} \frac{D\rho}{Dt} + \nabla . \left(\mathbf{u}_i^{k+1} - \mathbf{u}_i^* \right) = 0 \tag{12}$$

By representing the derivative of the ρ as $\frac{\rho_0 - \rho_k^*}{\Delta t}$ and substituting ρ for n, it is possible to deduce the PPE [14]:

$$\nabla^2 p_i^{k+1} = -\frac{\rho}{\Delta t^2} \frac{n_i^* - n_0}{n_0} \tag{13}$$

The Incomplete Cholesky Conjugate Gradient (ICCG) method is usually employed to solve the linear system [14] [4]. The pressure gradient term in Eq. (11) and the term in the left hand side of Eq. (13) are calculated by applying the gradient model shown in Eq. (6) and the Laplacian model presented in Eq. (7), respectively. By solving the PPE, the velocity in time step $k+1$ ($\mathbf{u}_{k+1}$) can be calculated, and, at last, the particle positions, denoted by r in Eq. (14), are updated through a simple first-order Euler integration.

$$\mathbf{r}_i^{k+1} = \mathbf{r}_i^k + \mathbf{u}_i^{k+1} \Delta t \tag{14}$$

The solid boundaries in standard MPS, as walls and fixed obstacles, are represented by fixed particles with no velocity. Some of these particles, however, are considered to solve the PPE. To tell which will be used for the pressure calculations, it is important to explain that there are two layers of wall particles. One of these layers will be referred as inner wall particles (those that, initially, come into direct contact with the fluid particles) and the other as dummy particles (which complement the solid boundary). Usually, just some lines (often two) of dummy particles are used [38]. A model can be seen in Figure 6. The PPE is solved by taking into account the inner wall particles only to repel the fluid from the solid boundaries, while the dummy particles were introduced so that the particle number density at the inner wall particles is not small and that they are not recognized as free-surface.

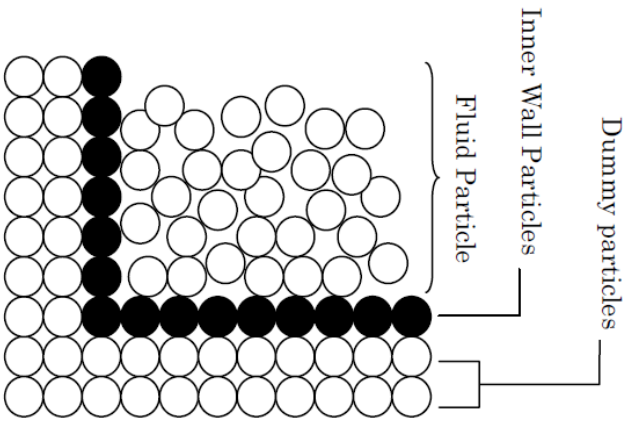


Figure 6: Dummy boundary scheme

To identify a free-surface particle, the particle number density of the ith particle just needs to satisfy the condition presented in Eq. (15) since on the free-surface the particle number density drops abruptly.

$$n_i < \beta n_0 \tag{15}$$

The coefficient β is a parameter below 1.0 and it will judge if a particle is on the free-surface or not. The bigger beta is, the bigger will be the number of particles recognized as free-surface. [14] indicates that the parameter's value can be set between 0.8 and 0.99 and recommends setting it to 0.97. Figure 7 gives an overview of the MPS method algorithm.

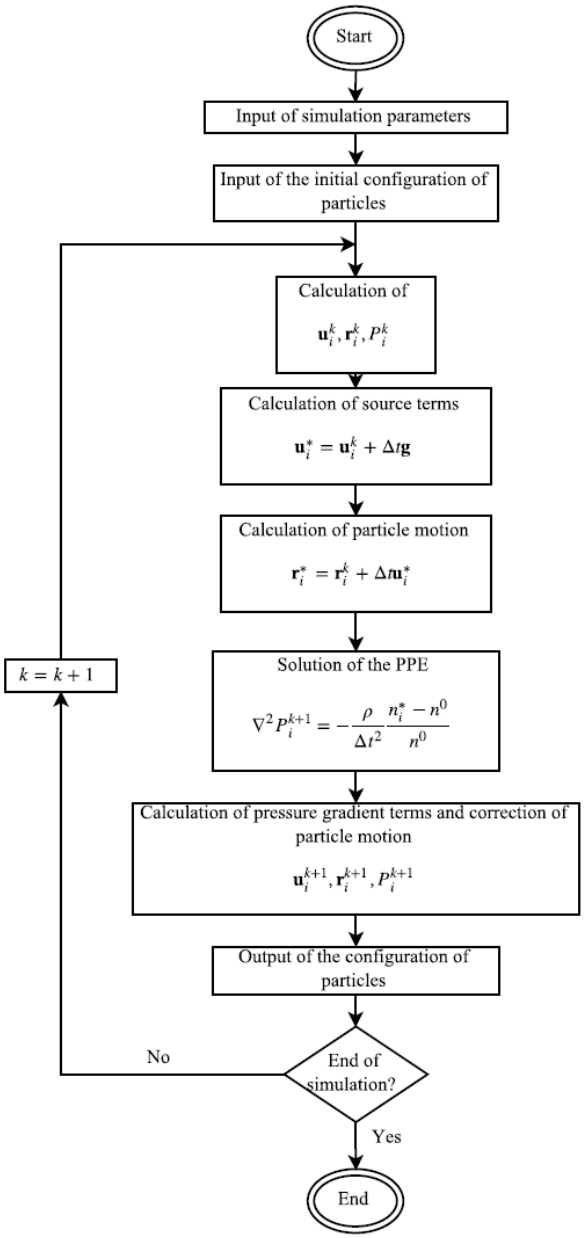


Figure 7: Algorithm of MPS method

3.3 MPS Enhancements

In this section, the improvements of the standard MPS that were implemented in this work are gathered so they can be described, as their versatility and wide range of applicability can be

shown. It is noteworthy that the universe of variations is much larger and only the ones that could really contribute with a more stable and physically coherent simulation were selected to be implemented and explained here. Some of the enhancements are related to achieving a more consistent momentum conservation, guaranteeing a more realistic pressure oscillation and improving numerical stability.

3.3.1 Momentum Conservation

In the MPS, the momentum conservation issue is poorly explored. [49] have developed the HMPS (Hamiltonian MPS) in which the momentum and mechanical energy of the system are preserved. Although it is superior, the HMPS carries heavy theory to its calculations making it extremely complicated to implement in comparison to the standard MPS method. More details on the extensive HMPS formulations can be seen in [49]. A simpler way to achieve a consistent conservation of linear momentum is to propose modifications on the models of differential operators that are directly connected to assure conservation of linear momentum in order to obtain more accurate results, which is the case of the gradient model. In the standard MPS method, the pressure gradient term does not guarantee the linear momentum conservation so, [23] suggested an alteration on the pressure gradient formulation:

$$\nabla p_i = \frac{D_S}{n_0} \sum_{j \neq i} \frac{\left(p_i + p_j\right) - \left(\hat{p}_i + \hat{p}_j\right)}{\left|\mathbf{r}_j - \mathbf{r}_i\right|^2} \left(\mathbf{r}_j - \mathbf{r}_i\right) w\left(\left|\mathbf{r}_j - \mathbf{r}_i\right|\right) \tag{16}$$

$$\hat{p}_i = min_{j \in J}(p_i, p_j), J = \{j : w(|\mathbf{r}_j - \mathbf{r}_i|) \neq 0\} \tag{17}$$

When the anti-symmetric Eq. (16) is applied, linear momentum is exactly conserved. This method is referred by the authors as Corrected MPS (CMPS).

3.3.2 Pressure Calculation

One of the major issues of the MPS method is the spurious pressure oscillation. Some studies propose enhancements to that calculation. The most recent works that presented substantial improvements in this area, making few and simple modifications in the implementation of the method, were proposed by [4] and [24]. The first one is called by the authors as MPS method with a Higher order Source term (MPS-HS), since it basically presents a new formulation for the calculation of the derivative of the particle number density with respect to time $\frac{Dn}{Dt}$. Using this method, the Eq. (13) is substituted by the Eq. (18) in the two dimensional case and Eq. (19) in the three dimensional implementation of the method [26].

$$\nabla^2 p_i^{k+1} = \frac{\rho}{n_0 \Delta t} \left(\frac{Dn}{Dt}\right)_i^* = -\frac{\rho}{n_0 \Delta t} \left(\sum_{i \neq j} \frac{r_e}{r_{ij}^3} \left(x_{ij} u_{ij} + y_{ij} v_{ij}\right)\right)^* \tag{18}$$

$$\nabla^2 p_i^{k+1} = -\frac{\rho}{n_0 \Delta t} \left(\sum_{i \neq j} \frac{r_e}{r_{ij}^3} \left(x_{ij} u_{ij} + y_{ij} v_{ij} + z_{ij} w_{ij}\right)\right)^* \tag{19}$$

It is important to note that all the enhancements shown so far can, and most of them normally are (mostly when they are suggested by the same authors) be combined in one single

method with the purpose of bringing forth a more robust outcome. So, for example, the CMPS and MPS-HS were merged to create the CMPS-HS. The results generated by this refined MPS method are presented in the work of Khayyer and Gotoh [4] which shows a comparison with the standard MPS through a simple test where the hydrostatic pressure varies with time at a fixed point at the center of the bottom of the fluid recipient. This comparison is pictured below in Figure 8.

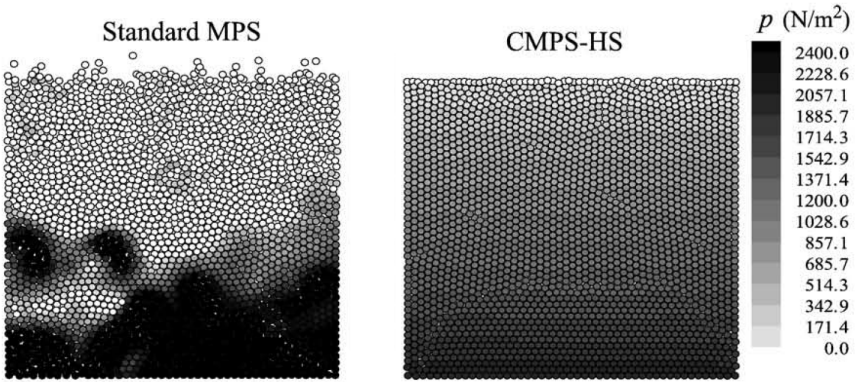


Figure 8: Fluid pressure comparison between MPS and CMPS-HS [4]

The second important and recent improvement to the pressure calculation of the method was the proposition of a new Laplacian model. A higher order source term is employed for the PPE. However, the Laplacian of the pressure at the left hand side of the PPE has been discretized by the standard MPS Laplacian (Eq. (7)). Thus, a higher order Laplacian was derived by [24] and [26] for both two (Eq. (20)) and three (Eq. (21)) dimensional simulations.

$$\nabla^2 \varphi_i = \frac{1}{n_0} \sum_{i \neq j} \left(\frac{3 \varphi_{ij} r_e}{r_{ij}} \right) \tag{20}$$

$$\nabla^2 \varphi_i = \frac{1}{n_0} \sum_{i \neq j} \left(\frac{2 \varphi_{ij} r_e}{r_{ij}} \right) \tag{21}$$

This new derivation was named MPS with a Higher order Laplacian of pressure (MPS-HL). The addition of this enhancement to the ones previously proposed by the same authors generated the 2D and 3D CMPS-HS-HL method.

3.3.3 Enhancement of Numerical Stability

In order to enhance even further the accuracy of numerical solutions, satisfy the fluid incompressibility, and achieve accurate pressures and velocity fields, [25] came up with a PPE's source term with error-compensating parts. The error-compensating terms should be measures for instantaneous and accumulative violations of fluid incompressibility. Eq. (22) shows the suggested terms to be added to the source term of the PPE and Eq. (23) shows the complete modified PPE.

$$ECS = \left| \left(\frac{n^k - n_0}{n_0} \right) \right| \left[\frac{1}{n_0} \left(\frac{Dn}{Dt} \right)_i^k \right] + \left| \left(\frac{\Delta t}{n_0} \left(\frac{Dn}{Dt} \right)_i^k \right) \right| \left[\frac{1}{\Delta t} \frac{n^k - n_0}{n_0} \right] \tag{22}$$

$$\nabla^2 p_i^{k+1} = \frac{\rho}{n_0 \Delta t} \left(\frac{Dn}{Dt} \right)_i^* + ECS \tag{23}$$

For more details on the derivation of the error-compensating terms for the source term of the PPE (MPS-ECS), it is recommended a deep investigation of [25].

The combination of all the refinements shown so far by Khayyer and Gotoh gives as outcome the CMPS-HS-HL-ECS (Corrected MPS with a Higher order Source term - Higher order Laplacian of pressure - Error Compensating parts in the Source term) method. According to [5], the CMPS-HS-HL-ECS method ensures satisfactory accuracy and stable computation, more specifically, under the absence of tensile stress. A comparison between CMPS-HS-HL-ECS and the standard MPS can be seen in the work of Gotoh [5] which presents a standard test case found in the literature, the breaking waves. This comparison is abridged in Figure 9, where different shades of gray represent different pressure levels.

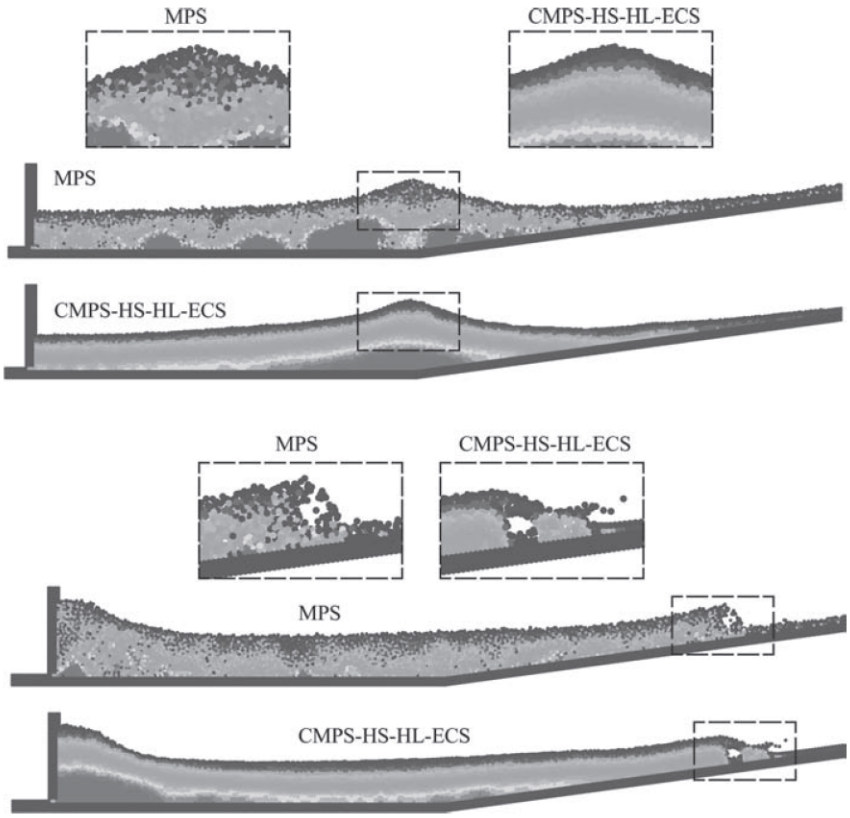


Figure 9: Comparison between standard MPS and CMPS-HS-HL-ECS through breaking waves test case [5]

4 Implementation Methodology

4.1 Software and Hardware Infrastructure

The development of the whole system was divided in two parts. In the first part, the standard MPS technique was implemented using the C++ programming language, taking advantage of its object oriented features. After that, each one of the improvements shown in the previous chapter were implemented in order to get a more stable and physically accurate MPS-based method, the CMPS-HS-HL-ECS method. In this first part, the program is run in the general purpose processor. In the second part of the development, the code previously implemented to run in the CPU would now be altered, using the CUDA C/C++ programming language, in order to obtain high performance during its execution in the GPU.

The integrated development environment (IDE) used to write, run and debug all the code created in this work was the Microsoft Visual Studio 2013, which ran on the Microsoft Windows 10 operating system. The CPU used was a Intel® Core™ i7-4790 CPU @ 3.60 GHz [50] with 7.86 GB of installed RAM and a 64-bit operating system (x64). The GPU used in this work is a NVIDIA GeForce GTX 760 with 1152 CUDA cores [51].

At the end of the execution of each time step, the program stores all the information regarding that step, such as each position (considering the x and y-axis), each x and y velocity component and the value of the pressure of every particle in the simulation that is being generated, in a XML-like file called a VTU file [52]. To visualize the simulation, the software Paraview [53] is used; it reads all the VTU files generated by the program and shows graphically their information, as shown in Figure 10.

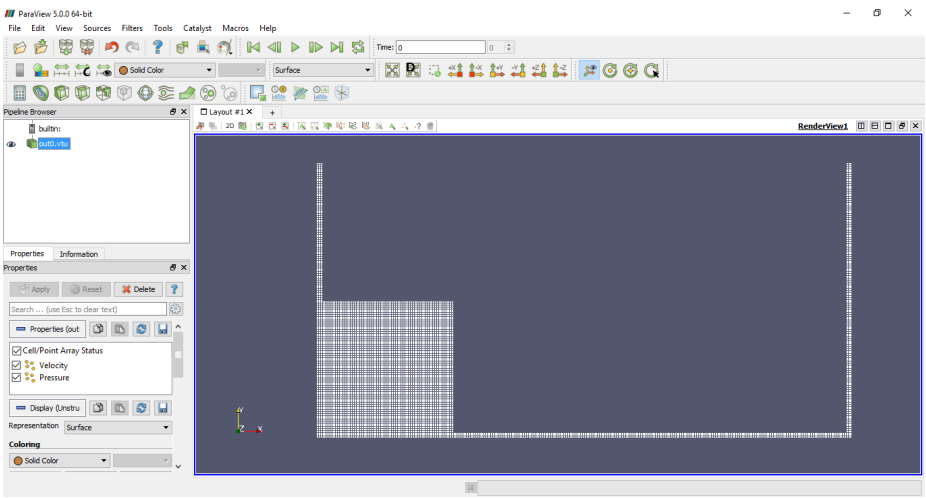


Figure 10: Paraview software

4.2 CPU Code Development

For developing a C++ version of the MPS method, the standard method implementation was used as a basis. A class called `particles` was created so it could store all the physical

quantities of a particle, such as position, velocity, pressure, particle number density and particle type, were this last one would store the information whether the particle was part of the fluid, the dummy particles (outer layer wall) or the inner wall particles (those that interact directly with fluid particles). These physical quantities are the class attributes and the class methods will return and set the values of these attributes.

Two configuration files are responsible to set the parameters of the simulation. The one called `mps.grid` stores all the attribute values of each particle present in the simulation, so this file will store the initial state of the simulation. As for the other, `mps.data`, stores information regarding how the particle and simulation will behave, such as average particle distance, time step length (in seconds), gravitational force value, fluid density, maximum number of iterations and many other parameters.

The neighboring particles search approach in this work is the optimized all-pair search algorithm proposed by [38]. The standard all-pair search algorithm was explained in the MPS method chapter. The optimized version of this algorithm allows that, while a particle j is being set as a neighbor of the target particle i, inside the same loop iteration the particle i is also set as a neighbor of particle j. This makes the computational cost of the algorithm go from $O(n^2)$ to $O(n^{1.5})$. Listing 1 shows a code snippet of this algorithm implementation.

Listing 1: Optimized all-pair search algorithm

```
(...)

/* Getting position of potential neighbors, particles i and j */
Part_j = (*particles)[j].get_po();
Part_i = (*particles)[i].get_po();

/* Getting distance in x and y-axis between both particles */
x = Part_j.x - Part_i.x;
y = Part_j.y - Part_i.y;

/* If their euclidean distance is less or equal to the radius
        of interaction value, i and j are neighbors */
if(((x*x)+(y*y)) <= r2)
{
        /* Increments neighbors number of each particle */
        neighbors[i][0]++;
        neighbors[j][0]++;

        /* Stores the neighbor index in the array of neighbors
                of the target particles */
        neighbors[i][neighbors[i][0]] = j;
        neighbors[j][neighbors[j][0]] = i;
}


(...)
```

It is important to note the way how the Poisson Pressure equation was solved has changed, so a more widely used method of good performance would be employed. The method chosen was the Biconjugate Gradient Stabilized (BiCGStab). The Gmm++ [54] library allowed the use of the method. This library provides a few basic types of sparse and dense matrices, vectors and some iterative linear systems solvers.

After implementing the basic MPS method in C++, the improvements shown in the previous chapter were implemented. It will be presented, in source code listing, the specific excerpts where the two codes differ from one another. The first difference is the pressure gradient calculation, which represents the CMPS improvement [23] and is shown in Listing 2.

Listing 2: Gradient model modification

```
(...)

/* Calculating euclidean distance between particles i and j */
dist = sqrt(dx*dx + dy*dy);

ddv = dt;

/* Commented line below is used for the standard MPS method */
//ddv = ddv*((*particles)[j].get_pr() - pmin[i]);
/* Line below substitutes the line above in order to improve
        momentum conservation */
ddv = ddv*(((*particles)[i].get_pr() + (*particles)[j].get_pr())
        - (pmin[i] + pmin[j]));

ddv = ddv / dist *k.weight(dist, radius);
ddv = ddv / rho;
ddv = ddv / n0p;
ddv = ddv* num_D;

/* Summing after each calculation to find gradient value */
DdvCorrector[i].x += ((ddv * dx) / dist);
DdvCorrector[i].y += ((ddv * dy) / dist);

(...)
```

The second change, the calculation of the higher order source term, combined with the previous change, generates the CMPS-HS [4]. Its implementation is shown in Listing 3.

Listing 3: Source term modification

```
(...)

/* Calculating euclidean distance between them */
dist = sqrt(dx*dx + dy*dy);

Part_jv = (*particles)[j].get_v();
Part_iv = (*particles)[i].get_v();

/* Getting velocity difference between particles */
vx = ((Part_iv.x) - (Part_jv.x));
vy = ((Part_iv.y) - (Part_jv.y));

/* Applying new calculation of the source term */
sum += (radius / (dist*dist*dist))*(dx*vx + dy*vy); // MPS-HS

(...)
```

After that, the calculation of the higher order Laplacian was implemented [24] in order to refine even more the pressure calculations and, now, shaping the CMPS-HS-HL method. Listing 4 shows the core of its implementation.

Listing 4: Laplacian model modification

```
(...)

/* Getting euclidean distance between particles i and j */
dist = sqrt(dx*dx + dy*dy);

/* Applying new formulations to improve the Laplacian calculation */
val = 3.0 * radius_ICCG;
val = val / (dist*dist*dist);
val = val / n0pICCG;
val = val / rho;

/* Forming the coefficient matrix */
matrix_line_aux[j] = -val;
matrix_line_aux[cont_line] += val;

(...)
```

Finally, the last improvement implemented was the error compensating parts in the source term (ECS) proposed by [25] and, at last, generating the CMPS-HS-HL-ECS method. Listing 5 shows a code snippet of its implementation.

Listing 5: Error compensating parts in the source term

```
1  (...)
2
3  /* Calculating the value of the error compensating parts */
4  alpha = ((*particles)[i].get_n() - n0p) / n0p;
5  beta = (dt / n0p)*(source_term_aux[i-1] / (-1.0 / (n0p*dt)));
6  ECS = fabs(alpha)*(beta / dt) + fabs(beta)*(alpha / dt);
7
8  (...)
9
10 /* Adding them to the source term calculation */
11 temp_b.push_back((-1.0 / (n0p*dt))*sum + ECS);
12
13 (...)
```

The calculation of the variable `sum` in the line 11 of Listing 5, is shown in Listing 3.

Regarding the system resolution using the Gmm++ library, an incomplete LU factorization [55] preconditioned BiCGStab method was used. Preconditioners are commonly used to accelerate convergence of iterative methods. In Listing 6 the core to the solution of the PPE using the Gmm++ is shown. Variable `M_aux` is the coefficient matrix, which is later copied to `Mgmm`, while `Bgmm` is the source term. The pressure of each particle in the system is stored in `Xgmm`.

Listing 6: Solving $M * X = B$, the PPE

```
1  (...)
2
3  /* 10 fill-in and a threshold of 1E-6 */
4  gmm::ilut_precond< gmm::row_matrix< gmm::rsvector<double> > >
5  P(M_aux, 10, 1E-6);
6
7  /* 1E-8 is the relative residual to be obtained to
8   achieve convergence */
9  gmm::iteration iter(1E-8);
10
11 /* Set to zero all elements whose modulus is less than
12   or equal to 1E-12 */
13 gmm::clean(M_aux, 1E-12);
14
15 gmm::csc_matrix<double> Mgmm;
16 gmm::copy(M_aux, Mgmm);
17
18 /* Solves for Xgmm */
19 gmm::bicgstab(Mgmm, Xgmm, Bgmm, P, iter);
20
21 (...)
```

4.3 GPU Code Development

The GPU code was developed based on the CPU version presented in the previous section. As said before, CUDA C/C++ was used to write and optimize the program. Each one of the functions, from the calculation of the time step value and the particle number density of the particles, to the assemble of the coefficient matrix and the source term was ported to execute in CUDA kernels. Inside these kernels, through the use of specific keywords and commands, CUDA allowed the assignment of tasks to threads inside the GPU, so independent tasks, which would be executed sequentially, can be parallelized through this. In other words, tasks that do not depend on each other can now be executed at the same time, each one by a different GPU thread.

NVIDIA GPUs are organized in grids. As it is pictured in Figure 11, each grid has a number of blocks, where each block contains a certain number of threads and all of these components have two or three dimensions, depending on the GPU compute capability. The number of grids, blocks per grid and threads per block also varies according to the GPU compute capability. The compute capability of the NVIDIA GeForce GTX 760 used in this work is 3.0, so it allows three dimensions in the components. The maximum number of threads per block is 1024 in the x and y-dimensions, while in the z-dimension this number is lowered to 64. As for the maximum x-dimension of a grid of thread blocks, $2^{31} - 1$ is the corresponding value, while in the y and z-dimensions this number is 65535. These information are important in development of a CUDA efficient program because, when calling a kernel, the number of blocks and threads that will be used has to be specified as configuration parameters. Inside the CUDA kernel, each thread inside a block and each block has its own index, in all dimensions. That is indispensable to identify and assign a task for each block's threads.

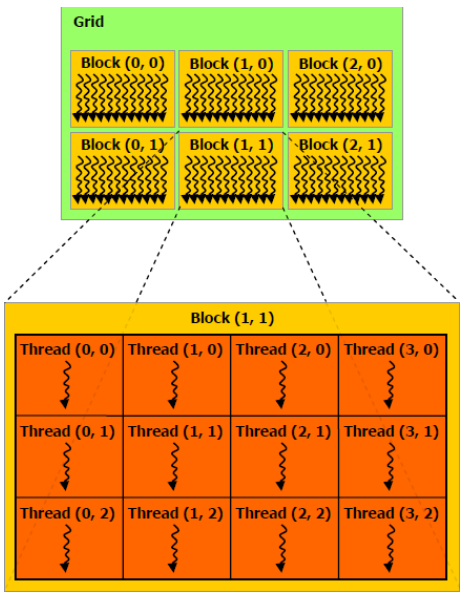


Figure 11: High-level NVIDIA GPU architecture [6]

Regarding the implementations, while the process to parallelize a portion of the functions was quite straightforward, some of them had to be adapted in order to be possible to develop a parallelized version of each of them.

One example of a straightforward function parallelization is the external force calculations, which in the test case used in this work is represented by the gravitational acceleration. Listings 7 and 8 show the CPU and GPU implementation of this calculation, respectively.

Listing 7: CPU code of the external forces calculation

```
1  void ParticlesActions::external_force(Particle2D* particles,
2          double dt, double g, int nump)
3  {
4          Point2D velocity;
5          for (int a = 0; a < nump; a++)
6          {
7                  if((particles)[a].is_fluid())
8                  {
9                          velocity = (particles)[a].get_v();
10                         (particles)[a].set_v(velocity.x,
11                                 velocity.y - g*dt);
12                 }
13         }
14 }
```

Listing 8: GPU code of the external forces calculation

```
1  __global__ void external_force_kernel(int offset,
2          Particle2D* particles, double dt, double g)
3  {
4          unsigned int a = offset +
5                  (blockDim.x * blockIdx.x + threadIdx.x);
6
7          Point2D velocity;
8          if (particles[a].is_fluid()){
9                  velocity = (particles)[a].get_v();
10                 (particles)[a].set_v(velocity.x, velocity.y - g*dt);
11         }
12 }
```

It is possible to note that, in lines 4 and 5 of Listing 8, the calculation of the index of each particle takes into consideration the case when the total particle number of the simulation exceeds the maximum number of threads in a block, so threads in other blocks are triggered and the thread count proceeds from where it stopped in the previous block.

One example where the code had to be adapted, is in the neighboring particle search algorithm, where the optimization proposed by [38] and used in the sequential version of the code in this work had to be undone since each thread in the GPU code is responsible to find the neighbors of one particle. This disables the need of the outer loop in the CPU version. Listing 9 shows the implementation of this parallelized algorithm.

Listing 9: Parallelized search of neighboring particles

```
__global__ void set_nei_kernel(int offset, Particle2D* particles,
        double r2, int nump, int *nei)
{
        unsigned int i = offset +
                (blockDim.x * blockIdx.x + threadIdx.x);

        Point2D Part_j, Part_i;
        double x, y;

        for (int j = 0; j < nump; j++){
                if (!((particles[j].is_wall() &&
                        particles[i].is_wall()) || (i == j)))
                {
                        Part_j = (particles)[j].get_po();
                        Part_i = (particles)[i].get_po();

                        x = Part_j.x - Part_i.x;
                        y = Part_j.y - Part_i.y;

                        if (((x*x) + (y*y)) <= r2)
                        {
                                /* Altered code below */
                                nei[(i * nump) + 0]++;
                                nei[(i * nump) +
                                        nei[(i * nump) + 0]] = j;
                        }
                }
        }
}
```

Regarding the PPE solution, the Cusp library [56] was utilized to solve the linear system equations directly in the GPU. This is a library for sparse linear algebra and graph computations based on Thrust [6]. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. It is developed as an open-source project by NVIDIA Research.

Using this library eliminates the overhead caused by memory operations, such as copying the coefficient matrix and the source term from the device memory back to host memory in order to solve it sequentially in the CPU. However, the great benefit in using Cusp is the GPU-accelerated Krylov methods [57], so, an optimized BiCGStab solver is available.

In order to be possible completely benefit from this solver's capabilities, some specific formats of matrix have to be utilized. One of these is referred as Compressed Sparse Row (CSR) matrix, which only stores non zero elements. The coefficient matrix generated during the generation of PPE is square and sparse and, since sparse matrices contain mostly zeros, storage size is drastically decreased. Also, the access to these non zero elements is much more efficient given its arrays that store row and column indexes to the nonzero elements. The

conversion between the regular dense format to the CSR type is shown in Listing 10 using a function from the Cusparse library [6] called `dense2csr`. The variable `contB[0]` stores the matrix order, `d_nnzPerVector` is the array of nonzero elements per line of the matrix and `descrA` is the matrix description.

Listing 10: Converting matrix from the dense format to CSR

```
(...)

/* Creating pointers to device memory
        that indicate the beginning of every array */

thrust::device_ptr<double> dev_ptr_d_A =
thrust::device_pointer_cast(d_A); /* Pointer to the array of non
        zero element values */

thrust::device_ptr<int> dev_ptr_d_A_RowIndices =
thrust::device_pointer_cast(d_A_RowIndices); /* Pointer to the array
        of row indexes */

thrust::device_ptr<int> dev_ptr_d_A_ColIndices =
thrust::device_pointer_cast(d_A_ColIndices); /* Pointer to the array
of column indexes */

thrust::device_ptr<double> dev_ptr_srcB_d =
thrust::device_pointer_cast(srcB_d); /* Pointer to the array of the
        source term values*/

/* Converting the dense matrix into CSR format */
cusparseSafeCall(cusparseDdense2csr(handle, contB[0], contB[0],
        descrA, srcA_d, contB[0], d_nnzPerVector, d_A,
                d_A_RowIndices, d_A_ColIndices));

(...)
```

After converting the matrix format, the information is assigned to the data structure provided by the Cusp library so the linear system equations can be solved as seen in Listing 11.

Listing 11: Solving $A * x = B$, the PPE, with GPU-optimized code

```
(...)

cusp::csr_matrix<int, double, cusp::device_memory> csrA;
/*
Assigning coefficient matrix information to csrA
*/
cusp::array1d<double, cusp::device_memory>
        x(csrA.num_rows, 0); /* Allocating a result array the
                same size as the matrix order */

cusp::array1d<double, cusp::device_memory> array1dB(dev_ptr_srcB_d,
        dev_ptr_srcB_d + contB[0]); /* Assigning source term
                information to array */

// Configuring the stop criteria:
//  iteration limit = 100
//  relative tolerance = 1e-6
//  absolute tolerance = 0
//  verbose = false
cusp::monitor<double> monitor(array1dB, 100, 1e-16, 0, false);

/* Configuring preconditioner (identity) */
cusp::identity_operator<double, cusp::device_memory>
        M(csrA.num_rows, csrA.num_rows);

/* Solving linear system A*x = b */
cusp::krylov::bicgstab(csrA, x, array1dB, monitor, M);

(...)
```

All the functions developed in the CPU version were successfully ported to CUDA C/C++ to run in the GPU.

5 Case Study

The collapse of a water column has been widely used in the literature to validate and study the various fluid simulation methods. The dam break problem, as it is known, usually is modeled with the water column initially located on the left side of the recipient, against the left vertical wall. When the simulation starts, collapsing water collides on the right vertical wall, which generates fragmentation and coalescence of the fluid itself. A variation of the problem, which is modeled with just taking out the right vertical wall and extending the horizontal boundary (floor) has already been used to verify codes for free-surfaces [58] [59]. Originally, Koshizuka and Oka [14] modeled the dam break problem as shown in Figure 12.

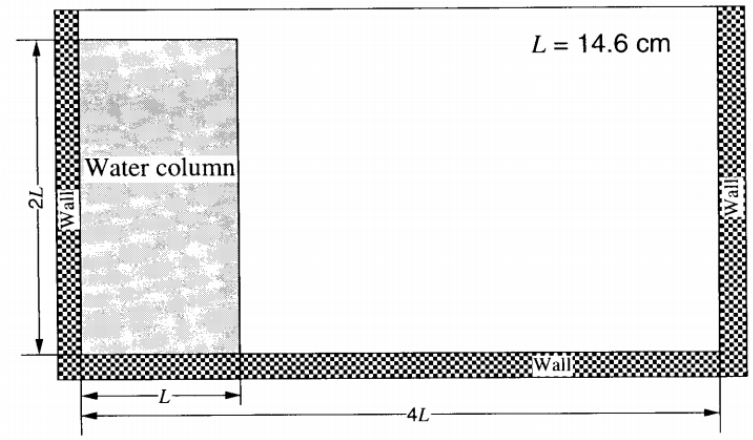


Figure 12: One possible dam break model

Although the dam break model used in this work is not equal to the Koshizuka's, it follows a similar approach. Figure 13 shows the dam break model that was used.

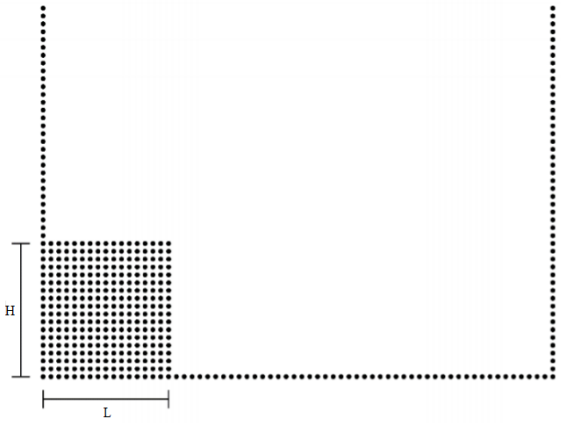


Figure 13: Dam break model employed

Both height H and length L are equal in the tests performed. A similarity between models that is not being explicitly shown is that floor in the model employed is also four times the length L of the water column. The size of the water column varies depending on how many particles the simulation has. The average particle distance is $10^{-2}m$ and the time step of the simulation is $10^{-3}s$. The parameter β is used to judge whether a particle belongs to the free-surface or not, as explained in the MPS chapter and shown in Eq. 17. The value used for β is 0.97 as recommended by the authors of the original MPS method. The kernel size is represented by the parameter r_e and is used as the radius of influence of a particle. Koshizuka and Oka show in [14] that the kernel size should be $< 3.0l_0$, where l_0 is the average particle distance, otherwise the particles will gather near the free-surface. On the other hand, they also show that the discretization of the Laplacian model is more accurate when the kernel size has a higher value. In order to satisfy this, two different kernel sizes were employed, $r_{eP} = 2.1l_0$ and $r_{eLap} = 3.1l_0$.

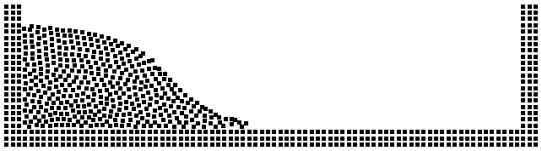
5.1 Results

In this chapter, all the results achieved will be presented and analysed. Firstly, the visual results of two simulations, one with fewer particles and another with a higher number, will be shown. After that, a discussion regarding the floating point and the double precision floating point data types and a numerical analysis of both codes developed (in the CPU and the GPU) will be made in order to show whether there is accuracy and/or stability loss in these implementations. Then, a performance analysis will be done. Memory usage and the functions durations of both versions will be evaluated too. Charts comparing the execution time of the programs with the increase of the total particle number (fluid and wall particles) will be presented. Lastly, the speedup of the GPU version over the CPU version will be calculated, analysed and presented graphically, as well as the frame rate of the GPU simulation.

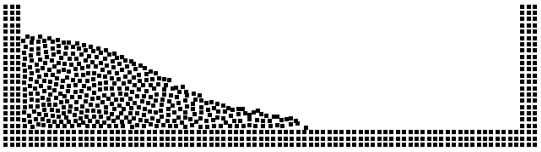
5.1.1 Simulation

Here, two simulations based on the dam break problem presented previously will be exposed in order to show the physical and visual coherence of the simulation. The first one, in Figure 15 has fewer particles, as the length L and height H of the water column is 0.2 m. The time of every screenshot taken is presented too.

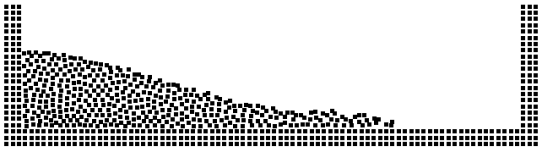
(a) $t = 0.0$s



(a) $t = 0.2$s



(b) $t = 0.3$s



(c) $t = 0.45$s

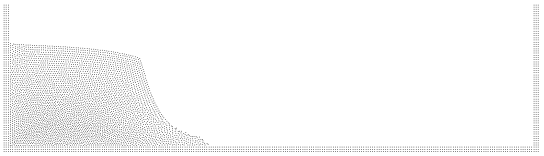
(d) $t = 0.6$s

(e) $t = 0.8$s

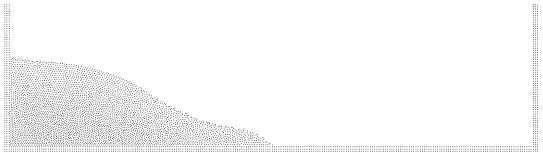
Figure 15: CMPS-HS-HL-ECS visual results for $L = H = 0.2\ m$

For the second simulation in Figure 17, $L = H = 0.6\ m$ and, also, the screenshot times are presented.

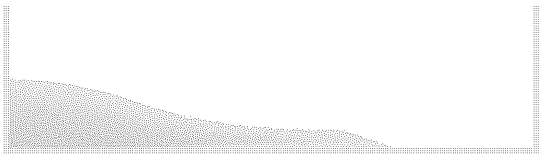
(a) $t = 0.0$s



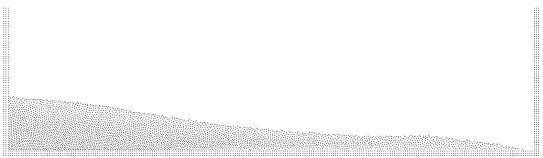
(a) $t = 0.2$s



(b) $t = 0.3$s



(c) $t = 0.45$s



(d) $t = 0.6$s

(e) $t = 0.8$s

Figure 17: CMPS-HS-HL-ECS visual results for $L = H = 0.6\ m$

5.1.2 Numerical Analysis

Before starting the numerical analysis between both CPU and GPU versions, a brief discussion about the differences and the importance of using `float` or `double` data types must be made. In computing, a floating point number is a real number, that is, a number that can contain a fractional part. Essentially computers are capable to represent real numbers by approximations, using complex codes to do it. One of the challenges in programming using the floating point data type is ensuring that these approximations will lead to reasonable results. When accuracy is a relevant requirement of the system, like the one in this work, errors accumulation during the calculations stands out clearly, like in the simulation results of this work. Based in this kind of issues a new data type was made available, the double precision floating point, where the total bits of the internal representation of the number doubles from

32 to 64 bits. The `float` data type normally guarantees the precision of 7 (seven) decimal digits, while `double` generally guarantees 16 (sixteen) decimal digits, appeasing some of the issues regarding precision in numerical calculations. CUDA initially did not give support to double precision floating point causing many complaints, but that changed starting with the CUDA compute capability 1.3, which is present since the Geforce GTX 260 and GTX 280 GPUs.

During the development of the GPU version of the method, strict attention was paid to the numerical precision and not to alter anything related to the data types used in the implementation. This was done so that the precision of the GPU version was maintained (compared to the CPU version). Below, Figure 18 shows the comparison between the methods. For this comparison, the wave front position (its absolute value is represented by x) is being monitored, as a function of time, since the beginning of the dam burst and it is represented by a dimensionless format, (x/L), where L is the water column initial length, which for this test is equal to 0.6 m. As said before, the size of the floor is four times the size of L, which implies that the maximum value x can reach is $4L$ (that being the reason that the information stops when $x/L = 4$ in Figure 18). The time in the chart is represented by a dimensionless format as well: $t\sqrt{g/L}$, where t is the time in seconds $(10^{-3}s)$ and g the gravitational acceleration $(9.8\ m/s^2)$.

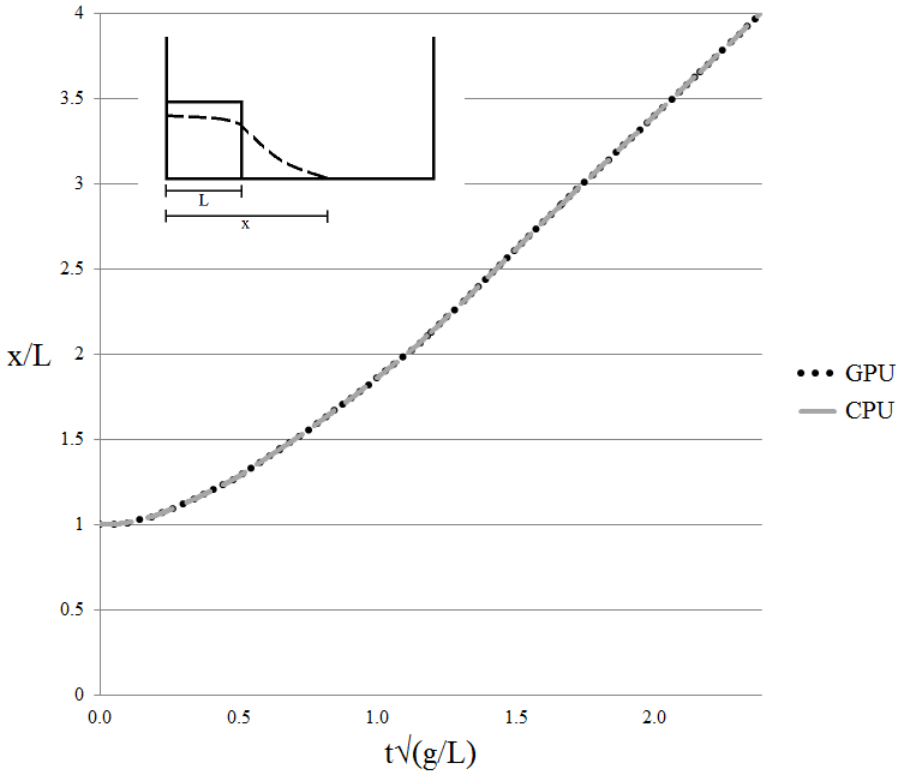


Figure 18: Evolution of the water wave front through dimensionless time

As it can be evidentiated, the GPU version of the method performed exactly the same way as the CPU version did, making it clear that the numerical precision from the CPU version was completely maintained.

5.1.3 Performance Analysis

In this subsection, the performance of the implementations will be exposed and analysed, as well as the percentage of time that was spent by each function and memory usage in both versions. Finally, the absolute time spent by each version will be analysed in order to calculate the speedup reached by the GPU implementation of the method.

5.1.3.1 Functions Duration & Memory Usage

In order to calculate the duration of each function of the CPU version, the Performance and Diagnostics tool of Microsoft Visual Studio 2013 was used. It returned the absolute time in milliseconds and the percentage each function spent executing in the CPU. With this last information it was possible to generate the chart presented in Figure 19. This result was generated by the same simulation scenario presented in subsection 5.1.2.

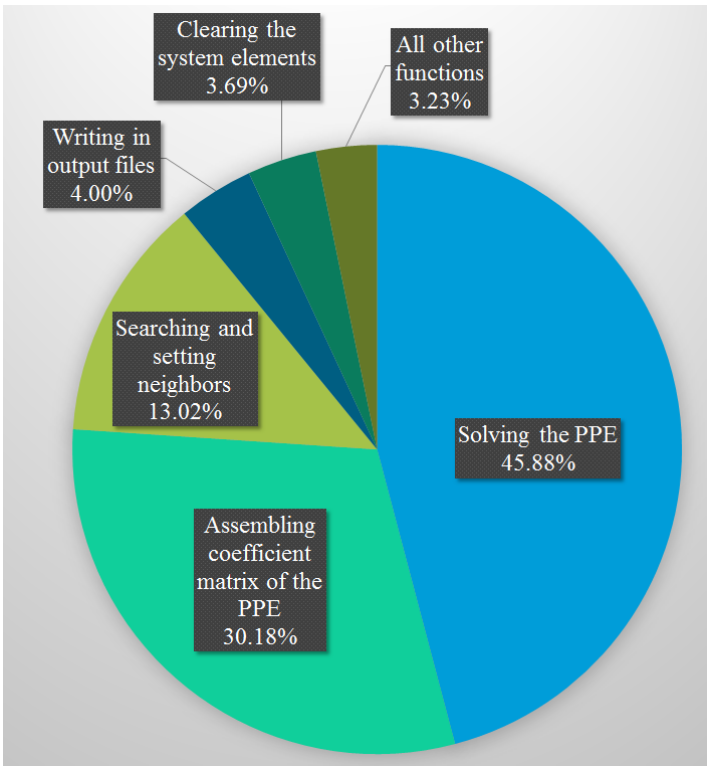


Figure 19: Functions relative execution time in the CPU

It is possible to see that a little more than three quarters of the program's execution is just to assemble and solve the Poisson Pressure equation in order to get the pressure values of each particle. It is noteworthy that a good portion of the execution time (13.02%) is due to the search and setting of the neighboring particles for each particle. The remaining 10.92% is due to all the other calculations and functions of the code, showing the significance of these three functions, which take almost 90% of program's execution time in the CPU.

In this sense, during the implementation of the GPU version special attention was paid to the PPE's assemble and solution. Through the use of the NVIDIA Visual Profiler, absolute and relative times of each kernel function were extracted from the GPU version of the program. Figure 20 shows the relative amount of time each kernel spent executing in the GPU.

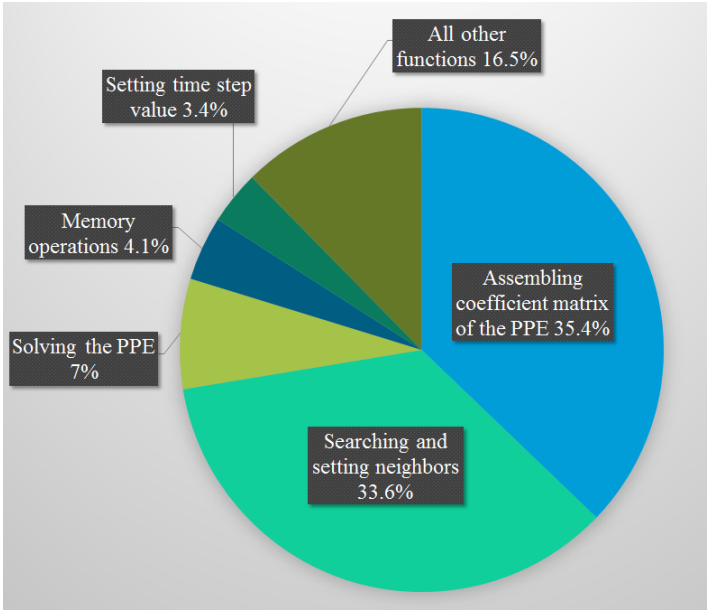


Figure 20: Functions relative execution time in the GPU

As it can be seen, the shrinkage of the relative amount of time taken by the PPE's solution is significant, as the gain for this operation was about 38.88%. This result shows the relevance of parallelizing the solution of the PPE's linear system. Assembling the coefficient matrix of the PPE and the neighborhood search and setting still dominate the execution in the GPU, however, it is possible to note that the execution time of the functions is better distributed than in the CPU version, which shows that the parallelization of the functions that dominated the execution in the CPU are taking less time in their execution, diminishing the bottleneck. All other functions spent less than 2% of the execution time in the GPU each.

In order to evaluate memory usage in the CPU version, the Performance and Diagnostics tool of Visual Studio was also used. As for the GPU version, the CUDA function `cudaMemGetInfo` was called before the end of the code, before the deallocation of the variables and arrays in the GPU memory. Figure 21 shows the amount of memory in MB used in both versions of the code

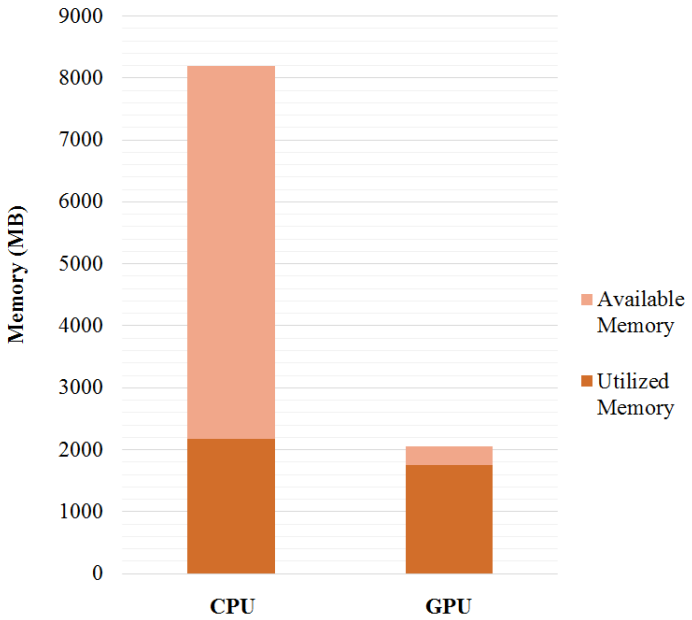


Figure 21: Memory used in both implementations

The memory usage, as it can be seen, is really similar in both versions (2170 MB from the CPU version against 1758 MB from the GPU) even though the CPU has much more available memory than the GPU. This ultimately shows that the GPU version uses less memory than the CPU.

5.1.3.2 Speedups

For this analysis, various test cases were built based on the dam break problem, only increasing the total particle number in the system. All the proportions were kept. For each test case, 100 iterations of the simulation were executed (again in the Microsoft Visual Studio 2013 Performance and Diagnostics tool for the CPU version and the NVIDIA Visual Profiler for the GPU version) in order to get the time spent on the GPU by the GPU implementation, and the time spent on the CPU by the CPU implementation. After that, by dividing the absolute execution time in the CPU by the GPU's, it was possible to obtain the speedup provided by the GPU version.

In each test, the size of the water column was increased in 0.1 m from 0.3 m to 0.7 m, such in length as in height (keeping the column and floor proportions). The execution time of each scenario and its total particle number, can be seen side by side in the chart shown in Figure 22.

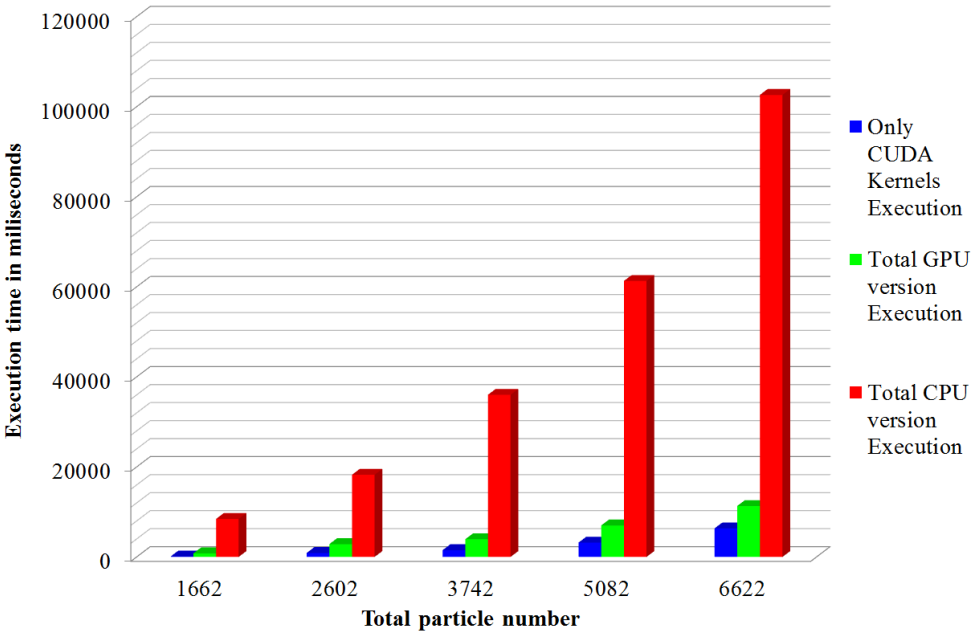


Figure 22: Execution time versus total particle number in the system

It is possible to see that the CPU execution time is clearly much higher than the GPU execution time, and even higher when only considering the CUDA kernel executions, in other words, without taking into account transfer operations between host and device memory, in any direction (any `cudaMemCpy` operation).

Now, with these information, it is possible to calculate how many times the GPU version of the implementation is faster than the CPU version (the absolute speedup) in each test case built. The chart presented in Figure 23 shows the speedup in each test, depending on the total particle number.

The speedup ranges from 6.41 to 10.69, and it is possible to calculate the average speedup in this set of scenarios: a considerable 8.82 times. Oddly, as the total particle number (system size) increased the speedup did not increase, as expected in a GPU-optimized system. In order to investigate this issue, a comparison of execution times was made, as exhibited in Figure 24. It shows the absolute time spent (in milliseconds) of the most time consuming functions of the execution in each test case.

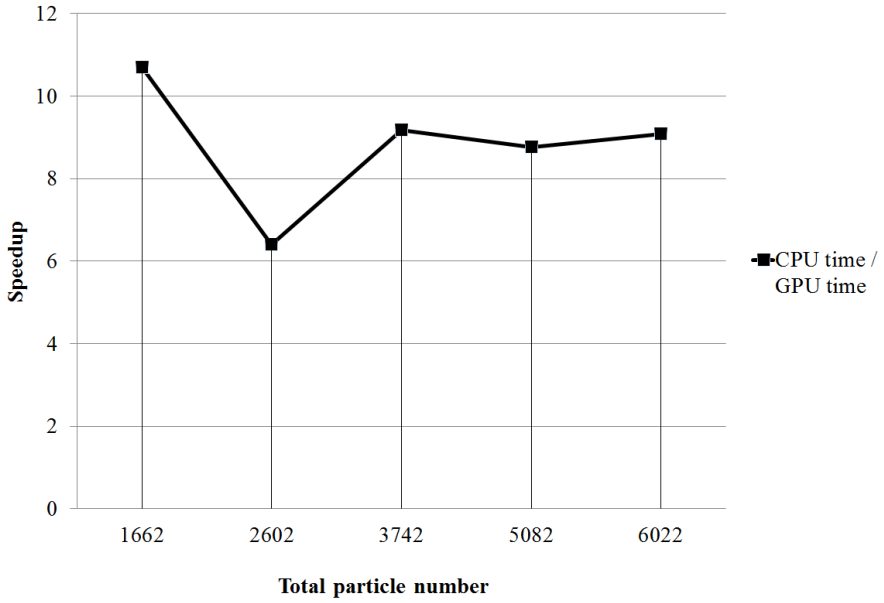


Figure 23: Speedup of the GPU version over the CPU version

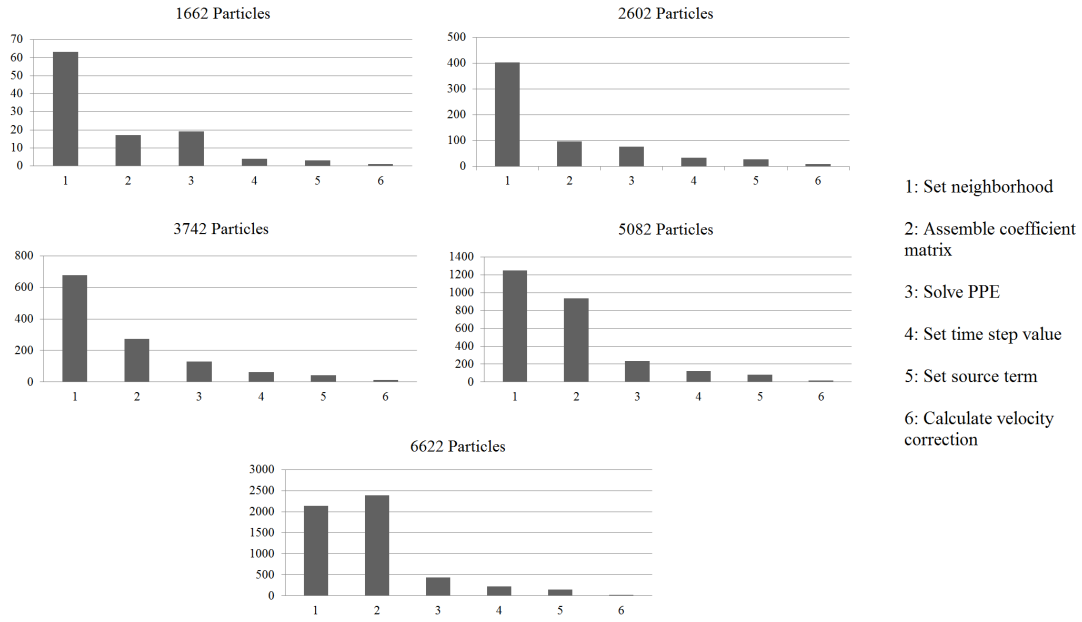


Figure 24: Execution time in milliseconds of each function for each test case on GPU

It is noted that, as the total particle number increases, the time for assembling the coefficient matrix (function 2) increases more rapidly than the other functions. Since the size of the matrix is the total number of fluid particles in the system squared (NFP^2), which is the majority of particles in the system, any minor change or issue in the implementation can cause an expressive change in the execution time or even problems in memory usage due to the high quantity of data. Certainly, this is preventing the GPU-accelerated system from achieving the higher speedup values it is capable of.

Even though this issue occurred, the achieved speedups enable entering the field of real-time simulation or at least interactive applications (depending on the number of particles) since more than one simulation frame is being generated within a second [60] [61]. Taking the last scenario with 6622 particles, where the total execution time of 100 iterations in the GPU is 11306 ms, approximately one frame is generated every 113.06 ms, which gives a rate of about 8.85 frames per second (fps) being generated. It is noteworthy that the speedup is shortened by the memory copy operations, i.e. when copying the particles' information from the device memory to the host memory to build the simulation. These operations, specifically, are not necessary when exhibiting the simulation in real-time using some graphics library, such as OpenGL [62] or DirectX [63]. Figure 25 shows the frame rate for each test case considering all operations in the code of the GPU version.

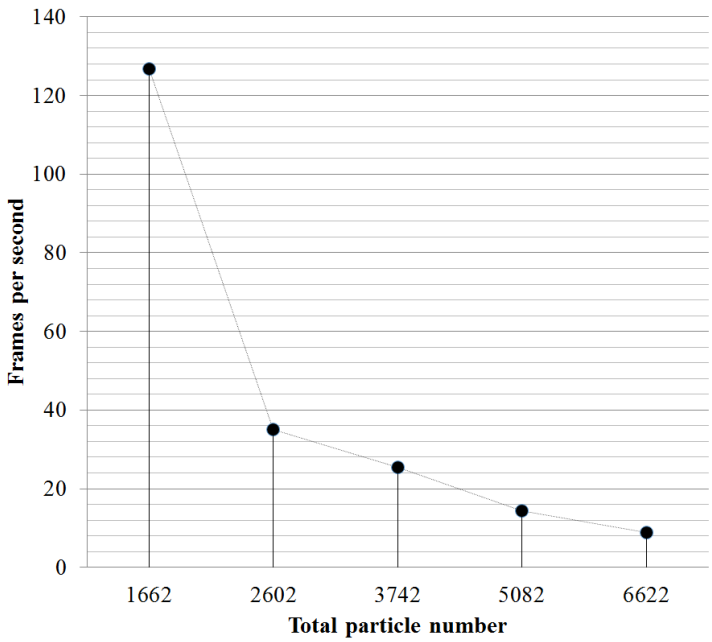


Figure 25: Frame rate as the total particle number increases on GPU

As said, the memory copy operation from device to host memory is not needed when the simulation is being displayed in real-time, so, depending on how big the rendering overhead is, the frame rates presented here can reach higher values for these same amount of particles in the simulations when using a graphics library for the exhibition.

6 Conclusion

The various works making efforts to improve further and further the stability and accuracy of the MPS method, show the complexity of this task, the importance of the method to the community and the great potential it has to simulate, increasingly more realistically, incompressible fluid flows. Regarding the MPS optimization, it has been gradually evolving through the combination of increasingly more sophisticated algorithms to minimize the communication during the solution of the system of linear equations (PPE) and to reorganize the system's coefficient matrix. Another aspect that helps the evolution of the method's parallelization is the hardware development. Even more powerful GPUs are being manufactured every year, constantly increasing parallelized systems performance.

6.1 Contributions

After a research in the literature, it was observed that the CMPS-HS-HL-ECS method, specifically, has not yet been parallelized in order to obtain a higher performance in GPUs. This graduation work provides a stable and physically coherent free-surface incompressible fluid flow simulation method that is GPU-accelerated with speedups ranging from 6.41 to 10.69 times and a frame rate of, approximately, 8.85 in a system with 6622 particles.

Throughout the development of this work, one paper was published in the Congress on Numerical Methods in Engineering (CMN 2015) [27]. A book chapter to be published in 2016 about meshless methods was written while working alongside with the University of São Paulo (USP) [28].

6.2 Future Works

There is an interesting number of possibilities for future developments of this work. One of them is the extension of the method to one more dimension, making it three dimensional.

The algorithm for the search of neighboring particles can still experience some great improvement, both in the CPU and GPU versions of the code, with the implementation of cell grids in order to narrow the search for neighbors to the closest particles to a target particle i [30] [47]. Another improvement is in the calculation of the time step duration (in seconds) in which the GPU version of the code can be accelerated, in this case, through a parallelized reduction operation, which, although easy to implement in CUDA, is hard to do it correctly. Surely, refinements in every small part of the code, mainly where improvement possibilities had not been seen, will lead to more a optimized version; this is considered the path to a notable real-time simulation.

Finally, the issue of the coefficient matrix assembling function presented in the previous section most likely forbade that the system could be fully explored with respect to its size, restraining the increase of the total particle number in order to see the results for larger systems. It also prevented that the speedup enabled by the GPU version could achieve even higher values as the total particle number increased. So, further investigation in the assembly of the PPE's coefficient matrix implementation and in the program's memory usage is necessary in order to enhance even more the robustness of the system, allowing the use of more particles and the achievement of higher speedup values as the total particle number increases.

References

[1] Byung-Hyuk Lee, Jong-Chun Park, Moo-Hyun Kim, and Sung-Chul Hwang. Step-by-step improvement of mps method in simulating violent free-surface motions and impact-loads. *Computer methods in applied mechanics and engineering*, 200(9):1113–1125, 2011.

[2] Tsunakiyo Iribe, Toshimitsu Fujisawa, and Seiichi Koshizuka. Reduction of communication in parallel computing of particle method for flow simulation of seaside areas. *Coastal Engineering Journal*, 52(04):287–304, 2010.

[3] Abbas Khayyer. *Improved particle methods by refined models for free-surface fluid flows*. PhD thesis, Kyoto University, 2008.

[4] Abbas Khayyer and Hitoshi Gotoh. Modified moving particle semi-implicit methods for the prediction of 2d wave impact pressure. *Coastal Engineering*, 56(4):419–440, 2009.

[5] H Gotoh. Advanced particle methods for accurate and stable computation of fluid flows. *Frontiers of Discontinuous Numerical Methods and Practical Simulations in Engineering and Disaster Prevention*, page 113, 2013.

[6] NVIDIA. Cuda zone — nvidia developer. https://developer.nvidia.com/cuda-zone. Accessed: 2016-01-09.

[7] PW Cleary, M Prakash, and J Ha. Novel applications of smoothed particle hydrodynamics (sph) in metal forming. *Journal of materials processing technology*, 177(1):41–48, 2006.

[8] Ted Belytschko, Yury Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: an overview and recent developments. *Computer methods in applied mechanics and engineering*, 139(1):3–47, 1996.

[9] Andrew A Johnson and Tayfun E Tezduyar. Advanced mesh generation and update methods for 3d flow simulations. *Computational Mechanics*, 23(2):130–143, 1999.

[10] Eugenio Oñate, Sergio R Idelsohn, Facundo Del Pin, and Romain Aubry. The particle finite element method—an overview. *International Journal of Computational Methods*, 1(02):267–307, 2004.

[11] Pascal-Jean Frey and Frédéric Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer methods in applied mechanics and engineering*, 194(48):5068–5082, 2005.

[12] Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

[13] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

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

[15] Abbas Khayyer, Hitoshi Gotoh, and Songdong Shao. Development of cisph method for accurate water-surface tracking in plunging breaker. *Proceedings of Coastal Research, Engineering*, 54:16–20, 2007.

[16] Songdong Shao and Edmond YM Lo. Incompressible sph method for simulating newtonian and non-newtonian flows with a free surface. *Advances in Water Resources*, 26(7):787–800, 2003.

[17] Seiichi Koshizuka, Hiroaki Tamako, and Yoshiaki Oka. A particle method for incompressible viscous flow with fluid fragmentation. *Comput. Fluid Dynamics J.*, 1995.

[18] Seichii Koshizuka. A particle method for incompressible viscous flow with fluid fragmentation. *Comput. Fluid Dynamics J.*, 4:29–46, 1995.

[19] S Tokura. Comparison of particle methods: Sph and mps. *13th International LS-DYNA Users Conference*, 2014.

[20] Hirotada Hashimoto, Nicolas Grenier, and David Le Touzé. Comparisons of mps and sph methods: Forced roll test of a two-dimensional damaged car deck. *Proceedings of Japan Society of Naval Architecture & Ocean Engineering*, (17):1–4, 2013.

[21] Antonio Souto-Iglesias, Fabricio Macià, Leo M González, and Jose L Cercos-Pita. On the consistency of mps. *Computer Physics Communications*, 184(3):732–745, 2013.

[22] YL Ng, KC Ng, and MZ Yusoff. The study of pressure source term in moving particle semi-implicit (mps). In *IOP Conference Series: Earth and Environmental Science*, volume 16, page 012152. IOP Publishing, 2013.

[23] Abbas Khayyer and Hitoshi Gotoh. Development of cmps method for accurate water-surface tracking in breaking waves. *Coastal Engineering Journal*, 50(02):179–207, 2008.

[24] Abbas Khayyer and Hitoshi Gotoh. A higher order laplacian model for enhancement and stabilization of pressure calculation by the mps method. *Applied Ocean Research*, 32(1):124–131, 2010.

[25] Abbas Khayyer and Hitoshi Gotoh. Enhancement of stability and accuracy of the moving particle semi-implicit method. *Journal of Computational Physics*, 230(8):3093–3118, 2011.

[26] Abbas Khayyer and Hitoshi Gotoh. A 3d higher order laplacian model for enhancement and stabilization of pressure calculation in 3d mps-based simulations. *Applied Ocean Research*, 37:120–126, 2012.

[27] André L Vieira e Silva, Mozart W Almeida, Caio J Brito, Veronica Teichrieb, José M Barbosa, and Cesar Salhua. A qualitative analysis of fluid simulation using a sph variation. 2015.

[28] G. Assi, H. Brinati, M. Conti, and M. Szajnbok. Meshless methods. In Gustavo Assi, Hernani Brinati, Mardel Conti, and Moyses Szajnbok, editors, *Applied Topics in Marine Hydrodynamics*. São Paulo, 2016.

[29] E Rustico, G Bilotta, G Gallo, A Hérault, C Del Negro, and RA Dalrymple. A journey from single-gpu to optimized multi-gpu sph with cuda. In *7th SPHERIC Workshop*, 2012.

[30] XiaoSong Zhu, Liang Cheng, Lin Lu, and Bin Teng. Implementation of the moving particle semi-implicit method on gpu. *SCIENCE CHINA Physics, Mechanics & Astronomy*, 54(3):523–532, 2011.

[31] G. Chen, Y. Onishi, L. Zheng, and T. Sasaki. *Frontiers of Discontinuous Numerical Methods and Practical Simulations in Engineering and Disaster Prevention*. Taylor & Francis Group, London, 8 2013.

[32] Takumi Kawahara and Yoshiaki Oka. Ex-vessel molten core solidification behavior by moving particle semi-implicit method. *Journal of Nuclear Science and Technology*, 49(12):1156–1164, 2012.

[33] Seiichi Koshizuka, Hirokazu Ikeda, and Yoshiaki Oka. Numerical analysis of fragmentation mechanisms in vapor explosions. *Nuclear engineering and design*, 189(1):423–433, 1999.

[34] Xiaosong Sun, Mikio Sakai, Kazuya Shibata, Yoshikatsu Tochigi, and Hiroaki Fujiwara. Numerical modeling on the discharged fluid flow from a glass melter by a lagrangian approach. *Nuclear Engineering and Design*, 248:14–21, 2012.

[35] Kazuya Shibata, Seiichi Koshizuka, and Yoshiaki Oka. Numerical analysis of jet breakup behavior using particle method. *Journal of nuclear science and technology*, 41(7):715–722, 2004.

[36] RH Chen, WX Tian, GH Su, SZ Qiu, Yuki Ishiwatari, and Yoshiaki Oka. Numerical investigation on coalescence of bubble pairs rising in a stagnant liquid. *Chemical Engineering Science*, 66(21):5055–5063, 2011.

[37] Asril Pramutadi Andi Mustari, Yoshiaki Oka, Masahiro Furuya, Watanabe Takeo, and Ronghua Chen. 3d simulation of eutectic interaction of pb–sn system using moving particle semi-implicit (mps) method. *Annals of Nuclear Energy*, 81:26–33, 2015.

[38] Seiichi Koshizuka, Atsushi Nobe, and Yoshiaki Oka. Numerical analysis of breaking waves using the moving particle semi-implicit method. *International Journal for Numerical Methods in Fluids*, 26(7):751–769, 1998.

[39] Masahiro Kondo, Kentaro Suto, Mikio Sakai, and Seiichi Koshizuka. Incompressible free surface flow analysis using moving particle semi-implicit method.

[40] Han Young Yoon, Seiichi Koshizuka, and Yoshiaki Oka. A particle–gridless hybrid method for incompressible flows. *International Journal for Numerical Methods in Fluids*, 30(4):407–424, 1999.

[41] Franz Durst, D Miloievic, and Bernhard Schönung. Eulerian and lagrangian predictions of particulate two-phase flows: a numerical study. *Applied Mathematical Modelling*, 8(2):101–115, 1984.

[42] B Ataie-Ashtiani and Leila Farhadi. A stable moving-particle semi-implicit method for free surface flows. *Fluid Dynamics Research*, 38(4):241–256, 2006.

[43] Guangtao Duan and Bin Chen. Stability and accuracy analysis for viscous flow simulation by the moving particle semi-implicit method. *Fluid Dynamics Research*, 45(3):035501, 2013.

[44] Marcio Michiharu Tsukamoto, Kazuo Nishimoto, and Takayuki Asanuma. Development of particle method representing floating bodies with highly non-linear waves. In *18th International Congress of Mechanical Engineering, COBEM*, 2005.

[45] H Ikari and H Gotoh. Parallelization of mps method for 3d wave analysis. In *Advances in Hydro-science and Engineering, 8th International Conference on Hydro-science and Engineering (ICHE)*, 2008.

[46] Hitoshi Gotoh, Abbas Khayyer, Hiroyuki Ikari, and Chiemi Hori. 3d-cmps method for improvement of water surface tracking in breaking waves. In *Proceedings of 4th SPHERIC Workshop. Nantes, France,:[sn]*, pages 265–272. World Scientific, 2009.

[47] Chiemi Hori, Hitoshi Gotoh, Hiroyuki Ikari, and Abbas Khayyer. Gpu-acceleration for moving particle semi-implicit method. *Computers & Fluids*, 51(1):174–183, 2011.

[48] Davi T. Fernandes. *Implementação de framework computacional de paralelização híbrida do Moving Particle Semi-implicit Method para modelagem de fluidos incompressíveis*. PhD thesis, Universidade de São Paulo, 2013.

[49] Yukihito Suzuki, Seiichi Koshizuka, and Yoshiaki Oka. Hamiltonian moving-particle semi-implicit (hmps) method for incompressible fluid flows. *Computer Methods in Applied Mechanics and Engineering*, 196(29):2876–2894, 2007.

[50] Intel Processor i7 4790 Specifications. http://ark.intel.com/products/80806/Intel-Core-i7-4790-Processor-8M-Cache-up-to-4_00-GHz. Accessed: 2016-01-15.

[51] NVIDIA GPU GeForce GTX 760 Specifications. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-760/specifications. Accessed: 2016-01-15.

[52] Will J Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit*. Kitware, 2004.

[53] James Ahrens, Berk Geveci, and Charles Law. 36 paraview: An end-user tool for large-data visualization. *The Visualization Handbook*, page 717, 2005.

[54] GetFEM++ project. Gmm++ library, 2015. Version 5.0.

[55] Yousef Saad. Ilut: A dual threshold incomplete lu factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

[56] Steven Dalton, Nathan Bell, Luke Olson, and Michael Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations, 2015. Version 0.5.1.

[57] Henk A Van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press, 2003.

[58] Balasubramaniam Ramaswamy and Mutsuto Kawahara. Lagrangian finite element analysis applied to viscous free surface fluid flow. *International Journal for Numerical Methods in Fluids*, 7(9):953–984, 1987.

[59] Ryszard Staroszczyk. Simulation of dam-break flow by a corrected smoothed particle hydrodynamics method. *Archives of Hydro-Engineering and Environmental Mechanics*, 57(1):61–79, 2010.

[60] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on gpus. In *Computer Graphics International*, pages 63–70. SBC Petropolis, 2007.

[61] Nobuhiko Mukai, Masashi Nakagawa, and Makoto Kosugi. Real-time blood vessel deformation with bleeding based on particle method. *Studies in health technology and informatics*, 132:313–315, 2007.

[62] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

[63] Bradley Bargen and Peter Donnelly. *Inside DirectX: in-depth techniques for developing high-performance multimedia applications*. Microsoft Press, 1998.