Investigo
Universidade de Vigo

Biblioteca Universitaria
Área de Repositorio Institucional

Accepted Manuscript

Please cite this article as: Crespo, A. J. C., Domínguez, J. M., Rogers, B. D., Gómez-Gesteira, M., Longshaw, S., Canelas, R., et al. (2015). DualSPHysics: Open-source parallel CFD solver based on smoothed particle hydrodynamics (SPH). Computer Physics Communications, 187, 204-216. doi:10.1016/j.cpc.2014.10.004

Link to published version: https://doi.org/10.1016/j.cpc.2014.10.004

General rights:

© 2014 Elsevier B.V. This article is distributed under the terms and conditions of the Creative Commons Attribution-Noncommercial-NoDerivatives (CC BY-NC-ND) licenses https://creativecommons.org/licenses/by-nc-nd/4.0/

*Manuscript

DualSPHysics: open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH)

A.J.C. Crespo[1]*, J.M. Domínguez[1], B.D. Rogers[2], M. Gómez-Gesteira[1], S. Longshaw[2], R. Canelas[3], R. Vacondio[4], A. Barreiro[1], O. García-Feal[1]

[1]EPHYSLAB Environmental Physics Laboratory, Universidade de Vigo, Spain

[2]Modelling and Simulation Centre (MaSC), School of Mechanical, Aerospace and Civil Engineering (MACE), University of Manchester, United Kingdom.

[3]CEHIDRO, Instituto Superior Tecnico, Lisbon, Portugal.

[4]Department of Civil Environmental Engineering, University of Parma, Parma, Italy.

E-mail addresses: alexbexe@uvigo.es (A.J.C. Crespo), jmdominguez@uvigo.es (J.M. Dominguez), benedict.rogers@manchester.ac.uk (B.D. Rogers), mggesteira@uvigo.es (M. Gómez-Gesteira), Stephen.Longshaw@manchester.ac.uk (S. Longshaw), ricardo.canelas@ist.utl.pt (R. Canelas), renato.vacondio@unipr.it (R. Vacondio), anxo.barreiro@uvigo.es (A. Barreiro), orlando@uvigo.es (O. García-Feal)

* Corresponding author: A.J.C. Crespo, alexbexe@uvigo.es, +34 988387425

Abstract

DualSPHysics is a hardware accelerated Smoothed Particle Hydrodynamics code developed to solve free-surface flow problems. DualSPHysics is an open-source code developed and released under the terms of GNU General Public License (GPLv3). Along with the source code, a complete documentation that makes easy the compilation and execution of the source files is also distributed. The code has been shown to be efficient and reliable. The parallel power computing of Graphics Computing Units (GPUs) is used to accelerate DualSPHysics by up to two orders of magnitude compared to the performance of the serial version.

Keywords: SPH, Free-surface, Meshfree methods, GPU.

Program summary

Program Title: DualSPHysics
Journal Reference:
Catalogue identifier:
Licensing provisions: GNU General Public License (GPL), http://www.gnu.org/licenses/
Programming language: C++ and CUDA
Computer: Tested on CPU Intel X5500 and GPUs: GTX 480, GTX 680, Tesla K20 and GTX Titan
Operating system: Any system with a C++ and NVCC compiler, tested on Linux distribution Centos 6.5
CUDA: Tested on versions 4.0, 4.1, 4.2, 5.0 and 5.5 with driver version 331.38.
RAM: Tens of MB to several GB, depending on problem
Number of processors used: Different threads of CPU or number of cores of GPU
Classification: 4.12
Nature of problem: The DualSPHysics code has been developed to study free-surface flows requiring high computational cost
Solution method: DualSPHysics is an implementation of Smoothed Particle Hydrodynamics, which is a Lagrangian meshless particle method

Running time: 6 hours on 8 processors of Intel X5500 (15 min on GTX Titan) for the dam-break case with 1 million particles simulating 1.5s of physical time (more than 26,000 steps)

1. Introduction

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian meshless method that is increasingly used for an extensive range of applications within the field of Computational Fluid Dynamics (CFD) [1] where particles represent the flow, interact with structures and can exhibit large deformation with moving boundaries. The SPH technique is approaching a mature stage, with continuing improvements and modifications meaning the accuracy, stability and reliability of the model are reaching an acceptable level for practical engineering applications.

SPHysics is an open-source SPH model developed by researchers at the Johns Hopkins University (US), the University of Vigo (Spain), the University of Manchester (UK) and the University of Rome, La Sapienza. The software is available to download from www.sphysics.org. A complete guide of the FORTRAN code is found in [2,3]. The SPHysics code was validated for different problems of wave breaking [4], dam-break behaviour [5], interaction with coastal structures [6] or with a moving breakwater [7]. A shallow water version was also developed [8,9]. Although SPHysics allows modelling problems with high resolution, the main problem for the application to real engineering problems is its high computational cost, therefore SPHysics is rarely applied to large domains. Hardware acceleration and parallel computing are required to make codes such as SPHysics more useful and versatile.

Supercomputers are expensive to buy and maintain and practitioners usually do not have access to classical High Performance Computing (HPC) facilities. Graphics Processing Units (GPUs) appear as a cheap alternative to accelerate numerical models. GPUs are designed to manage huge amounts of data and their computing power has developed in recent years to be much faster than conventional central processing units (CPUs) in certain scenarios. NVIDIA's Compute Unified Device Architecture (CUDA) is a parallel programming framework and language for GPU computing using extensions to the C/C++ language. Researchers and engineers of different fields are achieving high speedups implementing their codes with the CUDA language. The computing power of GPUs can be also applied to SPH methods [10], where the algorithmic structure inherently exposes parallelism.

The code DualSPHysics has been developed by starting from the FORTRAN SPH formulation implemented in SPHysics. This code is considered robust and reliable but not optimised for large simulations. DualSPHysics is implemented in C++ and CUDA and is designed to launch simulations either on multiple CPUs using OpenMP or on a GPU. The GPU portion of DualSPHysics [11] implements the most appropriate parallelisation to maximise speedup during particle interaction computation. The first

rigorous validations of DualSPHysics on GPUs were presented in [12] and the code has been recently applied to compute forces exerted by large waves on the urban furniture of a realistic promenade [13], to study the run-up on a real armour block coastal breakwater [14] and to simulate large waves generated by landslide events [15].

DualSPHysics is an open-source code developed and redistributed under the terms of the GNU General Public License as published by the Free Software Foundation. Along with the source code, documentation that describes the compilation and execution of the source files is also distributed. One of the purposes of this code is to encourage other researchers to try SPH. Most downloads to date have been registered by researchers and students that have conducted their research on fluid dynamics using Smoothed Particle Hydrodynamics models. Furthermore, the code has been downloaded not only by students and researchers from universities and institutes but also by companies with industrial interests. The increasing interest in SPH is indicated by the appearance of other important SPH solvers such as the open source JOSEPHINE [16], GPUSPH [17], AQUAgpusph [18], ISPH [19], GADGET [20], pysph [21] or closed source SPH-flow [22], SimPARTIX [23], Pasimodo [24].

In the following sections, the SPH formulation implemented in DualSPHysics and associated optimization techniques are described. Sections describing how to compile and run the code are also provided and finally, several study cases are presented including comparison with experimental data and a performance analysis.

2. Smoothed Particle Hydrodynamics Method

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian meshless method. The technique discretises a continuum using a set of material points or *particles*. When used for the simulation of fluid dynamics, the discretised Navier-Stokes equations are locally integrated at the location of each of these particles, according to the physical properties of surrounding particles. The set of neighbouring particles is determined by a distance based function, either circular (two-dimensional) or spherical (three-dimensional), with an associated characteristic length or *smoothing length* often denoted as h. At each time-step new physical quantities are calculated for each particle, and they then move according to the updated values.

The conservation laws of continuum fluid dynamics are transformed from their partial differential form to a form suitable for particle based simulation using integral equations based on an interpolation function, which gives an estimate of values at a specific point. Typically this function is referred to as the kernel function *(W)* and can take different forms, with the most common being cubic or quintic. In all cases however, it is designed to represent a function $F(\mathbf{r})$ defined in $\mathbf{r'}$ by the integral approximation

$$F(\mathbf{r}) = \int F(\mathbf{r'})W(\mathbf{r}-\mathbf{r'},h)d\mathbf{r'} \quad (1)$$

The smoothing kernel must fulfil several properties [25,26], such as positivity inside a defined zone of interaction, compact support, normalization and monotonically decreasing value with distance and differentiability. For a more complete description of SPH, the reader is referred to [27,28].

The function F in Eq. (1) can be approximated in a non-continuous, discrete form based on the set of particles. In this case the function is interpolated at a particle (a) where a summation is performed over all the particles that fall within its region of compact support, as defined by the smoothing length h

$$F(\mathbf{r}_a) \approx \sum_b F(\mathbf{r}_b) W(\mathbf{r}_a - \mathbf{r}_b, h) \Delta v_b \tag{2}$$

where Δv_b is the volume of a neighbouring particle (b). If $\Delta v_b = m_b / \rho_b$, with m and ρ being the mass and the density of particle b respectively then Eq. (2) becomes

$$F(\mathbf{r}_a) \approx \sum_b F(\mathbf{r}_b) \frac{m_b}{\rho_b} W(\mathbf{r}_a - \mathbf{r}_b, h) \tag{3}$$

2.1 The Smoothing Kernel

Performance of an SPH model depends heavily on the choice of the smoothing kernel. Kernels are expressed as a function of the non-dimensional distance between particles (q), given by $q = r/h$, where r is the distance between any two given particles a and b and the parameter h (the smoothing length) controls the size of the area around particle a in which neighbouring particles are considered. Within DualSPHysics, the user is able to choose from one of the following kernel definitions:

a) Cubic spline

$$W(r,h) = \alpha_D \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3 & 1 \le q \le 2 \\ 0 & q \ge 2 \end{cases} \tag{4}$$

where α_D is equal to $10/7\pi h^2$ in 2-D and $1/\pi h^3$ in 3-D.

The tensile correction method, proposed by Monaghan [29], is only actively used in the cases of a kernel whose first derivative goes to zero with the particle distance q.

b) Quintic [30]

$$W(r,h) = \alpha_D \left(1 - \frac{q}{2}\right)^4 (2q+1) \qquad 0 \le q \le 2 \tag{5}$$

where α_D is equal to $7/4\pi h^2$ in 2-D and $21/16\pi h^3$ in 3-D.

In the text that follows, only kernels with an influence domain of $2h$ ($q \le 2$) are considered.

2.2. Momentum Equation

The momentum conservation equation in a continuum is

$$\frac{d\boldsymbol{v}}{dt} = -\frac{1}{\rho}\nabla P + \boldsymbol{g} + \boldsymbol{\Gamma} \tag{6}$$

where $\boldsymbol{\Gamma}$ refers to dissipative terms and $\boldsymbol{g}$ is gravitational acceleration. DualSPHysics offers different options for including the effects of dissipation.

2.2.1. Artificial Viscosity

The artificial viscosity scheme, proposed by Monaghan [25], is a common method within fluid simulation using SPH due primarily to its simplicity. In SPH notation, Eq. 6 can be written as

$$\frac{d\boldsymbol{v}_a}{dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} + \Pi_{ab} \right) \nabla_a W_{ab} + \boldsymbol{g} \tag{7}$$

where P_k and ρ_k are the pressure and density that correspond to particle k (as evaluated at a or b). The viscosity term Π_{ab} is given by

$$\Pi_{ab} = \begin{cases} \dfrac{-\alpha \overline{c_{ab}} \mu_{ab}}{\overline{\rho_{ab}}} & \boldsymbol{v}_{ab} \cdot \boldsymbol{r}_{ab} < 0 \\ 0 & \boldsymbol{v}_{ab} \cdot \boldsymbol{r}_{ab} > 0 \end{cases} \tag{8}$$

where $r_{ab} = r_a - r_b$ and $v_{ab} = v_a - v_b$ with r_k and v_k being the particle position and velocity respectively. $\mu_{ab} = h\boldsymbol{v}_{ab} \cdot \boldsymbol{r}_{ab} / (r_{ab}^2 + \eta^2)$, $\overline{c_{ab}} = 0.5(c_a + c_b)$ is the mean speed of sound, $\eta^2 = 0.01h^2$ and α is a coefficient that needs to be tuned in order to introduce the proper dissipation.

2.2.2. Laminar viscosity and Sub-Particle Scale (SPS) Turbulence

Laminar viscous stresses in the momentum equation can be expressed as [31]

$$\left(\upsilon_0 \nabla^2 \boldsymbol{v}\right)_a = \sum_b m_b \left(\frac{4\upsilon_0 \, \boldsymbol{r}_{ab} \cdot \nabla_a W_{ab}}{(\rho_a + \rho_b)(r_{ab}^2 + \eta^2)} \right) \boldsymbol{v}_{ab} \tag{9}$$

where υ_o is kinematic viscosity (typically 10^{-6} m^2s for water). In SPH discrete notation this can be expressed as

$$\frac{d\boldsymbol{v}_a}{dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} \right) \nabla_a W_{ab} + \boldsymbol{g} + \sum_b m_b \left(\frac{4\upsilon_0 \boldsymbol{r}_{ab} \cdot \nabla_a W_{ab}}{(\rho_a + \rho_b)(r_{ab}^2 + \eta^2)} \right) \boldsymbol{v}_{ab} \tag{10}$$

The concept of the Sub-Particle Scale (SPS) was first described by [32] to represent the effects of turbulence in their Moving Particle Semi-implicit (MPS) model. The momentum conservation equation is defined as

$$\frac{d\boldsymbol{v}}{dt} = -\frac{1}{\rho}\nabla P + \boldsymbol{g} + \upsilon_0 \nabla^2 \boldsymbol{v} + \frac{1}{\rho}\nabla \cdot \bar{\boldsymbol{\tau}} \tag{11}$$

where the laminar term is treated as per Eq. 9 and $\vec{\tau}$ represents the SPS stress tensor. Favre-averaging is needed to account for compressibility in weakly compressible SPH [4] where eddy viscosity assumption is used to model the SPS stress tensor with Einstein notation for the shear stress component in directions i and j $\frac{\vec{\tau}_{ij}}{\rho} = \nu_t\left(2S_{ij} - \frac{2}{3}k\delta_{ij}\right) - \frac{2}{3}C_I\Delta^2\delta_{ij}\left|S_{ij}\right|^2$, where $\vec{\tau}_{ij}$ is the sub-particle stress tensor, $\nu_t = [(C_S\Delta l)]^2|S|$ the turbulent eddy viscosity, k the SPS turbulence kinetic energy, C_s the Smagorinsky constant (0.12), C_I=0.0066, Δl the particle to particle spacing and $|S|=0.5(2S_{ij}S_{ij})$ where S_{ij} is an element of the SPS strain tensor. Dalrymple and Rogers [4] introduced SPS into weakly compressible SPH using Favre averaging, Eq.11 can be re-written as

$$\begin{aligned}\frac{d\boldsymbol{v}_a}{dt} &= -\sum_b m_b\left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2}\right)\nabla_a W_{ab} + \boldsymbol{g} \\ &+ \sum_b m_b\left(\frac{4\upsilon_0 r_{ab}\cdot\nabla_a W_{ab}}{(\rho_a+\rho_b)(r_{ab}^2+\eta^2)}\right)\boldsymbol{v}_{ab} + \\ &+ \sum_b m_b\left(\frac{\vec{\tau}_{ij}^{\,b}}{\rho_b^2} + \frac{\vec{\tau}_{ij}^{\,a}}{\rho_a^2}\right)\nabla_a W_{ab}\end{aligned} \tag{12}$$

where the superscripts refer to particles a and b.

2.3. Continuity Equation

Throughout the duration of a weakly-compressible SPH simulation (as presented herein) the mass of each particle remains constant and only their associated density fluctuates. These density changes are computed by solving the conservation of mass, or continuity equation, in SPH form:

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab}\cdot\nabla_a W_{ab} \tag{13}$$

Within DualSPHysics it is also possible to apply a *delta-SPH* formulation, that introduces a diffusive term [33] to reduce density fluctuations

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab}\cdot\nabla_a W_{ab} + 2\delta h\sum_b m_b \overline{c_{ab}}\left(\frac{\rho_a}{\rho_b} - 1\right)\frac{1}{\mathbf{r}_{ab}^2+\eta^2}\cdot\nabla_a W_{ab} \tag{14}$$

where $\overline{c_{ab}} = 0.5(c_a + c_b)$ and $\eta^2 = 0.01h^2$ and δ is the delta-SPH coefficient. This technique is designed to filter relatively large wave numbers from the density field while solving for the conservation of mass of each particle, therefore reducing noise throughout the system of particles. The term can be expanded into a first and second order contributions, where the second order corresponds to its diffusive nature and the first order is approximately zero if the kernel is complete [34]. However, at open boundaries, where a non-complete interpolation kernel is inevitably present, the first order term originates a net contribution. For this reason, it is advised that the *delta-SPH* scheme is disabled for cases that rely on hydrostatic equilibrium. If the case represents a very dynamic situation the term contributes with a force that may be several orders of

magnitude smaller than the pressure and viscous terms, not contributing to a significant degradation of the solution. A *delta-SPH (δ)* coefficient of 0.1 is recommended for most applications.

2.4. Equation of State

Following the work of Monaghan [35], the fluid in the SPH formalism defined in DualSPHysics is treated as weakly compressible and an equation of state is used to determine fluid pressure based on particle density. The compressibility is adjusted so that the speed of sound can be artificially lowered; this means that the size of time step taken at any one moment (which is determined according to a Courant condition, based on the currently calculated speed of sound for all particles) can be maintained at a reasonable value. Such adjustment however, restricts the sound speed to be at least ten times faster than the maximum fluid velocity, keeping density variations to within less than 1%, and therefore not introducing major deviations from an incompressible approach. Following [36] and [37], the relationship between pressure and density follows the expression

$$P = B\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{15}$$

where $\gamma = 7$, $B = c_0^2\rho_0/\gamma$ where $\rho_0 = 1000\,\mathrm{kg\,m^{-3}}$ is the reference density and $c_o = c(\rho_o) = \sqrt{(\partial P/\partial\rho)}\big|_{\rho_o}$ which is the speed of sound at the reference density.

2.5. Particle Motion

Particles are moved according to a method proposed by Monaghan and referred to as XSPH [38]. This aims to move particles with a velocity close to the average of the velocity of all particles in their neighbourhood in order to assure a more ordered flow and to prevent penetration between continua, particles are therefore moved using

$$\frac{d\boldsymbol{r}_a}{dt} = \boldsymbol{v}_a + \varepsilon\sum_b \frac{m_b}{\overline{\rho_{ab}}}\boldsymbol{v}_{ba}W_{ab} \tag{16}$$

where ε is a problem specific parameter ranging from 0 to 1 and $\overline{\rho_{ab}} = 0.5(\rho_a + \rho_b)$.

2.6. Shepard Filter

The Shepard filter is a correction to the density field that can be applied every *M* time steps according to the following procedure

$$\rho_a^{new} = \sum_b \rho_b\widetilde{W}_{ab}\frac{m_b}{\rho_b} = \sum_b m_b\widetilde{W}_{ab} \tag{17}$$

where the kernel has been corrected using a zeroth-order correction

$$\widetilde{W}_{ab} = \frac{W_{ab}}{\sum_b W_{ab}\dfrac{m_b}{\rho_b}} \tag{18}$$

In cases where the *delta-SPH* method is in use, it may not be sensible to apply the Shepard density filter as well, however it is possible for both methods to be used simultaneously within DualSPHysics. The frequency *M* that the filter is applied is a free parameter that can be set to between *1* and an unbounded upper limit; however it is recommended that the value is set to between *30* and *40* time steps.

2.7 Time stepping

DualSPHysics includes a choice of numerical integration schemes, if the momentum (v_a), density (ρ_a) and position ($\boldsymbol{r}_a$) equations are first written in the form

$$\frac{d\boldsymbol{v}_a}{dt} = \boldsymbol{F}_a \tag{19a}$$

$$\frac{d\rho_a}{dt} = D_a \tag{19b}$$

$$\frac{d\boldsymbol{r}_a}{dt} = \boldsymbol{v}_a \tag{19c}$$

where $\boldsymbol{v_a}$ may also include an XSPH correction when these equations are integrated in time using a computationally simple Verlet based scheme or a more numerically stable but computationally intensive two-stage Symplectic method.

2.7.1. Verlet Scheme

This algorithm, which is based on the common Verlet method [39] is split into two parts and benefits from providing a low computational overhead compared to some other integration techniques, primarily as it does not require multiple (i.e. predictor and corrector) calculations for each step. The predictor step calculates the variables according to

$$\boldsymbol{v}_a^{n+1} = \boldsymbol{v}_a^{n-1} + 2\Delta t\boldsymbol{F}_a^{\,n};\ \boldsymbol{r}_a^{n+1} = \boldsymbol{r}_a^{n} + \Delta t\boldsymbol{V}_a^{\,n} + 0.5\Delta t^2\boldsymbol{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^{n-1} + 2\Delta t D_a^{\,n} \tag{20}$$

where $\boldsymbol{F_a}^n$ and D_a^n are calculated using Eq. 7 (or Eq. 12) and Eq. 13 (or Eq. 14) respectively.

However, once every N_s time steps (where $N_s \approx 50$ is suggested), variables are calculated according to

$$\boldsymbol{v}_a^{n+1} = \boldsymbol{v}_a^{n} + \Delta t\boldsymbol{F}_a^{\,n};\ \boldsymbol{r}_a^{n+1} = \boldsymbol{r}_a^{n} + \Delta t\boldsymbol{V}_a^{\,n} + 0.5\Delta t^2\boldsymbol{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^{n} + \Delta t D_a^{\,n} \tag{21}$$

This second part is designed to stop divergence of integrated values through time as the equations are no longer coupled. In cases where the Verlet scheme is used but it is found that numerical stability is an issue, it may be sensible to increase the frequency at which the second part of this scheme is applied, however if it should be necessary to increase this frequency beyond N_s=10 then this could indicate that the scheme is not able to capture the dynamics of the case in hand suitably and the Symplectic scheme should be used instead.

2.7.2. Symplectic Scheme

Symplectic integration algorithms are time reversible in the absence of friction or viscous effects [40]. They can also preserve geometric features, such as the energy time-reversal symmetry present in the equations of motion, leading to improved resolution of long term solution behaviour. The scheme used here is an explicit second-order Symplectic scheme with an accuracy in time of $O(\Delta t^2)$ and involves a predictor and corrector stage.

During the predictor stage the values of acceleration and density are estimated at the middle of the time step according to

$$\boldsymbol{r}_a^{n+\frac{1}{2}} = \boldsymbol{r}_a^n + \frac{\Delta t}{2}\boldsymbol{v}_a^n \quad ; \quad \rho_a^{n+\frac{1}{2}} = \rho_a^n + \frac{\Delta t}{2} D_a^n \tag{22}$$

where the superscript n denotes the time step and $t = n\Delta t$.

During the corrector stage $\mathrm{d}\boldsymbol{v}_a^{n+\frac{1}{2}}/dt$ is used to calculate the corrected velocity, and therefore position, of the particles at the end of the time step according to

$$\begin{aligned} \boldsymbol{v}_a^{n+1} &= \boldsymbol{v}_a^{n+\frac{1}{2}} + \frac{\Delta t}{2}\boldsymbol{F}_a^{n+\frac{1}{2}}, \\ \boldsymbol{r}_a^{n+1} &= \boldsymbol{r}_a^{n+\frac{1}{2}} + \frac{\Delta t}{2}\boldsymbol{v}_a^{n+1}. \end{aligned} \tag{23}$$

and finally the corrected value of density $d\rho_a^{n+1}/dt = D_a^{n+1}$ is calculated using the updated values of $\boldsymbol{v}_a^{n+1}$ and $\boldsymbol{r}_a^{n+1}$ [27].

2.7.3. Variable Time Step

With explicit time integration schemes the timestep is dependent on the Courant-Friedrich-Levy (CFL) condition, the forcing terms and the viscous diffusion term. A variable time step Δt is calculated according to [41] using

$$\begin{aligned} \Delta t &= 0.3 \cdot \min\left(\Delta t_f, \Delta t_{cv}\right) \\ \Delta t_f &= \min_a\left(\sqrt{h/|\mathbf{f}_a|}\right) \\ \Delta t_{cv} &= \min_a \frac{h}{c_s + \max_b \left|\dfrac{h\boldsymbol{v}_{ab}\cdot\boldsymbol{r}_{ab}}{(r_{ab}^2+\eta^2)}\right|} \end{aligned} \tag{24}$$

where Δt_f is based on the force per unit mass ($|\mathbf{f}_a|$), and Δt_{cv} combines the Courant and the viscous time step controls.

2.8. Boundary conditions

In DualSPHysics, the boundary is described by a set of particles that are considered as a separate set to the fluid particles. The software currently provides functionality for solid impermeable and periodic open boundaries. Methods to allow boundary particles to be moved according to fixed forcing functions are also present.

2.8.1. Dynamic Boundary Condition

The Dynamic Boundary Condition (DBC) is the default method provided by DualSPHysics [42]. This method sees boundary particles that satisfy the same equations as fluid particles, however they do not move according to the forces exerted on them. Instead, they remain either fixed in position or move according to an imposed/assigned motion function (i.e. moving objects such as gates or wave-makers).

When a fluid particle approaches a boundary and the distance between its particles and the fluid particle becomes smaller than twice the smoothing length (h), the density of the affected boundary particles increases, resulting in a pressure increase. In turn this results in a repulsive force being exerted on the fluid particle due to the pressure term in the momentum equation.

Stability of this method relies on the length of time step taken being suitably short in order to handle the highest present velocity of any fluid particles currently interacting with boundary particles and is therefore an important point when considering how the variable time step is calculated.

2.8.2. Periodic Open Boundary Condition

DualSPHysics provides support for open boundaries in the form of a periodic boundary condition. This is achieved by allowing particles that are near an open lateral boundary to interact with the fluid particles near the complimentary open lateral boundary on the other side of the domain.

In effect, the compact support kernel of a particle is clipped by the nearest open boundary that it is nearest to and the remainder of its clipped support applied at the complimentary open boundary.

2.8.3. Pre-imposed Boundary Motion

Within DualSPHysics it is possible to define a pre-imposed movement for a set of boundary particles. Various predefined movement functions are available as well as the ability to assign a time-dependant input file containing kinematic detail.

These boundary particles behave as DBC described in Section 2.8.1, however rather than being fixed, they move independently of the forces currently acting upon them.

This provides the ability to define complex simulation scenarios (i.e. a wave-making paddle) as the boundaries influence the fluid particles appropriately as they move.

2.8.4. Fluid-driven Objects

It is also possible to derive the movement of an object by considering its interaction with fluid particles and using these forces to drive its motion. This can be achieved by summing the force contributions for an entire body. By assuming that the body is rigid, the net force on each boundary particle is computed according to the sum of the contributions of all surrounding fluid particles according to the designated kernel function and smoothing length. Each boundary particle k therefore experiences a force per unit mass given by

$$\boldsymbol{f}_k = \sum_{a \in WPs} \boldsymbol{f}_{ka} \tag{25}$$

where $\mathbf{f}_{ka}$ is the force per unit mass exerted by the fluid particle a on the boundary particle k, which is given by

$$m_k \boldsymbol{f}_{ka} = -m_a \boldsymbol{f}_{ak} \tag{26}$$

For the motion of the moving body, the basic equations of rigid body dynamics can then be used

$$M \frac{d\boldsymbol{V}}{dt} = \sum_{k \in BPs} m_k \boldsymbol{f}_k \tag{27a}$$

$$I \frac{d\boldsymbol{\Omega}}{dt} = \sum_{k \in BPs} m_k (\boldsymbol{r}_k - \boldsymbol{R}_0) \times \boldsymbol{f}_k \tag{27b}$$

where M is the mass of the object, I the moment of inertia, $\boldsymbol{V}$ the velocity, $\boldsymbol{\Omega}$ the rotational velocity and $\boldsymbol{R}_0$ the centre of mass. Equations 27a and 27b are integrated in time in order to predict the values of $\boldsymbol{V}$ and $\boldsymbol{\Omega}$ for the beginning of the next time step. Each boundary particle within the body then has a velocity given by

$$\boldsymbol{u}_k = \boldsymbol{V} + \boldsymbol{\Omega} \times (\boldsymbol{r}_k - \boldsymbol{R}_0) \tag{28}$$

Finally, the boundary particles within the rigid body are moved by integrating Eq. 28 in time. Thw work of [27] and [43] show that this technique conserves both linear and angular momenta.

3. CPU & GPU implementations

The DualSPHysics code is the result of an optimised implementation that uses good practice approaches for CPU and GPU SPH computation, with simulation accuracy, reliability and numerical robustness given precedence over computational performance where necessary. SPH software frameworks (such as DualSPHysics) can be split into three main steps; (i) neighbour list, (ii) computation of forces between particles and solving momentum and continuity equations and (iii) update of the physical quantities

at the next time step using an integration scheme. Running a simulation therefore means executing these steps in an iterative manner:

1st STEP. Neighbour list (Cell-linked list described in [44] and developed by [45]):

- Domain is divided in square cells of side $2h$ (or the size of the kernel domain).
- Only a list of particles, ordered according to the cell to which they belong, is generated.
- All the arrays with the physical variables of the particles are reordered according the list of particles.
- Note that an actual neighbour list is not created, but instead a list of particles reordered according to the cell they belong to, which facilitates the identification of real neighbours during the next step.

2nd STEP. Particle interaction:

- Particles of the same cell and adjacent cells are candidates to be neighbours.
- Each particle interacts with all its neighbouring particles (at a distance $< 2h$) solving momentum and continuity equation.

3rd STEP. System update:

- New time step is computed.
- Physical quantities are updated in the next step starting from the values of physical variables at the present time step, the interaction forces and the new time step value.
- Particle information (velocity and density) are saved on local storage (the hard drive) at defined times.

The GPU implementation is initially focused on the force computation as this is the most consuming part in terms of runtime [44]. The most efficient technique has been found to be to minimise communication between the CPU and GPU, as the PCI-Express bus used by current GPU hardware is the slowest point in the computing infrastructure. If the neighbour list and system update are also implemented on the GPU a CPU-GPU memory transfer is only needed at the beginning of the simulation, while relevant data will be transferred to the CPU only when saving output data is required (usually infrequently). Hence, the three steps (Neighbour list, Particle interaction and System update) were implemented entirely on the GPU to minimise CPU-GPU data transfer. Crespo et al. [12] showed results of this implementation in the DualSPHysics code where the executions were performed entirely on the GPU to simulate a benchmark case of a dam break impacting on obstacle where the numerical results are in close agreement with the experimental results.

The GPU and CPU version of the code are optimized differently to exploit the characteristics of the two architectures. The main difference is the manner in which parallel execution is performed. For example, for all loops regarding particle interactions the GPU model utilises one thread of execution to compute the resulting force of one particle as it performs all interactions with its neighbours. In the CPU code however symmetry of particle interaction is exploited in order to reduce runtime. This optimisation is not applied in the GPU implementation as there is no efficient solution

to avoid typical parallel problems such as memory race conditions arising from using slightly faster but naïve approaches of assuming one particle per thread.

DualSPHysics is unique in that the same application can be run using either a CPU or GPU implementation; this facilitates the use of the code not only on workstations with a CUDA enabled NVIDIA GPU but also on machines with suitable CPU processing hardware. The main code has a common core for both the CPU and GPU implementations, with only minor source code differences implemented for the two devices applying the specific optimizations for CPU and GPU. This commonality ensures that debugging or maintenance of the code is easier and comparisons of results and computational time are more direct. It is important to note that the CPU and GPU versions of the code may produce results that exhibit minor differences given the same initial case. This is due to the fact that parallel operations may be performed in different orders, which, with floating point arithmetic, can lead to differences in the final few decimal digits. Also the use of different hardware can lead to small differences when IEEE-754 is not fully supported. This effect is common to parallel codes and is an expected phenomenon that should be kept in mind when comparing results obtained using different computing hardware. Figure 1 shows a flow diagram to represent the differences between the CPU and GPU implementations and the different steps involved in a complete execution.

4. Program documentation

4.1 Source files

A set of C++ and CUDA files need to be compiled to generate the DualSPHysics binary. Here all the source files are listed, however each file contains more detailed comments describing the SPH formulation and the algorithms. As mentioned before, the same application can be run using either a CPU or GPU implementation; therefore some files are common for the SPH solver while others are specific to CPU or GPU executions. Table 1 shows a general overview of the different source files integrated in the project.

The following tables show the goal of each individual file; Table 2 describes the files not related to the SPH solver; Table 3 describes the files of the SPH solver common to CPU and GPU implementations; and Table 4 and Table 5 describe the files for the specific execution on CPU and GPU, respectively.

Please note that both the C++ and CUDA version of the code contain the same features and options. Most of the source code is common to CPU and GPU (files in Table 2 and Table 3).

4.2 Compilation

The code can be compiled for either CPU or GPU execution. In order to compile the code for CPU execution, only a C++ compiler (for example GNU's g++) is needed with the resultant binary allowing the code to be run on workstations without a CUDA-enabled GPU.

To run DualSPHysics on GPU, an NVIDIA CUDA-enabled GPU is needed and the latest version of the GPU driver must be installed. However, to compile the source code, the GPU programming language CUDA and NVCC compiler must be installed on the computer. The CUDA Toolkits can be downloaded directly from NVIDIA (https://developer.nvidia.com/cuda-downloads). CUDA versions 4.0, 4.1, 4.2, 5.0, and 5.5 have been tested (the same numerical results are obtained with different CUDA versions).

Makefiles can be used to compile the code:

(i) make –f *Makefile_cpu* only for CPU compilation (files of Table 5 are not included in the compilation) leading to the binary *DualSPHysicsCPU_linux64,*

(ii) make –f *Makefile* for a full compilation creating a binary for CPU-GPU and the result of the compilation is the binary *DualSPHysics_linux64*.

The user can modify the compilation options such as the path of the CUDA toolkit directory or the GPU architecture By default the GPU code is compiled for "sm_12,compute_12" and "sm_20,compute_20" and CUDA v5.0, the log file generated by the compiler is stored in the file *DualSPHysics_ ptxasinfo.* For example, any possible error in the compilation of *JSphGpu_ker.cu* can be identified in this *ptxasinfo* file. This file is also parsed by the executable on initial startup in order to perform hardware specific kernel optimisation.

4.3 Format files

Different format files for the input and the output data are involved in the DualSPHysics execution: .xml, .bi2 and .vtk.

The XML (E*X*tensible *M*arkup *L*anguage) is a textual data format that can easily be read or written using any platform and operating system. It is based on a set of labels (tags) that organise the information and can be loaded or written easily using any standard text or dedicated XML editor. This format is used for input files for the code.

Data stored in text format (ASCII) consumes at least six times more memory than the same data stored in binary format. Reading and writing data in ASCII is computationally more expensive than using binary (this can be as high as two orders of magnitude). As DualSPHysics allows simulations to be performed with a large number of particles, a binary file format is necessary to avoid these problems. The use of a binary format reduces the stored size of the files and also the time dedicated to generating them. The format used in DualSPHysics is named BINX2 (.bi2), these files contain only the meaningful information of particle properties. Some variables are

removed, e.g. the pressure is not stored since it can be calculated starting from the density using the equation of state as a pre-processing step. The value for mass is constant for fluid and boundary particles and so only two values are used instead of an array. The position of fixed boundary particles is only stored in the first file since they remain unchanged throughout the simulation. Data for particles that leave the limits of the domain are stored in an independent file which leads to an additional saving. Hence, the advantages of BINX2 can be summarised as: (i) memory storage reduction, (ii) fast access, (iii) no precision lost and (iv) portability (i.e. to different architectures or different operating systems).

VTK (Visualization ToolKit) files are used for final visualization of the results and can either be generated as a pre-processing step or output directly by DualSPHysics instead of the standard BINX format (albeit at the expense of computational overhead). VTK not only supports the particle positions, but also physical quantities that are obtained numerically for the particles involved in the simulations. VTK supports many data types, such as scalar, vector, tensor, texture, and also supports different algorithms such as polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation. The VTK file format consists of a header that describes the data and includes any other useful information, the dataset structure with the geometry and topology of the dataset and its attributes. Here VTK files of POLYDATA type with legacy-binary format is used. This format is also easy for read-write operations.

4.4 Running DualSPHysics

The input files to run the DualSPHysics code include one XML file (*Case.xml*) and a binary file (*Case.bi2*). *Case.xml* contains all the parameters of the system configuration and its execution, such as key variables (i.e. smoothing length, reference density, gravity, coefficient to calculate pressure, speed of sound), the number of particles in the system, movement definition of moving boundaries and properties of moving bodies. The binary file *Case.bi2* contains the initial particle data; arrays of position, velocity and density and headers. The output files of DualSPHysics consist of binary format files (by default) with the particle information at different instants of the simulation: *Part0000.bi2, Part0001.bi2, Part0002.bi2 …, PartOut.bi2* with excluded particles and *Run.out* with a brief description of the simulation.

Different execution parameters can be changed in the XML file: time stepping algorithm specifying Symplectic or Verlet, choice of kernel function which can be Cubic or Wendland, the value for artificial viscosity or laminar+SPS viscosity treatment, activation of the Shepard density filter and how often it is applied, activation of the delta-SPH correction, the maximum time of simulation and time intervals to save the output data. To run the code, it is also necessary to specify whether the simulation is going to run in CPU or GPU mode, the format of the output files, files that summarise the execution process with the computational time of each individual process. For CPU executions, a multi-core implementation using OpenMP enables executions in parallel

using the different cores of the machine. It takes the maximum number of cores of the device by default or users can specify the number used. In addition, the parallel execution with OpenMP can use dynamic or static load balancing.

To run the program, type the command *./DualSPHysics_linux64 Case [options]*
,where *Case* is the name of the input files (*Case.xml* and Case.bi2). The configuration of the execution is mostly defined in the XML file, but it can be also defined or changed using execution parameters. Furthermore, new options and possibilities for the execution can be imposed using [options] as seen in Table 6. For example:

$dualsphysics $dirout/$name $dirout -svres –cpu
enables the simulation on the cpu, where $dirout is the directory with the file $name.bi2

$dualsphysics $dirout/$name $dirout -svres –gpu
enables the same simulation on the gpu.

$dualsphysics $dirout/$name $dirout -svres –gpu –partbegin:69
restarts the simulation from the time corresponding to files output Part0069.bi2

5. Performance analysis

The efficiency and performance of DualSPHysics are analysed in this section. The same case of study is executed in different devices (CPU and different GPUs) and runtimes and speedups are presented.

The test case consists of a dam break problem confined within a rectangular box 160 cm long, 67 cm wide and 40 cm high. The volume of water initially contained behind a thin gate at one end of the box is 40 cm long x 67 cm x 30 cm high. A tall structure, which is 12 cm x 12 cm x 45 cm in size, is placed 50 cm downstream of the gate and 24 cm from the nearest sidewall of the tank. A physical time of 1.5 seconds is calculated. Different instants of the simulation can be observed in Figure 2.

A validation of DualSPHysics using this test case has already been shown in [13] where experimental forces exerted onto the structure were in good agreement with the numerical values.

This case is executed using the Intel Xeon X5500 CPU and using different GPUs (NVIDIAs GTX 480, GTX 680, GTX Titan and Tesla K20) whose general specifications are summarised in Table 7.

The performance of different simulations of the same case is presented for 1.5 seconds of physical time. The performance is analysed for different resolutions by running calculations with different numbers of particles. Computational times of the executions

on CPU and GPU are shown in Figure 3 where it can be noticed that for a simulation of 3 million particles takes one hour using the GTX Titan GPU card while it takes almost 2 days using a CPU.

This important acceleration of the code using the new GPU technology can also be observed in Figure 4, where the speedups of different GPUs are shown by comparing their performance against the CPU device using a single core and also the full 8 cores of the Intel Xeon X5500. For the case chosen here, the use of a GPU can accelerate the SPH computations by almost two orders of magnitude, e.g. the GTX Titan card is 149 times faster than the single core CPU and 24 times faster than the CPU using all 8 cores.

Figure 5 shows the runtime distribution of the three main SPH steps; neighbour list (NL) creation, particle interaction (PI) and system update (SU) when simulating one million particles. The particle interaction takes 98.5% of the total computational time when using a CPU single-core and this percentage decreases when the code is parallelised. Hence PI takes 90% when using the 8 cores of the CPU and it is reduced to 88.3% and 85.7% when using GPU cards (GTX 480 and GTX Titan, respectively). On the other hand the percentages of NL and SU increase with the number of cores to parallelise over.

Finally, Figure 6 gives an idea of how many particles can be simulated on the different GPU devices employed when using the DualSPHysics code. It can be observed that the difference in terms of speedup between GTX 680 and Tesla K20 is negligible (see Figure 4) and the main difference of using these two GPU cards lies in the memory space that allows simulating more than 28 million particles in one Tesla K20 while less than the half can be simulated with a GTX 680.

6. Applications

DualSPHysics has proven to be efficient and reliable; results of efficiency have been shown in the previous section and DualSPHysics has also been validated experimentally. These validations were performed not only computing forces exerted onto a tall structure ([13]), but also studying wave propagation, where numerical values of surface elevation are in good agreement with experiments ([13]), comparing pressures such as in the first validation of DualSPHysics shown in [12], and computing the run-up in a sea breakwater ([14]).

Here, an application that includes some of the functionalities of DualSPHysics code is presented to demonstrate the capabilities of the code. Hence, this working example includes:

- bottom and wall of the numerical tank using fixed boundaries (subsection 2.8.1),
- periodic open boundaries at the lateral limits (subsection 2.8.2),
- piston wavemaker using predefined motion (subsection 2.8.3),

- a boat that behaves as a floating body (subsection 2.8.4),
- volume of water that fills the numerical wave basin.

The initial setup is depicted in Figure 7, where the dimensions of the numerical tank and the boat are shown. An initial particle distance of 0.03 m leads to 6,714,451 particles (6,184,843 fluid particles). The mass of the floating boat is set to 2,102.88 kg and the piston moves following a sinusoidal movement with frequency of 0.3 Hz and amplitude of 0.5 m.

Different instants of the simulation using DualSPHysics are shown in the frames of Figure 8. The simulation is performed using the GTX Titan where 6M particles and 10 seconds of real time take 41 hours to compute.

7. Conclusions and future work

The DualSPHysics code has been developed to study complex free-surface flows that require high computational resources. DualSPHysics is an open-source code, redistributed under the terms of the GNU General Public License as published by the Free Software Foundation. Along with the source code, documentation that makes compilation and execution of the source files easy is also distributed. In addition, working examples are also provided.

The code has been shown to be robust, efficient and reliable. The parallel power computing of Graphics Computing Units (GPUs) is used to accelerate DualSPHysics up to two orders of magnitude when compared to the performance achieved using a serial version.

The aim of DualSPHysics is two-fold. Firstly the code is a user-friendly platform designed to encourage other researchers to use the SPH technique to investigate a large number of novel CFD problems. Secondly, the method can be used by industry to simulate real problems that are beyond the scope of classical models.

New features are constantly being integrated into the DualSPHysis code; current examples include an MPI implementation for Multi-GPU execution [46], double precision [47], variable particle resolution [48], multiphase cases (gas-soil-water) [49,50], new boundary conditions [51]. The code is also being coupled with the Discrete Element Method [52], with a Mass Point Lattice Spring Model [53] and hybridised with the SWASH Wave Propagation Model [54].

Acknowledgement

This work was partially supported by Xunta de Galicia under project Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas (Grupos de Referencia Competitiva) co-funded by European Regional Development Fund (FEDER) and Ministerio de Economía y Competitividad under project BIA2012-38676-C03-03. The authors gratefully acknowledge the support of EPSRC EP/H003045/1 and a Research Councils UK (RCUK) fellowship.

References

[1] M. Gómez-Gesteira, B.D. Rogers, R.A. Dalrymple, A.J.C. Crespo, State-of-the-art of classical SPH for free-surface flows, Journal of Hydraulic Research 48 (2010) 6–27.

[2] M. Gómez-Gesteira, B.D. Rogers, A.J.C. Crespo, R.A. Dalrymple, M. Narayanaswamy, J.M. Domínguez, SPHysics - development of a free-surface fluid solver- Part 1: Theory and Formulations, Computers & Geosciences 48 (2012) 289-299.

[3] M. Gómez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Domínguez, A. Barreiro, SPHysics - development of a free-surface fluid solver- Part 2: Efficiency and test cases, Computers & Geosciences 48 (2012) 300-307.

[4] R.A. Dalrymple, B.D. Rogers, Numerical modeling of water waves with the SPH method, Coastal Engineering 53 (2006) 141–147.

[5] A.J.C. Crespo, M. Gómez-Gesteira, R.A. Dalrymple, Modeling Dam Break Behavior over a Wet Bed by a SPH Technique, Journal of Waterway, Port, Coastal and Ocean Engineering 134 (6) (2008) 313-320.

[6] M. Gómez-Gesteira, R. Dalrymple, Using a 3D SPH method for wave impact on a tall structure, Journal of Waterway, Port, Coastal and Ocean Engineering 130 (2) (2004) 63-69.

[7] B.D. Rogers, R.A. Dalrymple, P.K. Stansby, Simulation of caisson breakwater movement using SPH, Journal of Hydraulic Research 48 (2010) 135-141.

[8] R. Vacondio, B.D. Rogers, P.K. Stansby, P. Mignosa, A correction for balancing discontinuous bed slopes in two-dimensional smoothed particle hydrodynamics shallow water modelling, International Journal for Numerical Methods in Fluids 71 (2012) 850–872.

[9] R. Vacondio, B.D. Rogers, P.K. Stansby, P. Mignosa, Shallow water SPH for flooding with dynamic particle coalescing and splitting, Advances in Water Resources 58 (2013) 10-23.

[10] A. Herault, G. Bilotta, R.A. Dalrymple, SPH on GPU with CUDA, Journal of Hydraulic Research 48 (2010) 74–79.

[11] J.M. Dominguez, A.J.C. Crespo, M. Gómez-Gesteira, Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method, Computer Physics Communications 184(3) (2013) 617-627.

[12] A.J.C. Crespo, J.M. Dominguez, A. Barreiro, M. Gómez-Gesteira, B.D. Rogers, GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods, PLoS ONE 6 (6) (2011) e20685.

[13] A. Barreiro, A.J.C. Crespo, J.M. Domínguez and M. Gómez-Gesteira, Smoothed Particle Hydrodynamics for coastal engineering problems, Computers and Structures 120(15) (2013) 96-106.

[14] C. Altomare, A.J.C. Crespo, B.D. Rogers, J.M. Domínguez, X. Gironella, M. Gómez-Gesteira, Numerical modelling of armour block sea breakwater with Smoothed Particle Hydrodynamics, Computers and Structures 130 (2014) 34-45.

[15] R. Vacondio, P. Mignosa, S. Pagani, 3D SPH numerical simulation of the wave generated by the Vajont rockslide, Advances in Water Resources 59 (2013) 146-156.

[16] J.M. Cherfils, G. Pinon, Rivoalen, JOSEPHINE: A parallel SPH code for free-surface flows, Computer Physics Communications 183(7) (2012) 1468-1480.

[17] http://www.gpusph.org/ (date access 23-07-2014)

[18] http://canal.etsin.upm.es/aquagpusph/ (date access 23-07-2014)

[19] http://isph.sourceforge.net/ (date access 23-07-2014)

[20] http://www.mpa-garching.mpg.de/gadget/ (date access 23-07-2014)

[21] https://code.google.com/p/pysph/ (date access 23-07-2014)

[22] http://www.sph-flow.com/ (date access 23-07-2014)

[23] http://www.simpartix.com/ (date access 23-07-2014)

[24] http://www.itm.uni-stuttgart.de/research/pasimodo/pasimodo_en.php (date access 23-07-2014)

[25] J.J. Monaghan, Smoothed particle hydrodynamics, Annual Review of Astronomy and Astrophysics 30 (1992) 543-574.

[26] G.R. Liu, Mesh Free methods: Moving beyond the finite element method, CRC Press, 2003.

[27] J.J. Monaghan, Smoothed Particle Hydrodynamics. Rep. Prog. Phys. 68 (2005) 1703-1759.

[28] D. Violeau, Fluid Mechanics and the SPH Method: Theory and Applications, Oxford University Press (2012) ISBN: 0199655529.

[29] J.J. Monaghan, SPH without Tensile Instability, Journal Computational Physics, 159 (2000) 290-311.

[30] H. Wendland, Piecewiese polynomial, positive definite and compactly supported radial functions of minimal degree, Advances in Computational Mathematics 4 (1995) 389-396.

[31] E.Y.M. Lo, S. Shao, Simulation of near-shore solitary wave mechanics by an incompressible SPH method, Applied Ocean Research 24 (2002) 275-286.

[32] H. Gotoh, T. Shibihara, M. Hayashii, Subparticle-scale model for the MPS method-lagrangian flow model for hydraulic engineering, Computational Fluid Dynamics Journal 9 (2001) 339–347.

[33] D. Molteni, A. Colagrossi, A simple procedure to improve the pressure evaluation in hydrodynamic context using the SPH, Comput. Phys. Comm. 180(6) (2009) 861–872.

[34] M. Antuono, A. Colagrossi, S. Marrone, Numerical diffusive terms in weakly-compressible SPH schemes, Computer Physics Communications 183 (2012)

[35] J.J. Monaghan, Simulating free surface flows with SPH, Journal of Computational Physics 110 (1994) 399- 406.

[36] J.J. Monaghan, R.A.F. Cas, A.M. Kos, M. Hallworth, Gravity currents descending a ramp in a stratified tank, Journal of Fluid Mechanics 379 (1999) 39–70.

[37] G.K. Batchelor, Introduction to fluid dynamics, Cambridge University Press. U.K, 1974.

[38] J.J. Monaghan, On the problem of penetration in particle methods, Journal of Computational Physics 82 (1989) 1-15.

[39] L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Physical Review 159 (1967) 98-103.

[40] B.J. Leimkuhler, S. Reich, R.D. Skeel, Integration Methods for Molecular dynamic IMA Volume in Mathematics and its application. Springer, 1996.

[41] J.J. Monaghan, A. Kos, Solitary waves on a Cretan beach, Journal of Waterway, Port, Coastal and Ocean Engineering 125 (3) (1999) 145-154.

[42] A.J., Crespo, M. Gómez-Gesteira, R.A. Dalrymple, Boundary Conditions Generated by Dynamic Particles in SPH Methods, CMC: Computers, Materials, & Continua 5 (3) (2007) 173-184.

[43] J.J. Monaghan, A. Kos, N. Issa, Fluid motion generated by impact, Journal of Waterway, Port, Coastal and Ocean Engineering 129 (2003) 250-259.

[44] J.M. Dominguez, A.J.C. Crespo, M. Gómez-Gesteira, J.C. Marongiu, Neighbour lists in Smoothed Particle Hydrodynamics, International Journal for Numerical Methods in Fluids 67 (2011) 2026-2042.

[45] T.J. Purcell, I. Buck, W.R. Mark, P. Hanrahan, Ray Tracing on Programmable Graphics Hardware, ACM Transactions on Graphics 21(3) (2002) 703-712.

[46] J.M. Domínguez, A.J.C. Crespo, D. Valdez-Balderas, B.D. Rogers, M. Gómez-Gesteira, New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters, Computer Physics Communications 184 (2013) 1848-1860.

[47] J.M. Domínguez, A.J.C. Crespo, A. Barreiro, M. Gómez-Gesteira, B.D. Rogers, Efficient Implementation of Double Precision in GPU Computing to Simulate Realistic Cases with High Resolution, in: Proceedings of the 9th SPHERIC, 2014.

[48] R. Vacondio, B.D. Rogers, P.K. Stansby, P. Mignosa, J. Feldman, Variable resolution for SPH: a dynamic particle coalescing and splitting scheme, Computer Methods in Applied Mechanics and Engineering 256 (2013) 132-148.

[49] G. Fourtakas, B.D. Rogers, D. Laurence, Modelling sediment suspension in industrial tanks using SPH, La Houille Blanche 2 (2013) 39-45.

[50] A. Mokos, B.D. Rogers, P.K. Stansby, J.M. Domínguez, A multi-phase particle shifting algorithm for SPH simulations for violent hydrodynamics on a GPU, in: Proceedings of the 9th SPHERIC (2014).

[51] G. Fourtakas, J.M. Domínguez, R. Vacondio, A. Nasar, B.D. Rogers, Local Uniform STencil (LUST) Boundary Conditions for 3-D Irregular Boundaries in DualSPHysics, in: Proceedings of the 9th SPHERIC (2014).

[52] R. Canelas, R.M.L. Ferreira, J.M. Domínguez, A.J.C. Crespo, Modelling of Wave Impacts on Harbour Structures and Objects with SPH and DEM, in: Proceedings of the 9th SPHERIC, 2014.

[53] S.M. Longshaw, B.D. Rogers, P.K. Stansby, Whale to Turbine Impact Using the GPU Based SPH-LSM Method, in: Proceedings of the 9th SPHERIC (2014).

[54] C. Altomare, T. Suzuki, J.M. Domínguez, A.J.C. Crespo, M Gómez-Gesteira, Coupling Between SWASH and SPH for Real Coastal Problems, in: Proceedings of the 9th SPHERIC (2014).

Tables

Table 1. List of source files of DualSPHysics code.

Table 2. List of source files of DualSPHysics code not related to the SPH solver.
Table 3. List of source files of DualSPHysics code for the SPH execution.
Table 4. List of source files of DualSPHysics code for the SPH execution on CPU.
Table 5. List of source files of DualSPHysics code for the SPH execution on GPU.
Table 6. List of execution parameters of DualSPHysics.
Table 7. Specifications of different execution devices.

Figure captions

Figure 1. Flow diagram of the CPU (left) and total GPU implementation (right).
Figure 2. Instants of the simulation of a dam-break flow used to study the performance of DualSPHysics code. Colour represents velocity of the particles.
Figure 3. Runtime for CPU and different GPU cards.
Figure 4. Speedups of GPU against CPU simulating 1 million particles.
Figure 5. Computational runtime distribution on CPU and GPU simulating 1 million particles. Neighbour List corresponds to black bars, Particle Interaction to grey bars and System Update to the light bar.
Figure 6. Maximum number of particles simulated with different GPU cards using DualSPHysics code.
Figure 7. Initial setup of the application case.
Figure 8. Instants of the simulation of the application case to show the capabilities of DualSPHysics code (http://youtu.be/pnLTWUk6wPc?list=UU7I8ftAldTXKAWD6_VxE5Iw).

Figure1
Click here to download high resolution image

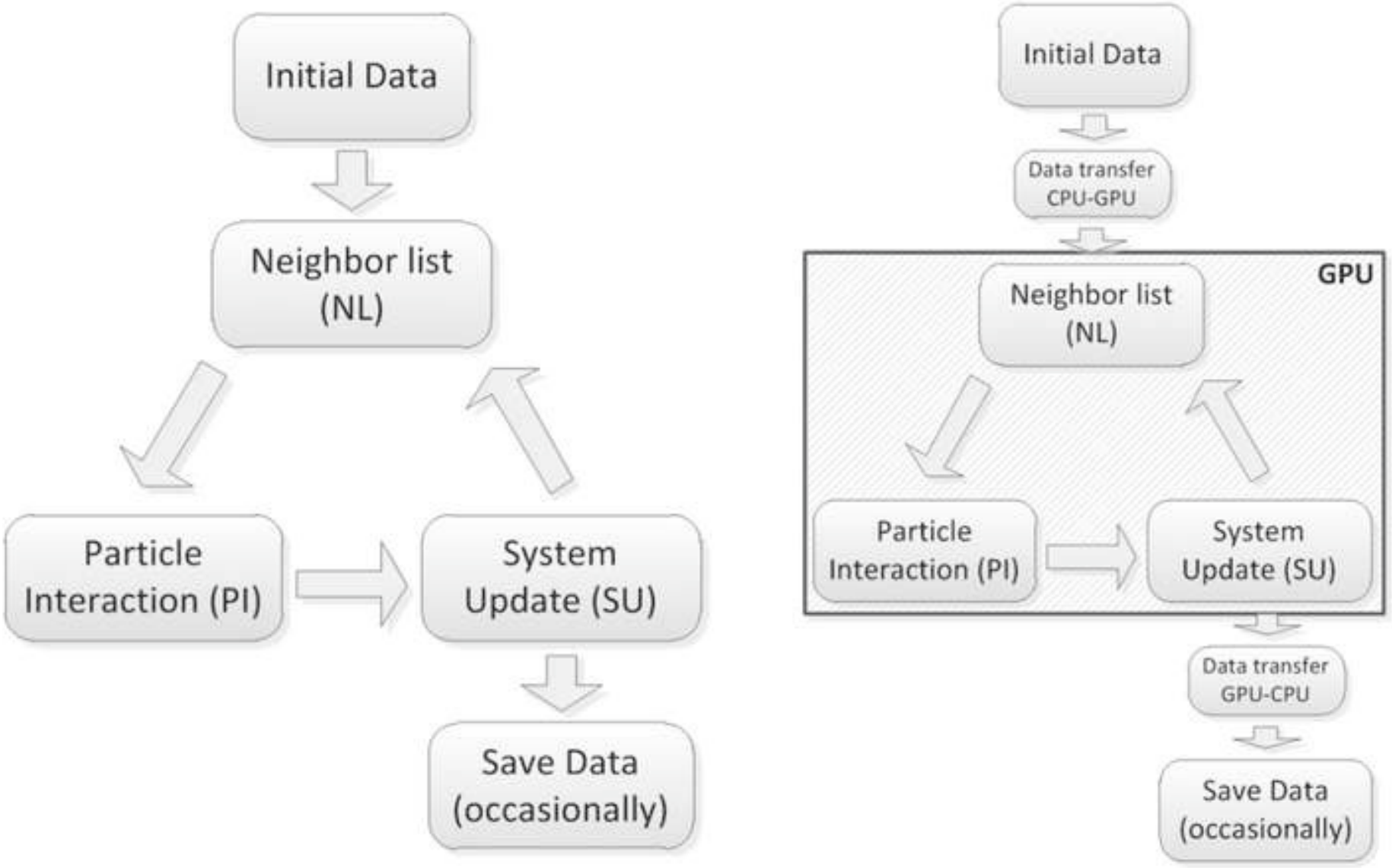

Figure2

Click here to download high resolution image

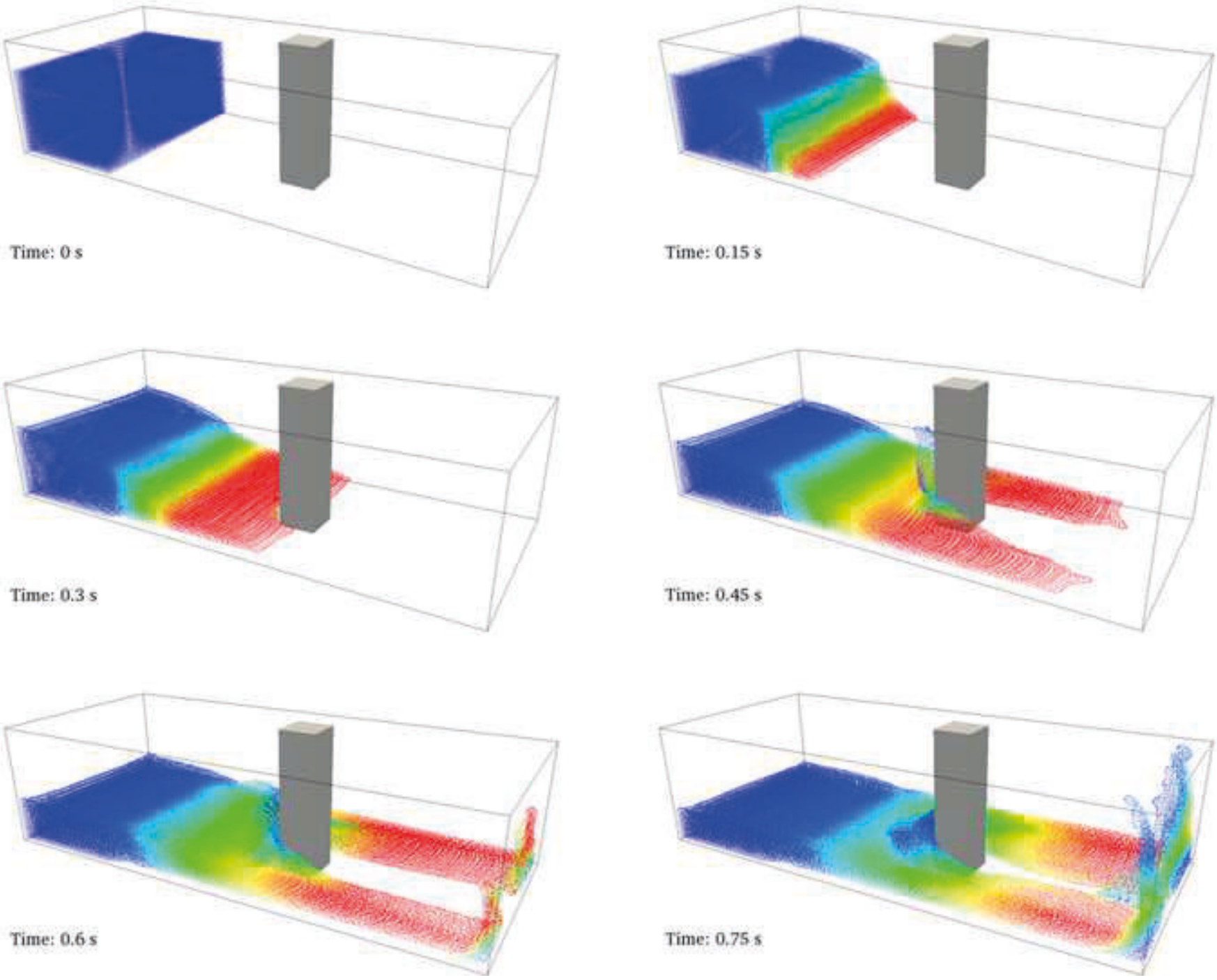

Figure3
Click here to download high resolution image

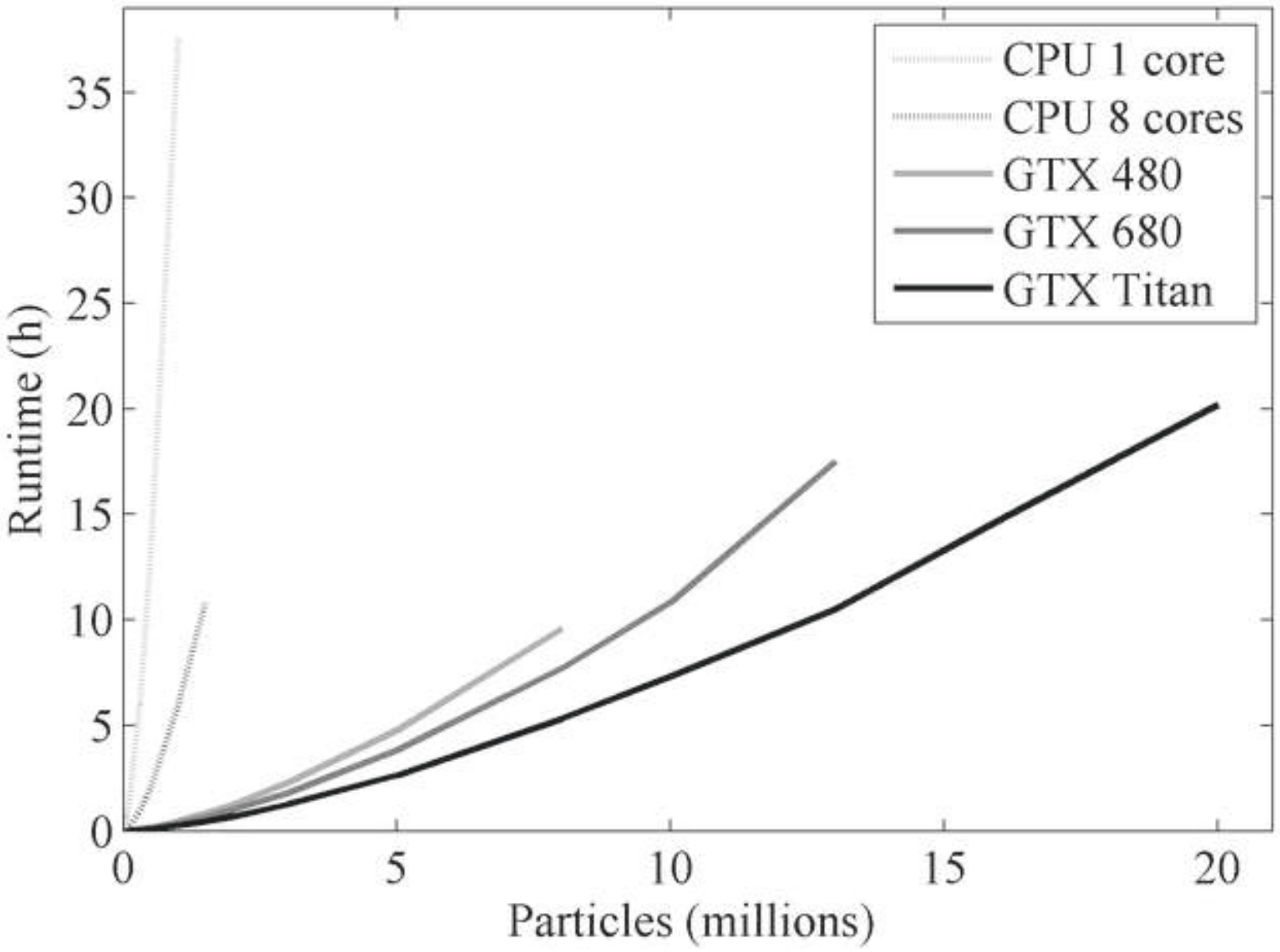

Figure4
Click here to download high resolution image

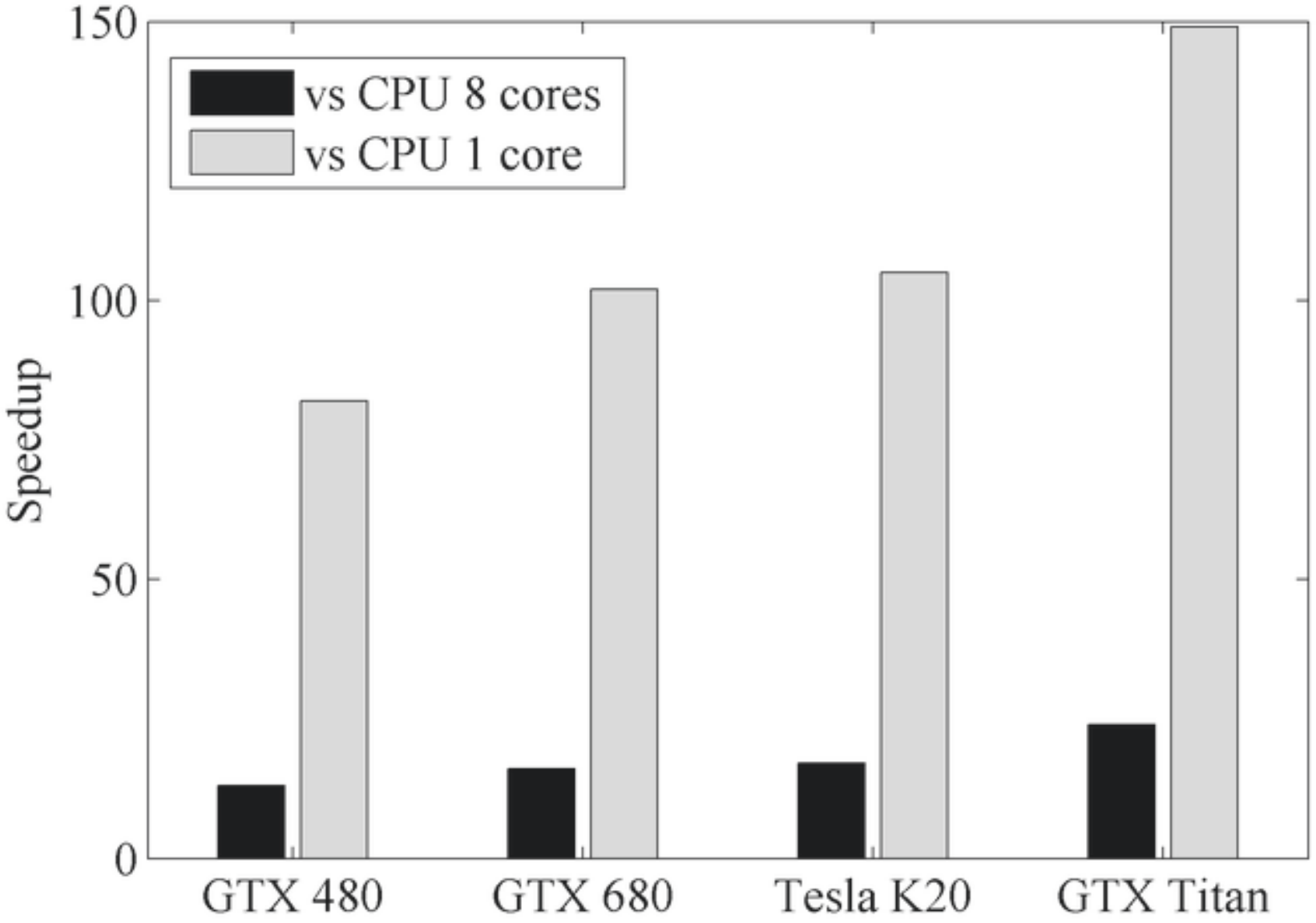

Figure5
Click here to download high resolution image

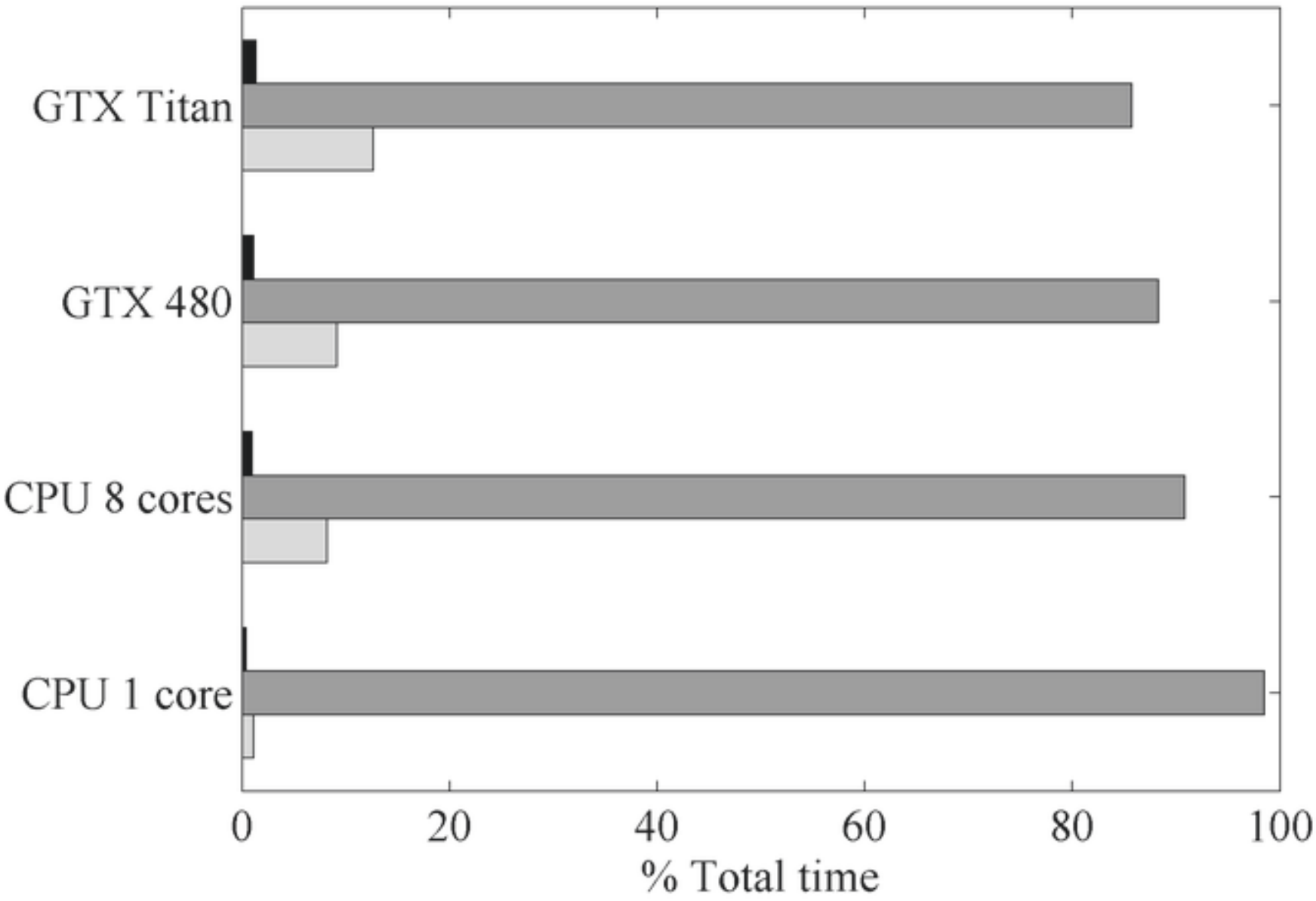

Figure6
Click here to download high resolution image

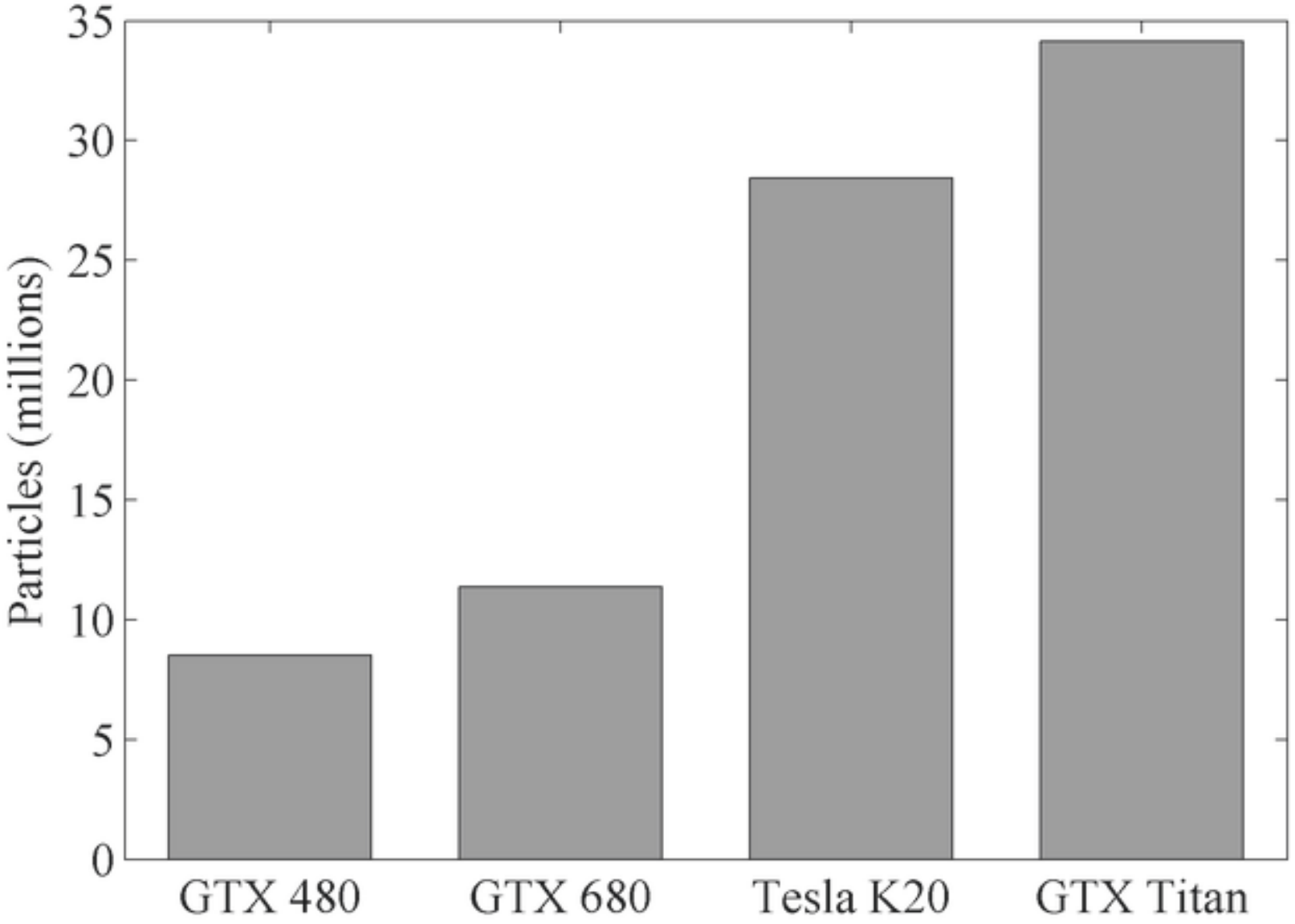

Figure7
Click here to download high resolution image

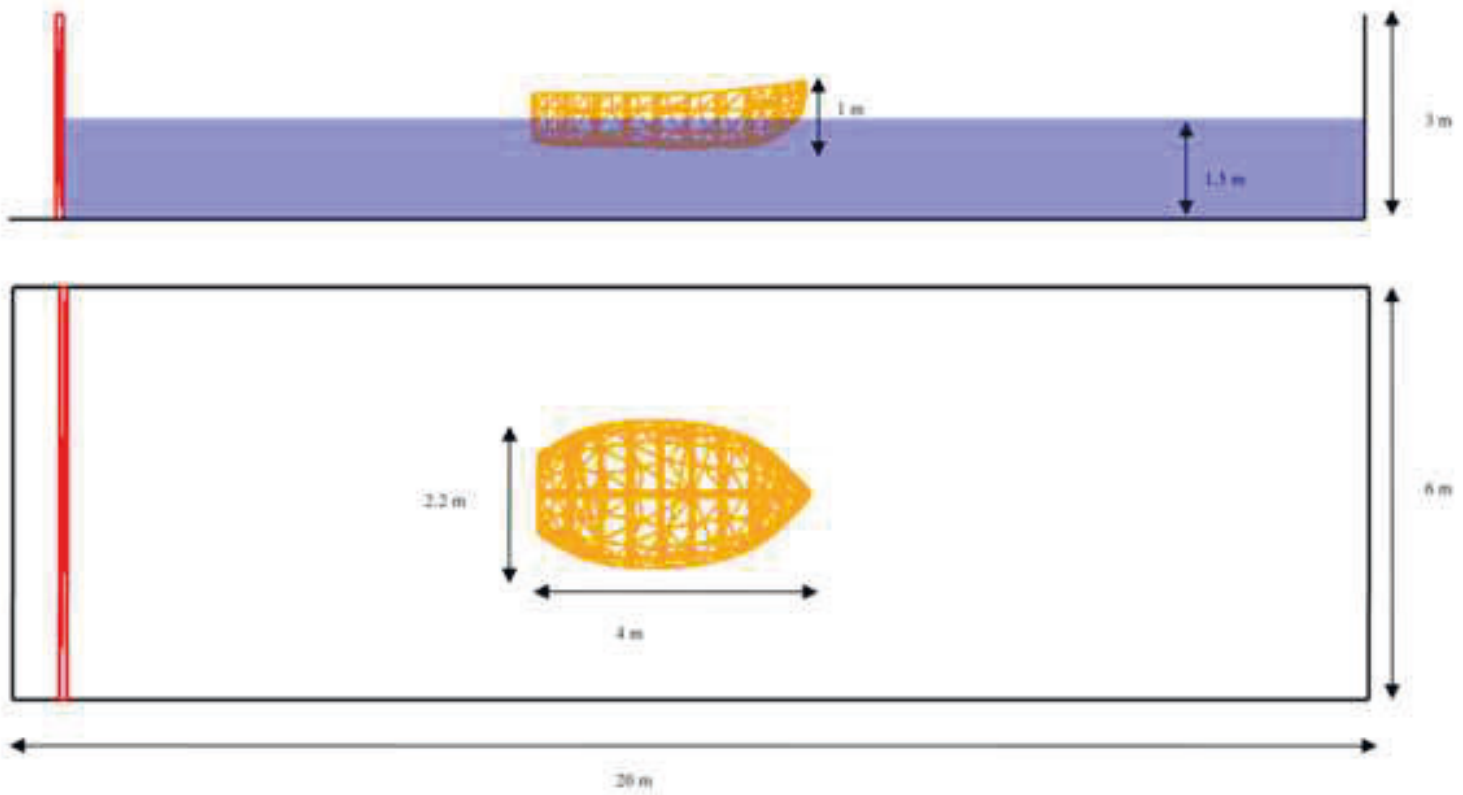

Figure8
Click here to download high resolution image

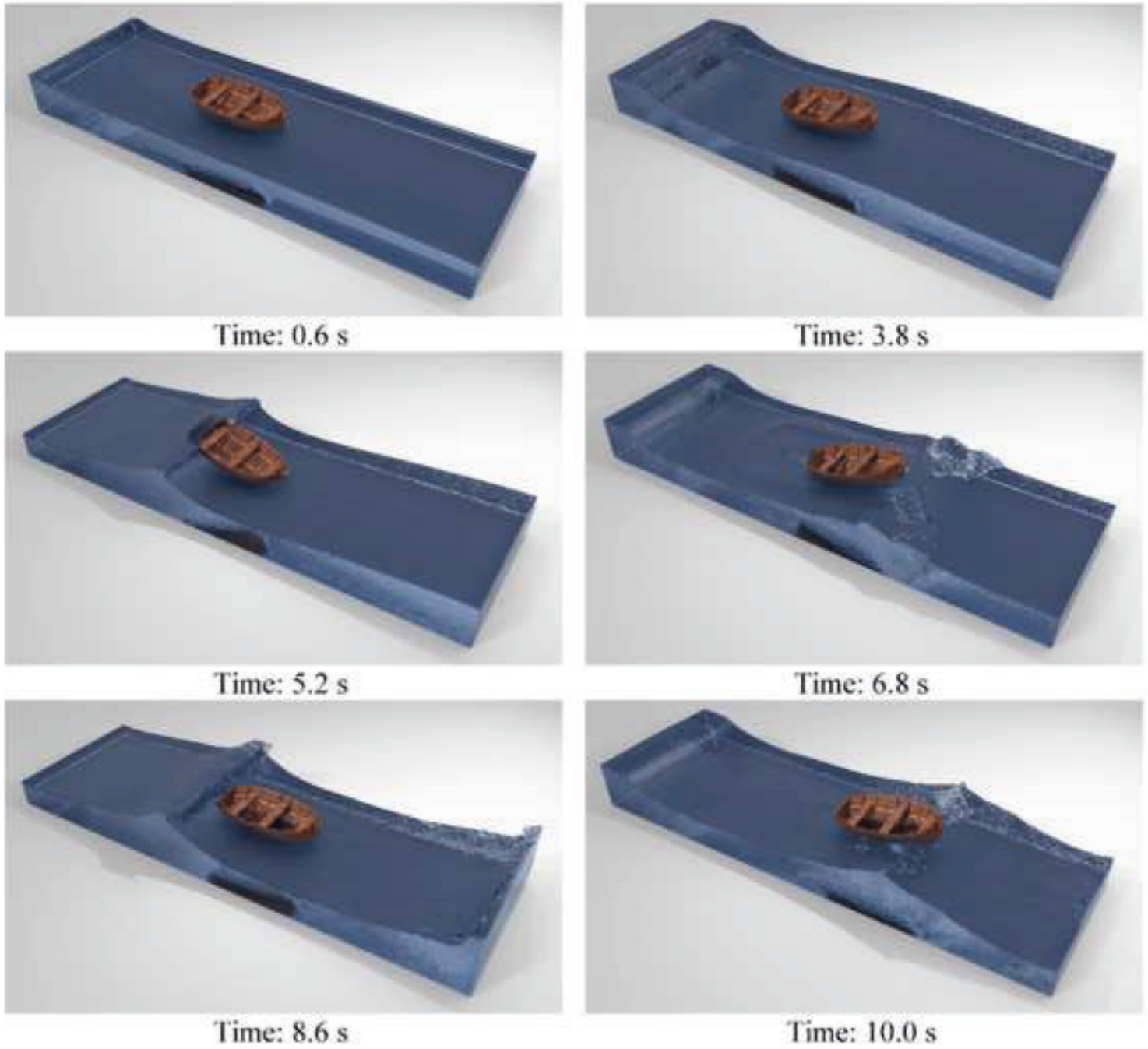

Table1

Table 1. List of source files of DualSPHysics code.

<table>
<tr><th>No SPH</th><th colspan="2">SPH on CPU & GPU</th></tr>
<tr><td rowspan="2">Functions (.h .cpp)
JException (.h .cpp)
JFloatingData (.h .cpp)
JLog2 (.h .cpp)
JObject (.h .cpp)
JObjectGpu (.h .cpp)
JPartData (.h .cpp)
JPtxasInfo (.h .cpp)
JSpaceCtes (.h .cpp)
JSpaceEParms (.h .cpp)
JSpaceParts (.h .cpp)
JSpaceProperties (.h .cpp)
JRangeFilter (.h .cpp)
JTimer.h
JTimerCuda.h
JVarsAscii (.h .cpp)
TypesDef.h

JFormatFiles2.h
JFormatFiles2.lib / libjformatfiles2.a

JSphMotion.h
JSphMotion.lib / libjsphmotion.a

JXml.h
JXml.lib / libjxml.a</td><td colspan="2">main.cpp
JCfgRun (.h .cpp)
JSph (.h .cpp)
JPartsLoad (.h .cpp)
JPartsOut (.h .cpp)
JSphDtFixed (.h .cpp)
JSphVarAcc (.h .cpp)
Types.h</td></tr>
<tr><td>SPH on CPU

JSphCpu (.h .cpp)

JSphCpuSingle (.h .cpp)

JSphTimersCpu.h

JCellDivCpu (.h .cpp)

JCellDivCpuSingle (.h .cpp)

JPeriodicCpu (.h .cpp)</td><td>SPH on GPU

JSphGpu (.h .cpp)
JSphGpu_ker (.h .cu)

JSphGpuSingle (.h .cpp)

JSphTimersGpu.h

JCellDivGpu (.h .cpp)
JCellDivGpu_ker (.h .cu)

JCellDivGpuSingle (.h .cpp)
JCellDivGpuSingle_ker (.h .cu)

JPeriodicGpu (.h .cpp)
JPeriodicGpu_ker (.h .cu)

JGpuArrays (.h .cpp)</td></tr>
</table>

Table2

Table 2. List of source files of DualSPHysics code not related to the SPH solver.

Non SPH FILES	
Functions (.h .cpp)	Declares/implements basic/general functions for the entire application.
JException (.h .cpp)	Declares/implements the class that defines exceptions with the information of the class and method.
JFloatingData (.h .cpp)	Declares/implements the class that allows reading/writing files with data of floating bodies.
JLog2 (.h .cpp)	Declares/implements the class that manages the output of information in the file Run.out and on screen.
JObject (.h .cpp)	Declares/implements the class that defines objects with methods that throws exceptions.
JObjectGpu (.h .cpp)	Declares/implements the class that defines objects with methods that throws exceptions about tasks in GPU.
JPartData (.h .cpp)	Declares/implements the class that allows reading/writing files with data of particles in formats binx2, ascii…
JPtxasInfo (.h .cpp)	Declares/implements the class that returns the number of registers of each CUDA kernel.
JSpaceCtes (.h .cpp)	Declares/implements the class that manages the info of constants from the input XML file.
JSpaceEParms (.h .cpp)	Declares/implements the class that manages the info of execution parameters from the input XML file.
JSpaceParts (.h .cpp)	Declares/implements the class that manages the info of particles from the input XML file.
JSpaceProperties (.h .cpp)	Declares/implements the class that manages the properties assigned to the particles in the XML file
JRangeFilter (.h .cpp)	Declares/implements the class that facilitates filtering values within a list.
JTimer.h	Declares the class that defines a class to measure short time intervals.
JTimerCuda.h	Declares the class that defines a class to measure short time intervals in GPU using cudaEvent.
JVarsAscii (.h .cpp)	Declares/implements the class that reads variables from a text file in ASCII format.
TypesDef.h	Declares general types and functions for the entire application.
JFormatFiles2.h	Declares the class that provides functions to store particle data in formats VTK, CSV, ASCII.
JSphMotion.h	Declares the class that provides the displacement of moving objects during a time interval.
JXml.h	Declares the class that helps to manage the XML document using library TinyXML

Table3

Table 3. List of source files of DualSPHysics code for the SPH execution.

SPH SOLVER	
main.cpp	Main file of the project that executes the code on CPU or GPU.
JCfgRun (.h .cpp)	Declares/implements the class that defines the class responsible of collecting the execution parameters by command line.
JSph (.h .cpp)	Declares/implements the class that defines all the attributes and functions that CPU and GPU simulations share.
JPartsLoad (.h .cpp)	Declares/implements the class that manages the initial load of particle data.
JPartsOut (.h .cpp)	Declares/implements the class that stores excluded particles at each instant till writing the output file.
JSphDtFixed (.h .cpp)	Declares/implements the class that manages the use of prefixed values of DT loaded from an input file.
JSphVarAcc (.h .cpp)	Declares/implements the class that manages the application of external forces to different blocks of particles (with the same MK).
Types.h	Defines specific types for the SPH application.

Table4

Table 4. List of source files of DualSPHysics code for the SPH execution on CPU.

SPH SOLVER ONLY FOR CPU EXECUTIONS	
JSphCpu (.h .cpp)	Declares/implements the class that defines the attributes and functions used only in CPU simulations.
JSphCpuSingle (.h .cpp)	Declares/implements the class that defines the attributes and functions used only in Single-CPU.
JSphTimersCpu.h	Measures time intervals during CPU execution.
JCellDivCpu (.h .cpp)	Declares/implements the class responsible of computing the Neighbour List in CPU.
JCellDivCpuSingle (.h .cpp)	Declares/implements the class responsible of computing the Neighbour List in Single-CPU
JPeriodicCpu (.h .cpp)	Declares/implements the class that manages the interactions between periodic edges in CPU

Table5

Table 5. List of source files of DualSPHysics code for the SPH execution on GPU.

SPH SOLVER ONLY FOR GPU EXECUTIONS	
JSphGpu (.h .cpp)	Declares/implements the class that defines the attributes and functions used only in GPU simulations.
JSphGpu_ker (.h .cu)	Declares/implements functions and CUDA kernels for the particle interaction and system update.
JSphGpuSingle (.h .cpp)	Declares/implements the class that defines the attributes and functions used only in Single-GPU.
JSphTimersGpu.h	Measures time intervals during GPU execution.
JCellDivGpu (.h .cpp)	Declares/implements the class that defines the class responsible of computing the Neighbour List in GPU.
JCellDivGpu_ker (.h .cu)	Declares/implements functions and CUDA kernels to compute operations of the Neighbour List.
JCellDivGpuSingle (.h .cpp)	Declares/implements the class that defines the class responsible of computing the Neighbour List in Single-GPU.
JCellDivGpuSingle_ker (.h .cu)	Declares/implements functions and CUDA kernels to compute operations of the Neighbour List.
JPeriodicGpu (.h .cpp)	Declares/implements the class that manages the interactions between periodic edges in GPU.
JPeriodicGpu_ker (.h .cu)	Declares/implements functions and CUDA kernels to obtain particles that interact with periodic edges.
JGpuArrays (.h .cpp)	Declares/implements the class that manages arrays with memory allocated in GPU.

Table6

Table 6. List of execution parameters of DualSPHysics.

PARAMETER	DESCRIPTION
-h	Shows information about parameters.
-opt <file>	Loads configuration from a file.
-cpu	Execution on Cpu (option by default).
-gpu[:id]	Execution on Gpu and id of the device.
-stable	Ensures the same results when repeated a simulation since operations are always carried out in the same order.
-ompthreads:<int>	Only for Cpu. Indicates the number of threads by host for parallel execution, it takes the number of cores of the device by default (or using zero value).
-ompdynamic	Only for Cpu. Parallel execution with symmetry in interaction and dynamic load balancing. Not compatible with **–stable.**
-ompstatic	Only for Cpu. Parallel execution with symmetry in interaction and static load balancing.
-cellorder:<axis>	Indicates the order of the axis. (xyz/xzy/yxz/yzx/zxy/zyx).
-cellmode:<mode>	Specifies the cell division mode, by default, the fastest mode is chosen **h** fastest and the most expensive in memory **2h** lowest and the least expensive in memory
-symplectic	Symplectic algorithm as time step algorithm.
-verlet[:steps]	Verlet algorithm as time step algorithm and number of time steps to switch equations.
-cubic	Cubic spline kernel.
-wendland	Wendland kernel.
-viscoart:<float>	Artifitical viscosity [0-1].
-viscolamsps:<float>	Laminar+SPS viscosity [order of 1E-6].
-shepard:steps	Shepard filter and number of steps to be applied.
-deltasph:<float>	Constant for DeltaSPH. By default 0.1 and 0 to disable.
-sv:[formats,...]	Specifies the output formats: **none** No files with particle data are generated **binx** Bynary files (option by default) **vtk** VTK files **ascii** ASCII files (PART_xxxx of SPHysics) **csv** CSV files
-svres:<0/1>	Generates file that summarizes the execution process.
-svtimers:<0/1>	Obtains timing for each individual process.
-svdomainvtk:<0/1>	Generates VTK file with domain limits.
-name <string>	Specifies path and name of the case.
-runname <string>	Specifies name for case execution.
-dirout <dir>	Specifies the output directory.
-partbegin:begin[:first] dir	RESTART option. Specifies the beginning of the simulation starting from a given PART (begin) and located in the directory (dir), (first) indicates the number of the first PART to be generated.
-incz:<float>	Allowable increase in Z+ direction. Case domain is fixed as function of the initial particles, however the maximum Z position can be increased with this option in case particles reach higher positions.
-rhopout:min:max	Excludes fluid particles out of these density limits.
-ftpause:<float>	Time to start floating bodies movement. By default 0.
-tmax:<float>	Maximum time of simulation.
-tout:<float>	Time between output files.
-ptxasfile <file>	Indicates the file with information about the compilation kernels in CUDA to adjust the size of the blocks depending on the needed registers for each kernel (only for gpu). By default, it takes the path and the name of the executable + _ptxasinfo.

Table7

Table 7. Specifications of different execution devices.

	Number of cores	Processor clock	Memory size
Xeon X5500	1-8	2.67GHz	----
GTX 480	480	1.40 GHz	1.5 GB
GTX 680	1536	1.14 GHz	2 GB
Tesla K20	2496	0.71 GHz	5 GB
GTX Titan	2688	0.88 GHz	6 GB