Computer Physics Communications 184 (2013) 1848–1860

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

# New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters

J.M. Domínguez [a], A.J.C. Crespo [a,*], D. Valdez-Balderas [b], B.D. Rogers [b], M. Gómez-Gesteira [a]

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

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

ARTICLE INFO

*Article history:*
Received 19 December 2012
Received in revised form
7 March 2013
Accepted 9 March 2013
Available online 15 March 2013

*Keywords:*
HPC
GPU
Multi-GPU
MPI
CUDA
SPH
Meshfree methods

ABSTRACT

A massively parallel SPH scheme using heterogeneous clusters of Central Processing Units (CPUs) and Graphics Processing Units (GPUs) has been developed. The new implementation originates from the single-GPU DualSPHysics code previously demonstrated to be powerful, stable and accurate. A combination of different parallel programming languages is combined to exploit not only one device (CPU or GPU) but also the combination of different machines. Communication among devices uses an improved Message Passing Interface (MPI) implementation which addresses some of the well-known drawbacks of MPI such as including a dynamic load balancing and overlapping data communications and computation tasks. The efficiency and scalability (strong and weak scaling) obtained with the new DualSPHysics code are analysed for different numbers of particles and different number of GPUs. Last, an application with more than $10^9$ particles is presented to show the capability of the code to handle simulations that otherwise require large CPU clusters or supercomputers.



## 1. Introduction

Smoothed Particle Hydrodynamics (SPH) is a purely Lagrangian method developed to model the continuum while avoiding the limitations of mesh-based methods. SPH is rapidly approaching a state of maturity, and continued improvements now offer the accuracy, stability and adaptability required in practical applications [1]. However, one of the main drawbacks of this method is its high computational cost when real engineering problems are studied, due to the large number of nodal interpolation points, or particles, required for an appropriate description. Therefore, it is imperative to develop parallel implementations of SPH capable of combining the resources of multiple devices and machines allowing simulations of millions of particles at reasonable runtimes.

In recent years, there has been a rapid development of the Graphics Processing Units (GPUs), whose performance has increased significantly over the last decade. In fact, GPUs offer now a higher computing power than CPUs. Since the appearance of specific languages such as CUDA [2] that facilitate programming of these devices, many applications with high computational costs are being implemented using this technology and, in many cases, speedups of one to two orders of magnitude have been demonstrated. However, it is important to note that not all applications are suitable for GPU, only those which exhibit a high degree of parallelism. It is no surprise therefore that new computation centres based on GPUs are emerging driven by their computing power and comparatively low energy costs per FLOP. Indeed, the current number one of the TOP500 List of the world's top supercomputers released in November 2012 (http://www.top500.org/lists/2012/11) is Titan, a Cray XK7 system that has 560,640 processors, including 18,688 NVIDIA K20x accelerator GPU cards.

The use of GPUs is therefore emerging as a key component in High Performance Computing (HPC) as an affordable option to accelerate SPH with a low economic cost (compared to traditional CPU clusters). However, the use of a single GPU card is not sufficient for engineering applications that require several million particles that predict the desired physical processes: execution times are high and the available memory space is insufficient. Multiple spatial scales are present in most phenomena involving free-surface waves. Scales that range from hundreds of metres to centimetres are necessary to describe accurately the coastal hydrodynamics. Thus, most of the relevant phenomena in coastal engineering involve spatial scales over 4–5 orders of magnitude. For large simulations it is therefore essential to harness the performance of multiple GPUs.

* Corresponding author. Tel.: +34 988387255.
*E-mail addresses:* jmdominguez@uvigo.es (J.M. Domínguez), alexbexe@uvigo.es, alexbexe@gmail.com (A.J.C. Crespo), daniel.valdezbalderas@manchester.ac.uk (D. Valdez-Balderas), benedict.rogers@manchester.ac.uk (B.D. Rogers), mggesteira@uvigo.es (M. Gómez-Gesteira).

0010-4655/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cpc.2013.03.008

This work presents a novel SPH implementation that utilises MPI and CUDA to combine the power of different devices making possible the execution of SPH on heterogeneous clusters. Specifically, the proposed implementation enables communications and coordination among multiple CPUs, which can also host GPUs, making possible multi-GPU executions.

A scheme for multi-GPU SPH simulations has been presented recently by [3]. In that work, a spatial decomposition technique was described for dividing a physical system into fixed subdomains, and then assigning a different GPU of a multi-GPU system to compute the dynamics of particles in each of those subdomains. The Message Passing Interface (MPI) was used for communication between devices, i.e. when particles migrate from one subdomain to another, and to compute the forces exerted by particles on one subdomain onto particles of a neighbouring subdomain. The algorithm was only tested up to 32 million particles on 8 GPUs at a fraction of the computational cost of a conventional HPC cluster. In the work of [4] and more recently in [5], MPI was also used to distribute the work load of SPH on multiple devices, although these studies used only CPUs. As in the case of [3], they applied a spatial decomposition of the domain into subdomains and each one was assigned to a processor but with a dynamic load balancing algorithm, which is not included in the scheme of [3]. However, in both cases the efficiency drops quickly by increasing the number of execution devices.

Apart from the recent work of [3] little research has been published using multi-GPU schemes for SPH, but other types of particle-based simulations have already been the target of parallelisation on multi-GPU systems. The field of classical molecular dynamics (MD) has perhaps seen the most extensive use of this technology, given the widespread use of this technique in the fields of physics, material science, and biology. For example, a series of publications focusing on the multi-GPU implementation of the code LAMMPS has been written recently. Those include the work of [6] describing the implementation of a hybrid GPU–CPU code for MD systems of short-ranged interactions; [7] who presented more general capabilities added to LAMMPS to simulate a wide variety of materials; and the efforts of [8] on porting, optimising, and tuning LAMMPS for biological simulations. Other significant works on MD on multi-GPU heterogeneous systems are [9]. Although SPH and classical MD are both particle-based simulation techniques with strong similarities in their algorithms and data structures, they are intended for simulations of different types of systems. In MD the particles represent atoms, molecules, or coarse-grained model modelling of a material system, a continuum is discretised in SPH, and particles represent interpolation nodes. Consequently, interactions, boundaries and initial conditions are different. SPH in the form presented here is intended to simulate free-surface hydrodynamic flows, which inherently present abrupt variations of the density at the fluid surfaces, which, in turn, move rapidly during a simulation. MD, on the other hand, is typically tested on systems in which comparatively smaller fluctuations of density both in space and time, while at the same time tend to include a wider variety of particles and types of interactions. Consequently, one can expect that aspects of the problem, like the dynamic balancing of computing load among all available devices, be significantly different in SPH and in MD.

The multi-GPU scheme for SPH simulations presented herein has been implemented in the code DualSPHysics, which is an open-source code available at www.dual.sphysics.org, and is based on the SPH code named SPHysics [10,11]. The single GPU code has been shown to achieve two orders of magnitude speedups compared to the CPU approaches [12]. Thus, the new MPI implementation was designed starting from an already optimised DualSPHysics code for CPU and GPU [13].

In this paper, the SPH theory is described in Section 2. The single GPU and multi-GPU implementations are presented in Section 3 while the proposed MPI implementation is described in detail in Section 4. The results of the multi-GPU approach (improvements and drawbacks) are discussed in Section 5. Finally, in Section 6 the multi-GPU code DualSPHysics is applied to perform a huge simulation that requires high resolution over a large domain.

Thereby, the implementation for SPH method proposed here, enables not only simulations of more than 1 billion particles with a small numbers of GPUs, but also maintains high efficiency when increasing the number of computing devices.

## 2. Smoothed particle hydrodynamics method

Smoothed Particle Hydrodynamics (SPH) is a meshfree particle method that performs a local integration of the hydrodynamic equations of motion in a Lagrangian formalism where interpolation nodes are referred to as 'particles'. Relevant physical quantities are thus computed for each particle as interpolation weighted average over the values on the nearest neighbouring particles, and particles move according to those values. In the SPH formulation, the Navier–Stokes equations are solved and the fluid is treated as weakly compressible. The conservation laws of continuum fluid dynamics, in the form of partial differential equations, are transformed into their particle forms by integral equations through the use of an interpolation function that gives the kernel estimate of the field variables at a given point. Computationally, information is known only at discrete points (the particles), so that the integrals are evaluated as sums over neighbouring particles. For a more complete description of the SPH method and formulation used herein, the reader is referred to [10,14].

Mathematically, the contribution of the neighbouring particles is weighted according to distance using a kernel function ($W$) and a smoothing length ($h$). The smoothing length defines the area of influence of the kernel beyond which the contribution with the rest of the particles can be neglected.

The key idea is to approximate any function by the integral approximation:

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

These smoothing kernel functions must fulfil several properties [14], such as positivity inside a defined zone of interaction, compact support, normalisation and monotonically decreasing with distance. For instance, one possible choice is the quintic kernel by [15] where the weighting function vanishes for inter-particle distances greater than $2h$.

The function $F$ in Eq. (1) can be also expressed in a non-continuous form using the discrete form based on the particles. Thus, the approximation of the function is interpolated at a particle $a$ and a summation is performed over all the particles within the region of compact support of the kernel:

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

where $\Delta v_b$ is the volume of neighbouring particle $b$.

If $\Delta v_b = m_b/\rho_b$, with $m$ and $\rho$ being the mass and the density of particle $b$ respectively, Eq. (2) becomes:

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

The momentum equation proposed by [14] has been used to determine the acceleration of a particle ($a$) as the result of the particle interaction with its neighbours (particles $b$):

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b m_b\left(\frac{P_b}{\rho_b^2}+\frac{P_a}{\rho_a^2}+\Pi_{ab}\right)\nabla_a W_{ab} + \mathbf{g} \tag{4}$$

where $\mathbf{v}$ is velocity, $P$ is pressure, $\rho$ is density, $m$ is mass, $\mathbf{g}$ is gravitational acceleration and $W_{ab}$ the kernel function that depends on the distance between particle $a$ and $b$. $\Pi_{ab}$ is the viscous term according to the artificial viscosity proposed in [14].

The mass of each particle is constant, so that changes in fluid density are computed by solving the conservation of mass or continuity equation in SPH form [14]:

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab}. \tag{5}$$

Pressure is calculated starting from density using Tait's equation of state [16]:

$$P_a = B\left(\left(\rho_a/\rho_0\right)^{\gamma} - 1\right) \tag{6}$$

where $\gamma = 7$, $B = c_0^2\rho_0/\gamma$, $\rho_0$ is the reference density and $c_0$ is the speed of sound given by:

$$c_o = c\left(\rho_o\right) = \sqrt{(\partial P/\partial \rho)|_{\rho_o}}. \tag{7}$$

The time integration schemes must be at least second order since the particles represent computation points moving according to the governing dynamics. In particular, a Verlet [17] algorithm will be used in the present work. A variable time-step which depends on the CFL (Courant–Friedrich–Levy) condition, the forcing terms and the viscous diffusion term are used according to [18].

Different boundary conditions can be implemented in SPH to create a repulsive force that prevents fluid particles from penetrating the limits of the domain or solid objects. Here we use dynamic boundary conditions [19] which satisfy the same equations of continuity and state as the fluid particles, but their positions remain unchanged or are imposed externally. This boundary condition is easy to implement due to its computational simplicity where the fluid–boundary interactions can be calculated inside the same loops as fluid particles.

## 3. Parallel implementations

This section describes the different parallel implementations applied to the DualSPHysics code. They consist of dividing the computing load among the different processing units and adapting the algorithm to the specific features of each piece of hardware. For a full description of the SPH implementation on CPU and GPU, the reader is referred to [10–12] and is invited to download the documented source files of DualSPHysics at www.dual.sphysics.org.

The implementation of the SPH method consists of the iteration of three main steps: (i) generation of a neighbour list, (ii) calculation of forces by computing particle interactions and (iii) updating all physical magnitudes describing the state of the particles in the system. The most expensive step is the force computation since it takes more than 98% of the total runtime using DualSPHysics on a single CPU [13]. Therefore, efforts will be mostly focused on accelerating the particle interaction stage.

### 3.1. GPU using CUDA

The parallel programming architecture Compute Unified Device Architecture (CUDA) developed by Nvidia is used to obtain an efficient and extensive use of the capabilities of the GPU architecture. For SPH, the sequential tasks by which the particle dynamics are obtained are performed by threads of execution on the GPU. The three main steps of SPH ((i) neighbour search, (ii) force computation, (iii) system update) described above are implemented on the GPU. In this approach, there is first a memory transfer of particle data from CPU to GPU and then all particle information remains on the GPU memory throughout the simulation. When saving simulation information, such as particle positions, velocities, etc., only the necessary data are transferred from GPU to CPU. Since this task is performed occasionally, the computationally expensive CPU $\leftrightarrow$ GPU data transfers are minimised, making the program more efficient. The type of neighbour list implemented on the GPU is very similar to those typical of serial approaches used on CPUs, like the cell-linked list [20]. The use of the *radixsort* algorithm by CUDA thus improves the parallelisation of some operations during the creation of the cell-linked list. Updating the physical quantities of the particles can be easily parallelised using different execution threads of the GPU and the use of the reduction algorithm by CUDA also optimises the parallelisation of some tasks. Particle interaction, which, as mentioned above, is the most expensive part of the algorithm, is implemented on the GPU by executing one thread to compute the force for only one particle resulting from the interaction with all its neighbours. The thread looks for the neighbours of each particle among the adjacent cells and computes the resulting force considering all the interactions. Due to the Lagrangian nature of the method, different problems appear such as lack of balance of the computational load among the computing resources of the GPU, branching, code divergence and lack of perfect coalescent access to the global memory of the device. A complete description of the GPU implementation can be found in [12] and the applied optimisations are presented in [13].

### 3.2. Multi-GPU using MPI

A second level of parallelisation is applied by using MPI, where a set of directives enables communication between devices and allows combining the resources of several machines connected by a network. The execution power can therefore be increased easily by adding new machines. However, the division of the work load among different independent devices implies an extra computational cost. This extra runtime comes from; (i) the execution of new processes dedicated to manage the distribution of the work load, (ii) the time dedicated to data exchange and, (iii) the time consumed during synchronisations. These were investigated previously by [3].

The parallel implementation for SPH methods presented in this work use these parallelisation techniques with one or several machines connected in a network. This enables computations on heterogeneous clusters taking advantage of all available processing units. This is important since clusters can then be extended with new GPUs of different specifications. In the next section, we introduce a new implementation of this parallelism to obtain greater efficiencies from the additional hardware.

## 4. New MPI implementation

This section describes in more detail the proposed MPI implementation that will give rise to some interesting improvements but also to drawbacks (these will be assessed in Section 5).

The physical domain of the simulation is divided into subdomains among the different MPI processes. In this way, each process only needs to assign resources to manage a subset of the total amount of particles for each subdomain. Thus, the size of the simulation scales with the number of machines.

The two main sources of efficiency loss when the number of MPI processes is increased include data exchange between the devices managed by the processes and synchronisation. In the previous MPI implementation [3], it was observed that the time dedicated to communication constitutes a high percentage of the total execution time and that this percentage increases significantly with the number of processes. One option to reduce this time is to subdivide

the calculation of forces on each subdomain so that communications and computation can overlap. When considering synchronisation of processes across devices, the SPH algorithm benefits from using a variable time step computed following [18]. The new value is obtained from variables (force and viscous terms) that are known only after computing all particle interactions, that is, when all MPI processes have finished the force computation step. This synchronisation requires all processes to wait for the slowest process. Since the number of steps to complete a simulation is large, typically on the order of $10^4$–$10^6$, this implies a non-negligible loss of efficiency that also increases with the number of processes. To address this problem, the computational load or demand must be evenly divided among all processes, minimising the difference between the computation time needed for the fastest and the slowest process.

### 4.1. Subdivision of the domain

As mentioned above, the domain is divided into subdomains or blocks of particles that are assigned to the different MPI processes. This division can be performed in any direction ($X$, $Y$ or $Z$) adapting to the nature of the simulation case. In this way, each subdomain has two neighbouring subdomains, one on either side, except those subdomains at the perimeter of the simulation box, which have only one neighbour. Each MPI process needs to obtain, at every time step, the data of neighbouring particles from the surrounding processes within the interaction distance ($2h$ here). Therefore, to calculate the forces exerted on the particles within its assigned subdomain, each process needs to know the data of particles from the neighbouring subdomains that are located within the interaction distance. We call this the *halo* of the process (or subdomain) existing on the *edge* of the neighbouring process (or subdomain).

Fig. 1 shows the division of a domain into three subdomains (0, 1 and 2). Thus, grey particles belong to subdomain 1 but some of them, those that are in the left *edge* and at a distance less than $2h$ from domain 0, constitute the *halo* of subdomain 0 while the grey particles in the right *edge* constitute the *halo* of subdomain 2.

Unlike the scheme presented in [3], the data of *halo* particles of a given subdomain are not stored in the same data structure that holds the subdomain particles. Instead, in order to determine the *halo* of a given subdomain, information from a previous stage in the algorithm is used. That is, during the neighbour list stage, particles are sorted in cells of size $2h$ [13] and the order can be $XYZ$, $XZY$ or $YZX$ according to the division axis $Z$, $Y$ or $X$. As a consequence of this cell sorting, particles are also sorted within slices $2h$ wide within each subdomain. The subdomain assigned to each process is chosen to have a minimum width of $6h$ to ensure a minimum size of $2h$ for the two *edges* of that domain ($2h$ from left *edge* $\pm$ $2h$ inside the domain $\pm$ $2h$ from right *edge*).

This approach to divide the domain among the MPI processes where particles are reordered according to the direction of the domain subdivision gives rise to interesting advantages that increase performance:

- If particles of a subdomain are not merged with particles of the *halo*, no time is wasted in reordering all particles when receiving data from the *halo* before a force computation, and no time is wasted in separating them after the force computation. The memory usage is also reduced since only the basic properties of *halo* particles need to be stored (position, velocity and density).
- Each process can adjust the size of its subdomain with the limits of the fluid particles inside. With the number of cells minimised for each subdomain, the total number of cells over the entire is also minimised leading to a reduction of the execution time and memory requirements.
- Particle data of the subdomains are stored in slices. Data existing in the *edges* can be sent to the neighbouring processes much faster since all data are grouped in consecutive memory positions.
- This reordering system also enables automatic identification of particles contained in a subdomain needing to interact with the halo. Thus, task overlapping is possible by computing particle interactions of particles not on an *edge*, while simultaneously performing the reception of the *halo* (needed only by *edge* particles), thereby inherently saving the communication time. For example, the grey particles in Fig. 1 that belong to the left *edge* of the subdomain 1 also form the *halo* of subdomain 0 and force interactions between red particles not on the *edge* of subdomain 0 can be computed while they wait for the *halo*.
- Symmetry of pairwise interactions is not necessary for the particles of the *halo* since *halo* and *edge* particles only interact once in each process. This is relevant to the GPU implementation since symmetry is not applied for the pair-wise computations and does not represent any loss of performance in comparison to single-GPU version.

### 4.2. Communication among processes

Reducing time dedicated for exchanging data among MPI processes is essential to increase the number of processes without drastically decreasing efficiency. One method to achieve this is by overlapping the communication with the computation using asynchronous communications. Hence asynchronous send operations and synchronous receptions are used in the present algorithm. In this way, one process can send information to another while carrying out other tasks without waiting for the end of the transfer. This is an improvement over an algorithm that uses synchronous operations, in which an MPI process cannot continue execution of tasks until the operation is complete, implying a wait to receive data from another process or processes, thereby rendering computational resources idle, and consequently causing loss of efficiency.

Fig. 2 shows the data exchanges that take place at each time step when using MPI. Three different processes are considered in this example. There are two important communications; the first one occurs during the neighbour list creation (solid arrows in Fig. 2) and the second one in the force computation (dashed arrows in Fig. 2). The dark arrows represent the submissions from process $N$ to process $N+1$ and the light ones from process $N$ to process $N-1$. Double-headed arrows show the synchronisation point after the force computation stage. Note that all the tasks corresponding to interactions among particles correspond to the boxes with grey background in the figure.

At the beginning of each time step, during the neighbour list creation, each process looks for the particles that move from one subdomain to another and these displaced particles are sent to the corresponding process. This search is only performed checking the particles of the *edges* ($2h$ wide) of the domains since a particle cannot travel further than $2h$ during one time step. While data of displaced particles are sent (solid arrows in Fig. 2), the neighbour list of the particles in the *interior* of the domain (particles not in an *edge*) is processed. Finally, the new particles that entered the domain are received for each process and *all* particles are sorted. At this stage, the computation time that is overlapped with the transfer of particles is reduced, but this is not a problem because the number of particles that change from one domain to another at each step of the simulation is typically much smaller than the number of particles in a given subdomain. And this occurs regardless of the flow direction and flow speed, since the duration of the step is adjusted accordingly.

During the force computation, each process sends its *edges* to the surrounding processes (dashed arrows in Fig. 2). While *edges* are being sent and the *halo* is received, computation of the force on the *interior* particles is performed. Once this is finished, the process waits for the reception of the first *halo* and computes forces of one *edge* with this *halo*. After that, the process waits to receive

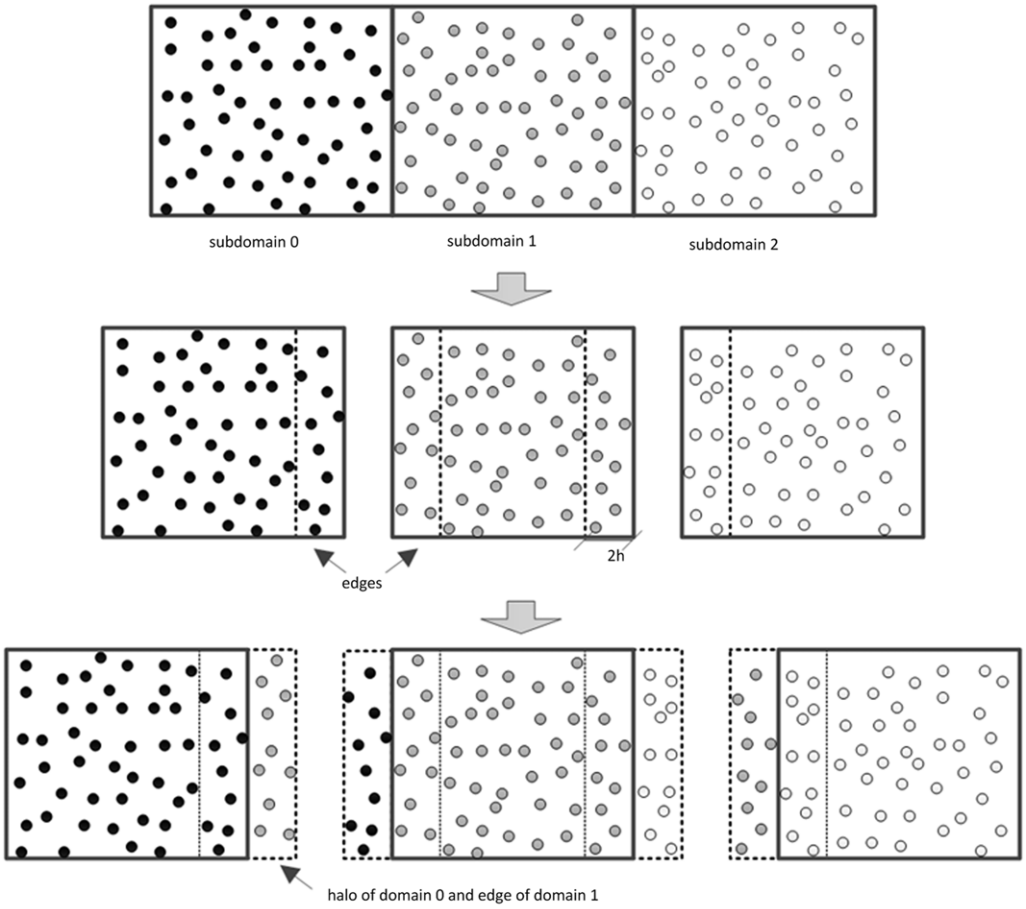


**Fig. 1.** Example of subdivision of a domain (*halos* and *edges*).

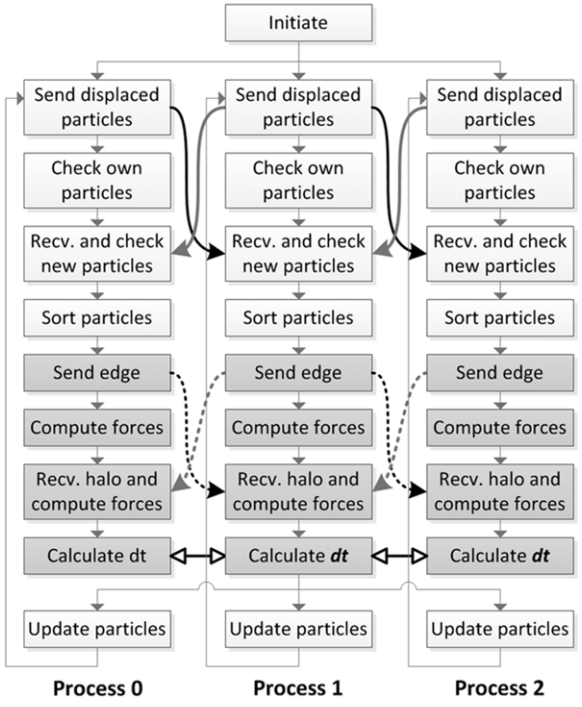


**Fig. 2.** Scheme of the communications among 3 MPI processes.

the second *halo* and computes the forces of the other *edge*. Thus the most expensive *halo-edge* data transfers are overlapped by calculating the forces between particles (also very time consuming). As explained in Section 3.1, all particle data are allocated in the GPU memory, so data transfer is also needed between CPU and GPU memories. However, it should be noted that the cost is negligible since the volume of information is low and one of the advantages of the method proposed here is that the data to be transferred are stored in contiguous memory locations, which accelerates the process.

### 4.3. Dynamic load balancing

Due to the Lagrangian nature of SPH, particles move through space during the simulation so the number of particles must be redistributed after some time steps to maintain a balanced work load among the processes and minimise the synchronisation time. Most of the total execution time is spent on force computation, and this time depends mainly on the number of fluid particles. For an equal load per processor, the domain must be divided into subdomains with the same number of fluid particles (including particles of the *halos*) or with the number of particles appropriate to the computing power of the device assigned to it.

Two different dynamic load balancing algorithms are used. The first one assigns the same *number* of particles to each computing device, and is suitable when the simulation is executed on machines that present the same performance. The second load balancing algorithm is used when hardware of different specifications and features are combined, such as different models of GPU. This second approach takes into account the *execution time on each device*. In particular, a weighted average of the computing time per integration step over several steps (on the order of 30) is used, with a

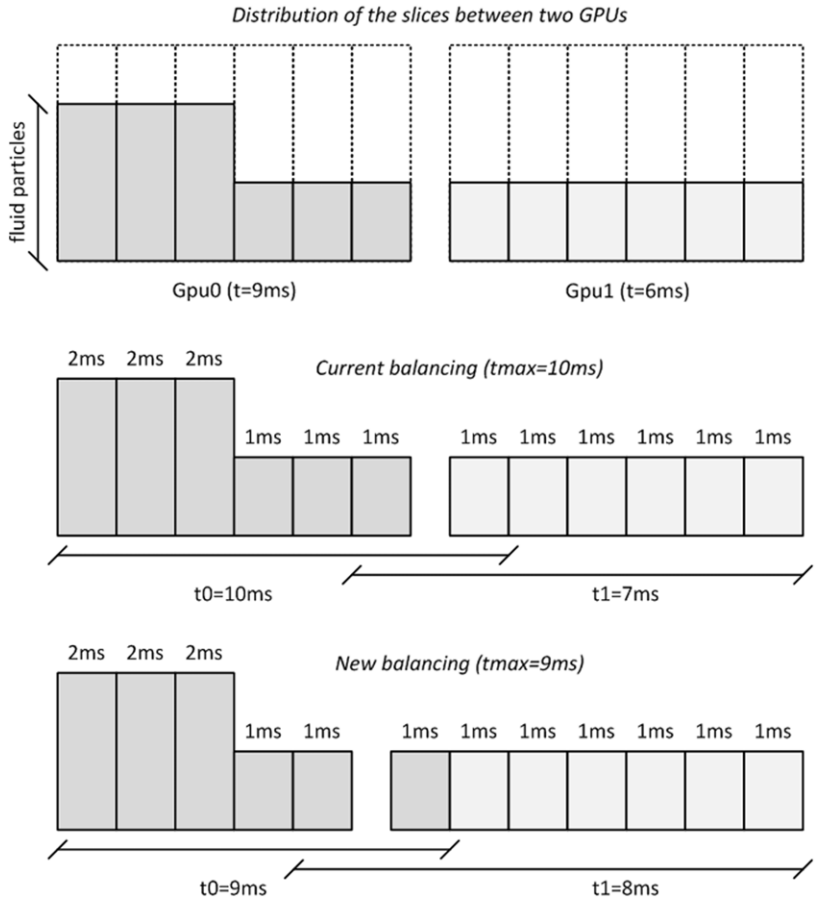


**Fig. 3.** Example of the dynamic balancing scheme between 2 GPUs.

higher weight to the most recent steps. An average over many time steps is chosen because a single time step presents large fluctuations. This average time is used to distribute the number of particles so that the fastest devices can compute subdomains with more particles than the slowest devices.

The example depicted in Fig. 3 can help to explain the second approach that takes into account the *execution time on each device* to balance the work load between GPUs. Thus, in the example, the first row in Fig. 3 shows the distribution of the slices between two GPUs at a given step where the average time of the force computation during the last 30 steps was 9 ms in the first GPU and 6 ms in the second one. Therefore, a new distribution of the slices between GPUs is desired where the maximum computation time must be minimal. The second row shows the actual time dedicated to force computation for each device since this time is the summation of the times required to compute forces of particles within the slices plus the particles of the *halo*, i.e. 9 ms + 1 ms = 10 ms for the first GPU and 6 ms + 1 ms = 7 ms for the second one. That is, the maximum computation time in this case is 10 ms and it is therefore desirable to apply a new balancing if the current maximum time can be reduced in a given percentage. In the third row of the figure, it can be seen how a redistribution of the slices between the two GPUs is performed where the second GPU (GPU1) will compute particles within one extra slice originating from GPU0. So that the maximum time has been reduced from 10 to 9 ms leading to an improvement of a 10% can be achieved with this example distribution.

This second type of dynamic load balancing enables the adaptation of the code to the features of a heterogeneous cluster achieving a better performance. Thus although the balance is checked every few steps, it is only applied when it implies an improvement in the performance, and therefore its cost is minimal. In fact, the runtime consumed by this checking is usually higher than the computational time dedicated to balance since this is not carried out very often.

## 5. Results

### 5.1. Testcases and hardware

Two testcases are used to analyse the performance of the new MPI–CUDA implementation. The first one is the experiment of Yeh and Petroff at the University of Washington, also described in [21] for validation of their 3D SPH model, used here to evaluate the proposed dynamic load balancing scheme. In the experiment, a dam break was reproduced within a rectangular tank, with a volume of water initially contained behind a thin gate at one end of the box and a tall structure placed inside the tank. A similar validation was carried out by [12] using data from the experiment of [22] to show the accuracy of DualSPHysics. Fig. 4 shows a sketch and several instants of the simulation of the testcase involving six million particles for a physical time of 1.5 s.

The second testcase is also a dam break similar to the previous one, but the main differences are that there is no obstacle in the middle of the numerical tank and the width of the tank can be modified according to the number of particles to be simulated. Note that modifying the width, the number of particles can vary keeping the same number of steps to complete the simulation and the same number of neighbouring particles of each particle. Thus, this testcase, shown in Fig. 5, is used to analyse the performance for different numbers of particles (from 1 to 1024 million) to simulate 0.6 s of physical time.

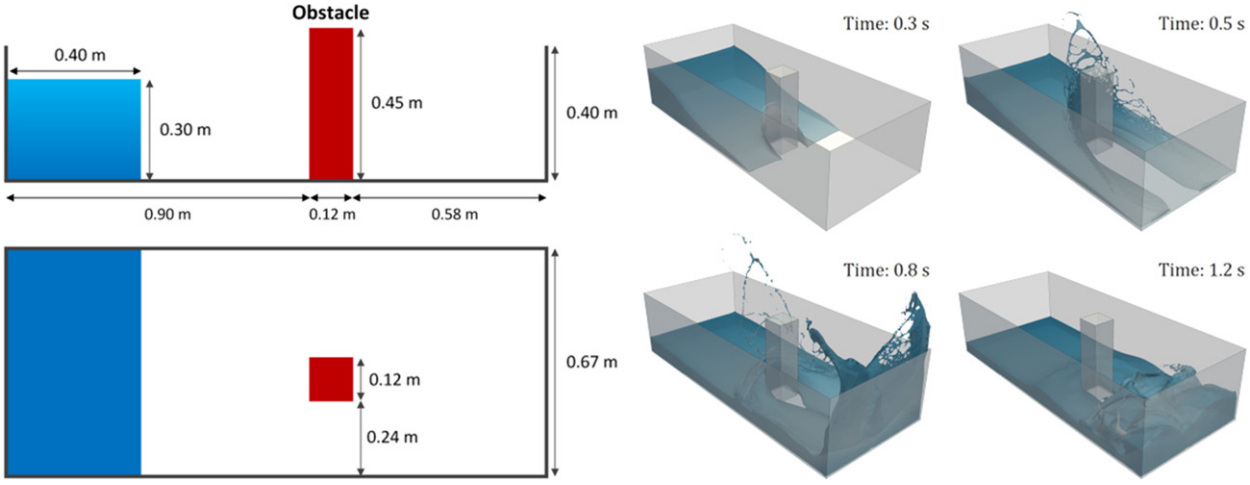


**Fig. 4.** Testcase1: dam break flow impacting on a structure.

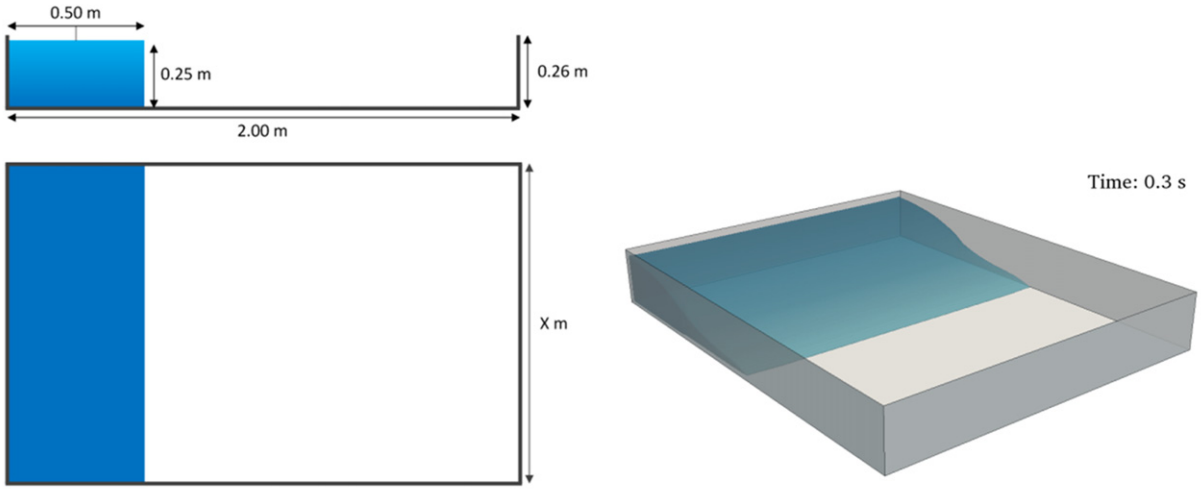


**Fig. 5.** Testcase2: dam break flow.

The simulations were carried out in four different systems at the University of Vigo (Spain), the University of Manchester (United Kingdom) and the Barcelona Supercomputing Center BSC-CNS (Spain). The specifications of each of those systems are summarised in Table 1. Two systems that belong to the University of Vigo (system #1a and system #1b) which have only one node were used to evaluate the different approaches of the dynamic load balancing (according to the number of particles and according to the time required for each machine). The system #2 (The University of Manchester) and system #3 (BSC-CNS) are built with several nodes (8 and 64 respectively) and they were used to analyse the performance and scalability (strong and weak scaling). The efficiency achieved using 8 nodes (16 GPUs) will be also confirmed analysing the efficiency with 64 nodes (128 GPUs). All the results presented in this work were obtained using CUDA 4.0, single precision and Error-correcting code memory (ECC) disabled.

First, the difference of the different load balancing schemes are compared for homogeneous and heterogeneous clusters.

### 5.2. Applying dynamic load balancing in a homogeneous cluster

This section presents the results when using the dynamic load balancing according to the number of particles. Testcase1 (Fig. 4) is simulated using system #1a (3 × GTX 480) so the domain is divided in 3 processes (3 GPUs) along the $x$-direction. Different instants of the simulations are shown in Fig. 6. The limits of the three different subdomains are depicted using coloured boxes. The size of the different subdomains changes with time to keep the work load evenly distributed among processes (similar number of particles per process).

The left panel of Fig. 7 shows the distribution of the fluid particles among the 3 processes showing how the balancing is achieved since about the 33.33% of the particles are always computed for each process during the simulation. Since system #1a is homogeneous with the same three GPUs, the time dedicated to the force computation step for each GPU is also balanced as seen in the right panel of Fig. 7. A total amount of 42,624 steps were performed to complete this simulation, the balancing was checked every 50 steps, so 852 times (0.04% of the total simulation time) but it was performed only 94 times (0.03% of the total simulation time).

### 5.3. Applying dynamic load balancing in a heterogeneous cluster

The dynamic load balancing scheme was also applied in the same testcase1 but using system #1b, which is a heterogeneous system since the 3 GPUs present different specifications and performances. From Fig. 8 it can be concluded that the approach of the balancing according to the number of particles is not suitable in this

**Table 1**
Features of the different systems.

| | | |
|---|---|---|
| *System* #1 | *Software* | – CentOS 5.5<br>– MPICH2 1.2.1<br>– CUDA 4.0<br>– gcc 4.1.2 |
| | *Hardware* | *System* #1a: 1 *homogeneous* node with:<br>– 2 × Intel Xeon E5620 (4 cores at 2.4 GHz with 16 GB RAM)<br>– 4 × GTX 480 Fermi (15 Multiprocessors, 480 cores at 1.40 GHz, 1.5 GB GDDR5, Compute capability 2.0)<br>*System* #1b: 1 *heterogeneous* node with:<br>– 2 × Intel Xeon E5620 (4 cores at 2.4 GHz with 16 GB RAM)<br>– 1 × GTX 680 Kepler (8 Multiprocessors, 1536 cores at 1.14 GHz, 2 GB GDDR5, Compute capability 3.0)<br>– 1 × GTX 480 Fermi (15 Multiprocessors, 480 cores at 1.40 GHz, 1.5 GB GDDR5, Compute capability 2.0)<br>1 × GTX 285 (30 Multiprocessors, 240 cores at 1.48 GHz, 1 GB GDDR5, Compute capability 1.3) |
| *System* #2 | *Software* | – Red Hat Enterprise Linux Server 6.2<br>– Open MPI 1.5.4<br>– CUDA 4.0<br>– gcc 4.4.6 |
| | *Hardware* | 8 nodes connected via QDR Infiniband with:<br>– 2 × Intel Xeon L5640 (6 cores at 2.27 GHz with 24 GB RAM)<br>– 2 × Tesla M2050 (14 Multiprocessors, 448 cores at 1.15 GHz, 3 GB GDDR5, Compute capability 2.0) |
| *System* #3 | *Software* | – Red Hat Enterprise Linux Server 6.0<br>– BullxMPI 1.1.11<br>– CUDA 4.0<br>– Intel C++ Compiler XE 12.0 |
| | *Hardware* | 128 nodes connected via QDR Infiniband with:<br>– 2 × Intel Xeon E5649 (6 cores at 2.53 GHz with 24 GB RAM)<br>– 2 × Tesla M2090 (16 Multiprocessors, 512 cores at 1.30 GHz, 6 GB GDDR5, Compute capability 2.0) |

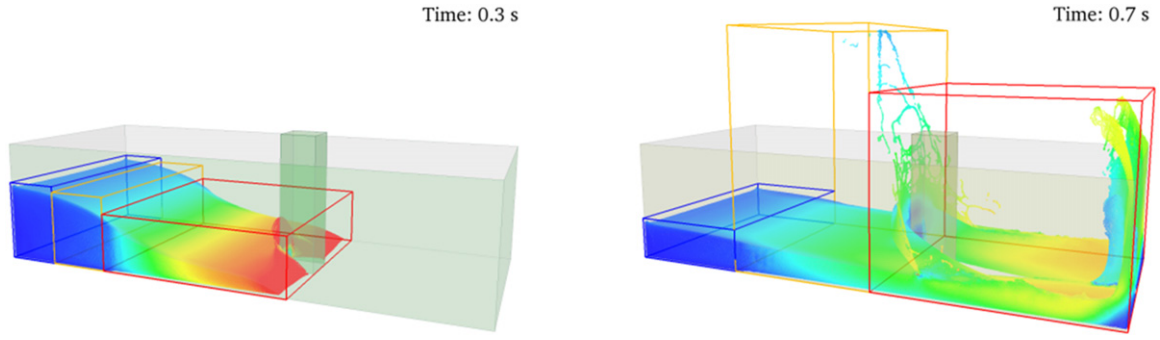


**Fig. 6.** Different instants of the simulation of testcase1 when using the dynamic load balancing according to the number of particles.

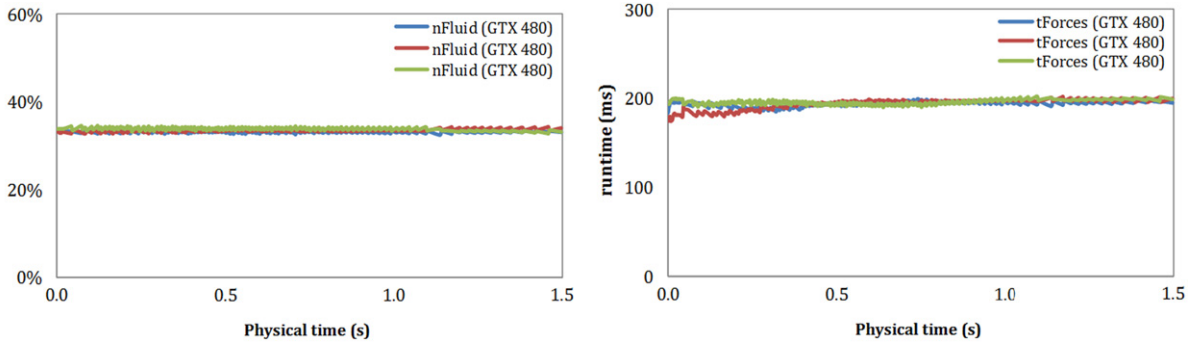


**Fig. 7.** Distribution of the fluid particles and execution times of force computation among the 3 GPUs of system #1a using load balancing according to the number of particles.

case since despite the even distribution of the number of particles among the processes, the computation times are not balanced at all. The GPU card GTX 285 is much slower than the other two cards and the time required to compute the same number of particles is considerably higher than needed for the other two cards.

The second proposed option consists of the algorithm described in Section 4.3 based on the computation time required to compute the forces. In this way, a number of particles is assigned to each GPU card according to its performance to get a correct balance of the work load among the different GPUs. Fig. 9 shows the different distribution of particles assigned to the three GPUs of the system #1b and the execution times to compute the particle interactions, which are very similar. Thus, the slowest card no longer presents a bottleneck.

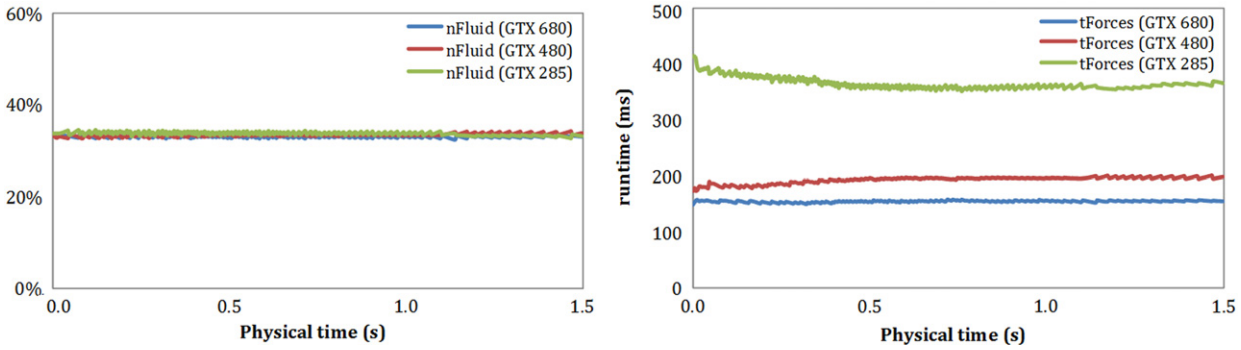


**Fig. 8.** Distribution of the fluid particles and execution times of force computation among the 3 different GPUs of system #1b using load balancing according to the number of particles.

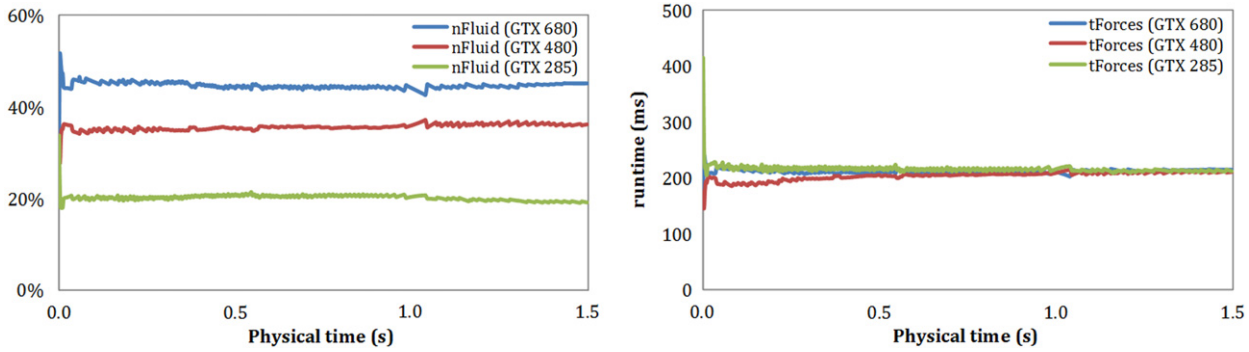


**Fig. 9.** Distribution of the fluid particles and execution times of force computation among the 3 different GPUs of system #1b using load balancing according to the computation time.

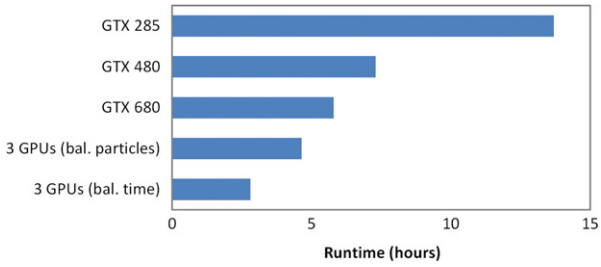


**Fig. 10.** Execution times of the 3 GPUs of the system #1b used individually and together applying dynamic load balancing.

The execution of testcase1 on one GTX 680 card takes 5.8 h; combining this GPU with GTX 285 and GTX 480, whose performance characteristics are lower, the run takes 4.6 h applying the dynamic load balancing according to the number of fluid particles, and only 2.8 h applying the balancing based on the computation time of each device. Fig. 10 summarises the execution times of the 3 GPUs of the system #1b when used individually and together.

### 5.4. Efficiency and scalability

One of the main objectives of the proposed multi-GPU implementation using MPI is the possibility of simulating large systems ($10^7$–$10^9$ particles) at reasonable computational times, which is imperative to use the model in real-life applications that require high resolutions. An efficient use of the resources to minimise the computational and economical cost will make these large scale simulations viable. Therefore, a study of the efficiency and scalability of the multi-GPU implementation is shown in this section.

The performance is measured as the number of steps computed per second using two approaches; (i) strong scaling $S(N)$ that determines how the solution time $T$ varies with the number of processors $N$ for a fixed total problem size; and (ii) weak scaling $s(N)$ that defines how the solution time varies with the number of processors for a fixed problem size per processor. Table 2 shows the formulae used to measure the speedups and efficiency using these two measures.

**Table 2**
Formulae to measure efficiency and scalability.

| | |
|---|---|
| Strong scaling | $S(N) = \frac{T(N_{\text{ref}})}{T(N)}$ |
| Weak scaling | $s(N) = \frac{T(N_{\text{ref}})\cdot N}{T(N)\cdot N_{\text{ref}}}$ |
| Efficiency | $E(N) = S(N)/N$ |

Testcase2 (Fig. 5) is used here to evaluate the performance using the different number of GPUs of the systems #1, #2 and #3. The achieved speedups are shown in Fig. 11 analysing the strong scaling (left) and the weak scaling (right) for the different hardware systems.

For system #1 with only 4 Fermi GTX 480, the speedup is shown for 2, 3 and 4 GPUs simulating the testcase2 with sizes ranging from 1 to 8 M particles for strong scaling and from 1 to 8 M particles per GPU to quantify the weak scaling. As expected, the efficiency decreases with the number of GPUs. Thereby, using 4 GPUs and analysing the strong scaling, an efficiency of only 66% is achieved simulating 1 M particles but 94% is achieved when simulating 8 M because the proportion of time spent on communication is far smaller. Examining the weak scaling, an efficiency of 85.6% is obtained simulating 1 M particles per GPU, but this value increases to 99.9% computing 8 M per GPU.

In the case of the system #2 with 16 T M2050, the efficiency analysing the strong scaling is significantly reduced when the number of GPUs increases for a small number of particles. For example, an efficiency of 50% is obtained simulating 1 M particles with 8 GPUs, but this amount cannot be simulated with more than 8 GPUs. As mentioned before, a minimum width of $6h$ is assigned to

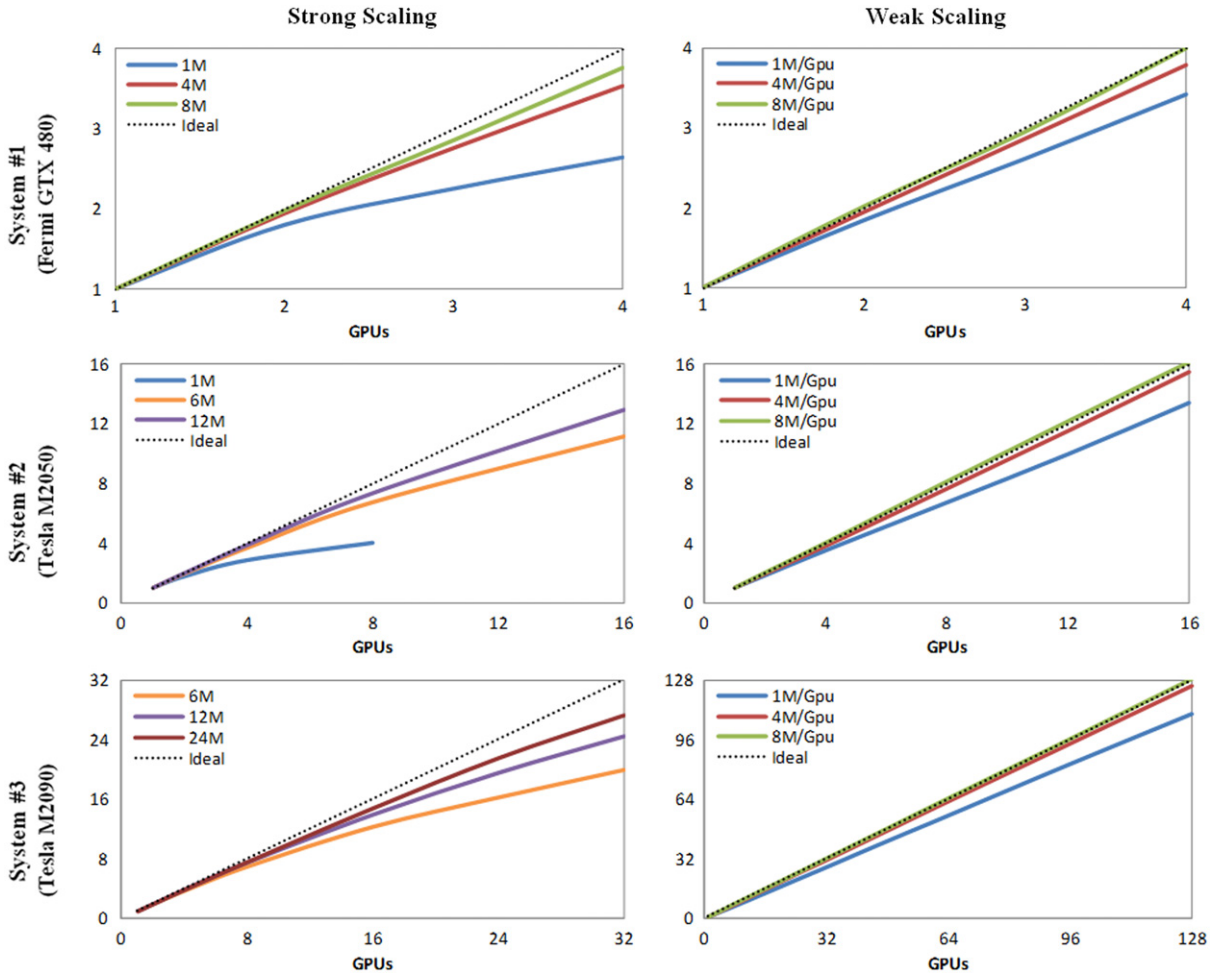


**Fig. 11.** Speedup for different number of GPUs using strong and weak scaling with the hardware systems #1, #2 and #3.

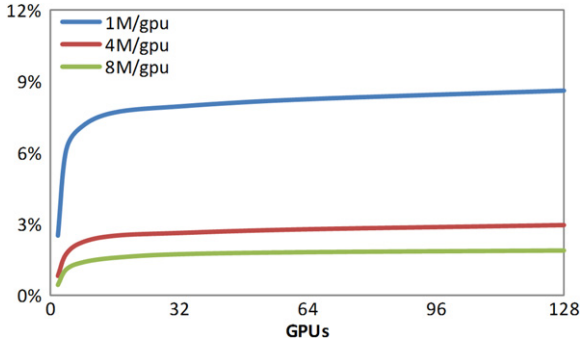


**Fig. 12.** Percentage of time dedicated to tasks exclusive of the multi-GPU executions using the system #2.

the subdomain of each process, so the maximum number of GPUs is restricted. The simulation of 12 M particles using the 16 GPUs of the system #2 provides an efficiency of 81%. The results for the weak scaling are 96.8% simulating 4 M particles per GPU and higher than 99.9% when 8 M per GPU are performed.

Finally, for the system #3 using a maximum of 128 T M2090, an efficiency of 97.4% is achieved simulating 4 M per GPU and higher than 99.9% with 8 M per GPU. Note that the highest execution simulated with this system simulates 1024 M (128 GPUs $\times$ 8 M) to study the weak scaling.

Values of efficiency higher than 99.9% are obtained since the testcase2 does not scale perfectly. In spite of the efforts to choose a case where the execution time per number of particles was the same for different sizes, this was not possible due to different factors such as the ratio between fluid and boundary particles. Thus, the execution time of one step per million particles is slightly higher with the case of 8 M than in the cases with more than 32 M. Therefore, when using the case of 8 M as reference to compute the results of weak scaling in comparison to bigger cases, the efficiency is slightly higher than the expected. Fig. 12 shows the percentage of time dedicated to tasks exclusive of the multi-GPU simulations, which represent the overcost comparing to single-GPU. It can be observed how these tasks take less than 1.9% when simulating 8 M/GPU with 128 GPUs and increases to 3% and 9% with 4 M/GPU and 1 M/GPU respectively.

### 5.5. Bottlenecks: loss of efficiency

This section discusses the origin of the loss of efficiency when the number of particles per GPU is low. The MPI implementation requires tasks that represent an extra overhead thereby reducing efficiency. Synchronisation tasks and communication between processes are unavoidable and their cost increases with the number of processes. The only way to minimise their impact is to increase the number of particles per GPU. In the previous section, it was shown how efficiency about 99.9% is achieved by simulating 8 M particles per GPU, but the efficiency drops significantly when simulating 1 M per GPU or less. The main causes of this loss of efficiency can be analysed using Fig. 13 which shows the percentage of computational time dedicated to the synchronisation tasks

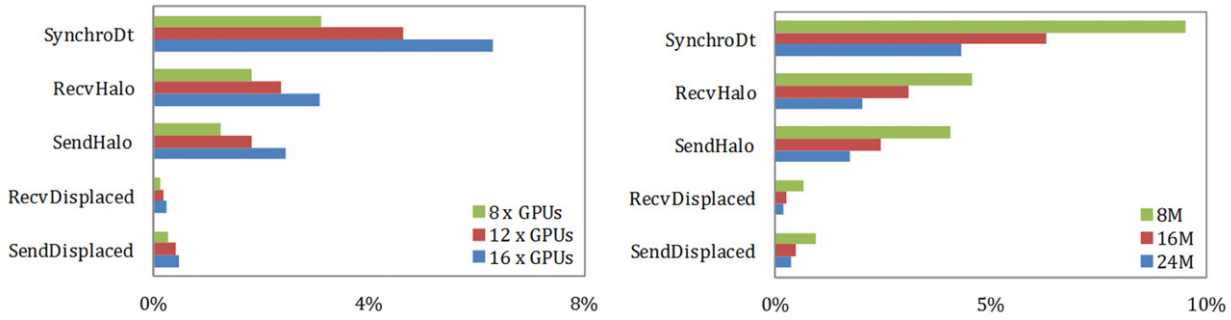


**Fig. 13.** Percentage of the computational time dedicated to specific MPI tasks simulating 16 M particles using different number of Tesla M2050 GPUs (left) and simulating different number of particles with 16 T M2050 (right).

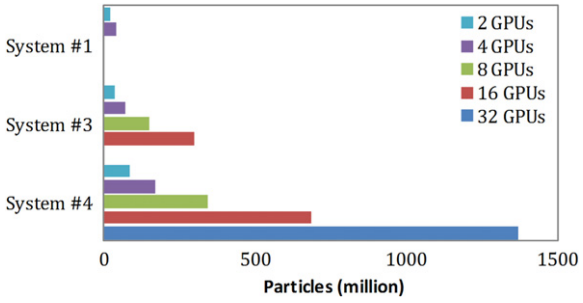


**Fig. 14.** Maximum number of particles that can be simulated for the testcase2 with the systems #1, #2 and #3.

(*SynchroDt*), reception and transmission of the *halos* (*RecvHalo* and *SendHalo*) and reception and submission of the particles that have moved among domains (*RecvDisplaced* and *SendDisplaced*). Results of the simulations of 16 M in 8, 12 and 16 GPUs are shown in the left panel, while results of the simulation of 8, 16 and 24 M in 16 GPUs are shown in the right panel. These plots reflect how the synchronisation is the first cause of loss of efficiency and the second one is the data exchange of the *halos*. Hence, this loss of efficiency increases with the number of GPUs (left panel of Fig. 13) but decreases with the number of particles (right panel of Fig. 13), so the loss of efficiency increases by reducing the number of particles per GPU.

### 5.6. Memory requirements

Earlier in this article, it was mentioned that the use of GPUs is an attractive alternative to accelerate the SPH simulations, but the limited GPU memory can be a serious drawback for very large systems. Therefore, one of the objectives of this multi-GPU implementation is to eliminate this limitation, so it is necessary to ensure an efficient use of memory for all the GPUs used in the simulation.

In the code version without MPI, all the required memory is allocated at the beginning of the execution since the maximum number of cells and particles is known *a priori*. However, in the multi-GPU version, this information is unknown for each GPU and the number of particles and cells allocated in each GPU change during the simulation. To solve this problem, the amount of memory needed for the particles and cells and an extra 10% is allocated initially, and only when this memory is no longer enough, a new allocation is performed. The maximum number of particles that can be simulated with this multi-GPU implementation for the testcase2 is about 7.14 million particles per GB of memory using single precision. As it can be observed in Fig. 14, a maximum of 40 M can be simulated with the 4 GPUs of the system #1, 300 M with the 16 GPUs of the system #2 and more than 1370 M with 32 GPUs of system #3.

## 6. Applicability to realistic problems

As mentioned above, one of the main objectives of the multi-GPU code DualSPHysics is simulating real-life applications that require high resolution over a large domain. Thus, once the different algorithms have been described and their efficiency and main drawbacks have been discussed, the code is now applied to perform a huge simulation with more than $10^9$ particles. This application consists of the interaction of a large wave with an oil rig using realistic dimensions and simulating 12 s of physical time. The fluid domain is 170 m $\times$ 114 m $\times$ 68 m and the dimensions of the platform can be seen in Fig. 15. The initial inter-particle distance is 6 cm which implies a simulation of 1,015,896,172 particles (1,004,375,142 fluid particles). This real application has been chosen since a huge number of particles is required to represent with very high resolution the smallest spatial scales in some objects of the oil platform (on the order of centimetres) and also need to describe properly the propagation of large waves (with wavelengths on the order of one hundred metres).

The simulation was carried out using 64 GPUs Tesla M2090 of the hardware system #3. Different instants of the simulation can see in Fig. 16. A total number of 237,342 steps have been carried out in 91.9 h. Data were saved every 0.04 s of physical time, which represents more than 8980 GB of output data.

## Conclusions and future work

A new parallel implementation for SPH methods has been developed. The new approach includes CUDA and MPI programming languages to combine the parallel performance of several GPUs, in a host machine or in multiple machines connected by a network.

The multi-GPU implementation presented here has shown a high efficiency using a significant number of GPUs. Thus, using 128 GPUs of the Barcelona Supercomputing Center (which represents the 50% of the total amount of GPUs of one of the main clusters in Europe), efficiencies of 85.9%, 97.4% and close to 100% have been achieved simulating 1 M/GPU, 4 M/GPU and 8 M/GPU respectively.

The possibility of combining the resources of several GPUs and the efficient use of the memory enables simulations with a huge number of particles. For example, 40 M particles can be simulated with 4 GPUs GTX 480, more than 300 M with 16 GPUs M2050 and more than 2000 M with 64 GPUs M2090.

To show the capabilities of the code, a realistic interaction of a large wave with an oil rig using more than $10^9$ particles have been carried out. In this example, 12 s of physical time were executed in less than 92 h using 64 GPUs M2090.

Therefore, the efficiency and performance of the new MPI–CUDA implementation of DualSPHysics are presented and analysed in this work. The main contributions of this manuscript can be summarised as follows:

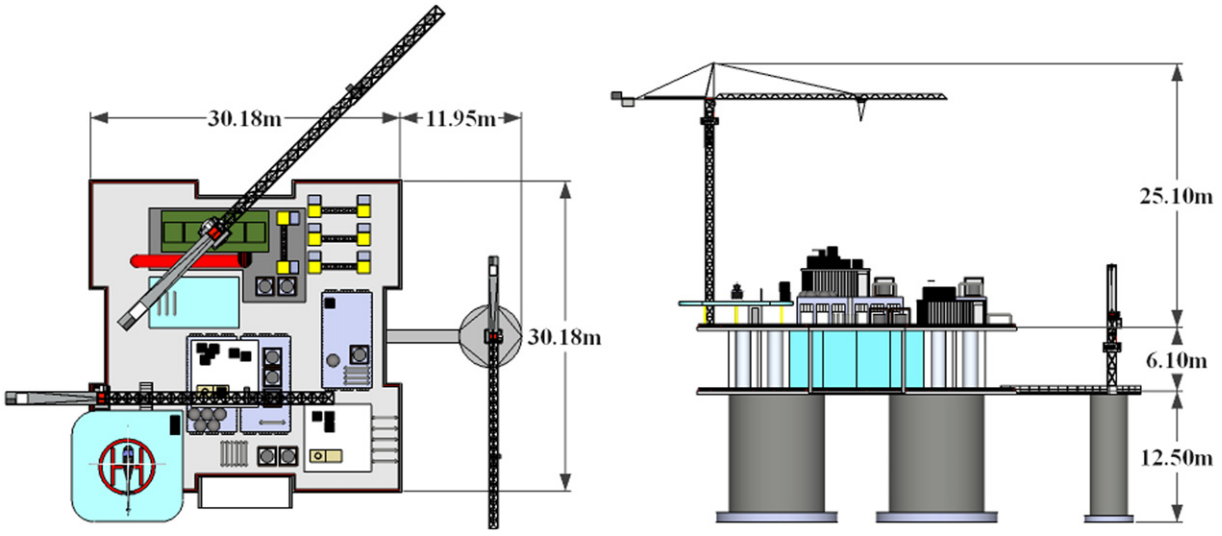


**Fig. 15.** Realistic dimensions of the oil rig simulated in the application.

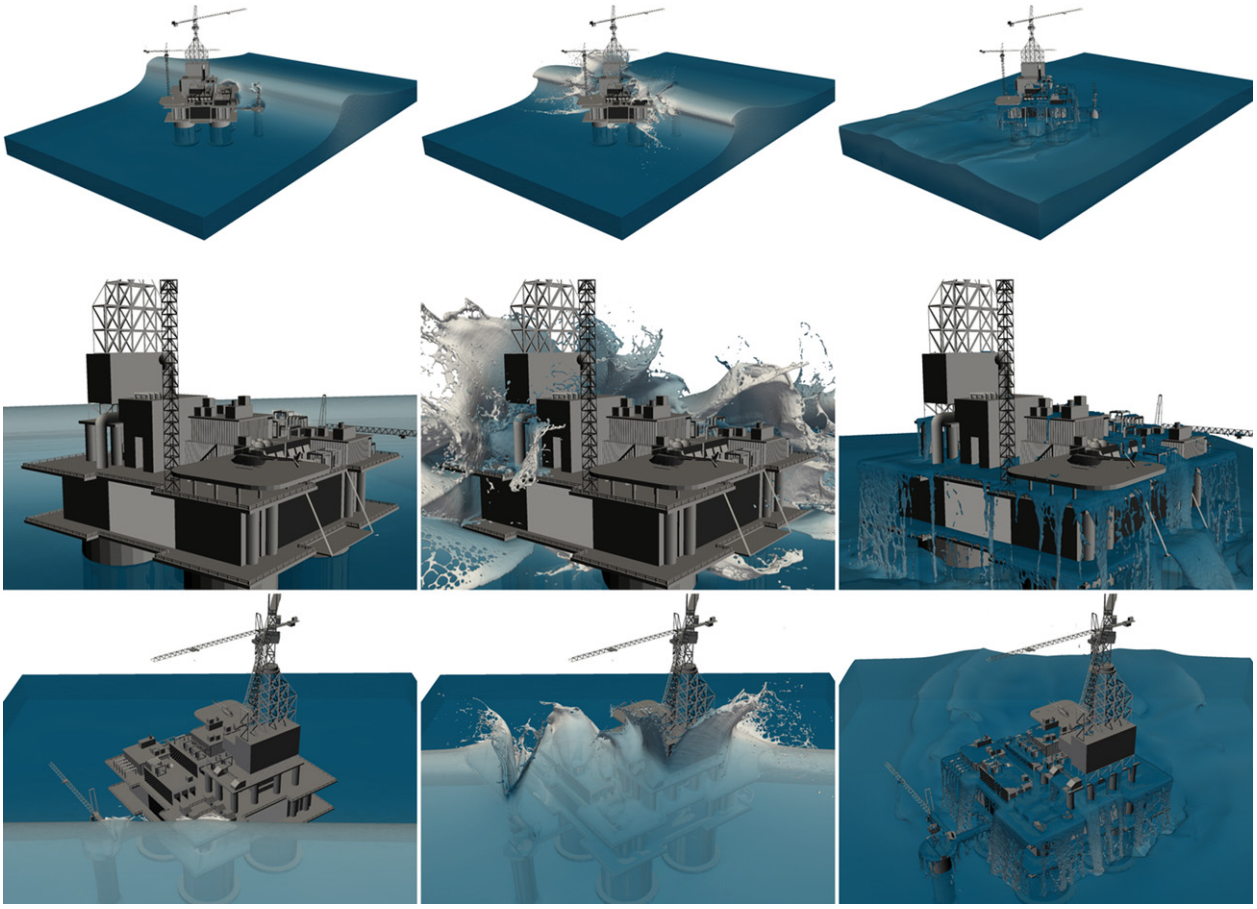

**Fig. 16.** Different instants (2.2, 3.2 and 10 s) of the simulation of a large wave interacting with an oil rig using more than $10^9$ particles.

- A dynamic load balancing is implemented to distribute work load across the multiple processes to achieve optimal resource utilisation and minimise response time.
- Overlapping between data communications and computations tasks is introduced to compensate latency and to reduce computational times.
- The proposed multi-GPU code can be executed on different GPUs with identical specifications or old and new cards can be exploited together. Thus, the heterogeneous version allows a more efficient use of different machines with different GPU cards.
- The scalability is analysed in terms of strong and weak scaling, indicating how the runtime varies with the number of processes for a fixed total problem size and how it varies with the number of processes for a fixed problem size per processor.

- The simulation of billions particles is possible in medium-size clusters of GPUs.

Different developments can now be considered. The use of double precision for the position variables is imperative in cases with one dimension much larger than the others to maintain the accuracy of the results. A 2D and 3D decomposition must be also a future development since it would be imperative to carry out simulation using hundreds of GPUs.

## Acknowledgements

This work was supported by Universidade de Vigo under project INOU12-03, and was also supported by Xunta de Galicia under project Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas (Grupos de Referencia Competitiva), financed by European Regional Development Fund (FEDER) and by Ministerio de Economía y Competitividad under de Project BIA2012-38676-C03-03. The authors gratefully acknowledge the support of EPSRC EP/H003045/1 and a Research Councils UK (RCUK) fellowship. The authors thank Simon Hood at the University of Manchester and Orlando Garcia Feal at the University of Vigo, for their support solving hardware and software issues. And finally we would also like to thank Barcelona Supercomputing Center (BSC-CNS) for the use of their facilities under the activities FI-2012-2-0006 and FI-2012-3-0004.

## References

[1] D. Violeau, Fluid Mechanics and the SPH Method: Theory and Applications, Oxford University Press, ISBN: 0199655529, 2012.
[2] Nvidia, CUDA Programming Guide, 4.0., 2011. http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf.
[3] D. Valdez-Balderas, J.M. Dominguez, A.J.C. Crespo, B.D. Rogers, Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters, Journal of Parallel and Distributed Computing (2012) http://dx.doi.org/10.1016/j.jpdc.2012.07.010.
[4] F. Fleissner, P. Eberhard, Load balanced parallel simulation of particle–fluid DEM-SPH systems with moving boundaries, Parallel Computing: Architectures, Algorithms and Applications 48 (2007) 37–44.
[5] P. Maruzewski, D. Le Touzé, G. Oger, F. Avellan, SPH high-performance computing simulations of rigid solids impacting the free-surface of water, Journal of Hydraulic Research 48 (2010) 126–134.
[6] W.M. Brown, P. Wang, S.J. Plimpton, A.N. Tharrington, Implementing molecular dynamics on hybrid high performance computers—short range forces, Computer Physics Communications 182 (2011) 898–911.
[7] C.R. Trott, L. Winterfeld, P.S. Crozier, General-purpose molecular dynamics simulations on GPU-based clusters, Computer Physics Communications (2012) arXiv:1009.4330v2.
[8] P.K. Agarwal, S. Hampton, J. Poznanovic, A. Ramanthan, S.R. Alam, P.S. Crozier, Performance modeling of microsecond scale biological molecular dynamics simulations on heterogeneous architectures, Concurrency and Computation: Practice and Experience (2012) http://dx.doi.org/10.1002/cpe.2943.
[9] W. Qiang, Y. Canqun, T. Tao, L. Kai, Fast parallel cutoff pair interactions for molecular dynamics on heterogeneous systems, Tsinghua Science and Technology 17 (2012) 265–277.
[10] 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.
[11] 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.
[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. http://dx.doi.org/10.1371/journal.pone.0020685.
[13] 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 (2012). http://dx.doi.org/10.1016/j.cpc.2012.10.015.
[14] J.J. Monaghan, Smoothed particle hydrodynamics, Annual Review of Astronomy and Astrophysics 30 (1992) 543–574.
[15] H. Wendland, Piecewiese polynomial, positive definite and compactly supported radial functions of minimal degree, Advances in Computational Mathematics 4 (1995) 389–396.
[16] G.K. Batchelor, Introduction to Fluid Dynamics, Cambridge University Press, UK, 1974.
[17] L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Physical Review 159 (1967) 98–103.
[18] J.J. Monaghan, A. Kos, Solitary waves on a Cretan beach, Journal of Waterway, Port, Coastal and Ocean Engineering 125 (3) (1999) 145–154.
[19] A.J.C. Crespo, M. Gómez-Gesteira, R.A. Dalrymple, 3D SPH simulation of large waves mitigation with a dike, Journal of Hydraulic Research 45 (5) (2007) 631–642.
[20] 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.
[21] 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.
[22] K.M.T. Kleefsman, G. Fekken, A.E.P. Veldman, B. Iwanowski, B. Buchner, A volume-of-fluid based simulation method for wave impact problems, Journal of Computational Physics 206 (2005) 363–393.