II International Conference on Particle-based Methods - Fundamentals and Applications
PARTICLES 2011
E. Oñate and D.R.J. Owen (Eds)

PARTICLE METHODS PARALLEL IMPLEMENTATIONS BY GP-GPU STRATEGIES.

ANTÓN REY-VILLAVERDE*, JOSE L. CERCÓS* , ANTONIO SOUTO-IGLESIAS* AND LEO M. GONZÁLEZ*

*Canal de Ensayos Hidrodinmicos de la ETSI Navales (CEHINAV)
Universidad Politécnica de Madrid
Avda. de la Victoria s/n , 28040 Madrid, Spain
e-mail: leo.gonzalez@upm.es, http://canal.etsin.upm.es/

Key words: OpenCL, CUDA, GPU, speed-up

Abstract. This paper outlines the problems found in the parallelization of SPH (Smoothed Particle Hydrodynamics) algorithms using Graphics Processing Units. Different results of some parallel GPU implementations in terms of the speed-up and the scalability compared to the CPU sequential codes are shown. The most problematic stage in the GPU-SPH algorithms is the one responsible for locating neighboring particles and building the vectors where this information is stored, since these specific algorithms raise many difficulties for a data-level parallelization. Because of the fact that the neighbor location using linked lists does not show enough data-level parallelism, two new approaches have been proposed to minimize bank conflicts in the writing and subsequent reading of the neighbor lists. The first strategy proposes an efficient coordination between CPU-GPU, using GPU algorithms for those stages that allow a straight forward parallelization, and sequential CPU algorithms for those instructions that involve some kind of vector reduction. This coordination provides a relatively orderly reading of the neighbor lists in the interactions stage, achieving a speed-up factor of x47 in this stage. However, since the construction of the neighbor lists is quite expensive, it is achieved an overall speed-up of x41. The second strategy seeks to maximize the use of the GPU in the neighbor's location process by executing a specific vector sorting algorithm that allows some data-level parallelism. Although this strategy has succeeded in improving the speed-up on the stage of neighboring location, the global speed-up on the interactions stage falls, due to inefficient reading of the neighbor vectors. Some changes to these strategies are proposed, aimed at maximizing the computational load of the GPU and using the GPU texture-units, in order to reach the maximum speed-up for such codes. Different practical applications have been added to the mentioned GPU codes. First, the classical dam-break problem is studied. Second, the wave impact of the sloshing fluid contained in LNG vessel tanks is also simulated as a practical example of particle methods.

1 INTRODUCTION

The recent implantation of graphic process units (GPUs) in scientific computation has increased drastically the speed processing in several applications. Not many years ago, parallel computing was restricted to super-computing centers or large and expensive clusters. Nowadays, thanks to the arrival of GPU multicore processors, originally designed for graphic processing, massively parallel processing is becoming increasingly more accessible and cheaper for the developer.

Increasing the efficiency of algorithms involved not only depends on the specific hardware improvements, but also on the new approaches aimed at maximizing available resources and minimizing costs. In the case of GPU processors, it is necessary to note that the computational power computer lies in its specialization. The GPU multicore architecture is designed for highly efficient graphic processing. This requires the implementation of affine and projective orthogonal transformations (matrix operations) on a set of elements(vertices, fragments) between different spaces of the 3D graphic scene. Likewise, the level of independence between these elements is perfect for parallel processing through which it is possible to distribute among a large number of processors such arithmetic operations on each of the vertices and fragments of the scene, commonly arranged as large vectors.

In order to exploit this powerful calculation, the GP-GPU emerges as a discipline where graphics shader functions are redesigned for processing data vectors, not necessarily graphics, which might exhibit a high level of parallelism from the point of view of SIMD (Single Instruction-Multiple Data) architectures.

To explore the degree of adaptability of the GPU technology to certain algorithms which simulate large particle systems, first we analyze which steps of the SPH (Smoothed Particle Hydrodynamics) code are more suitable to be parallelized, as well as different strategies of parallelization of the main subroutines. This requires the evaluation of the problematic aspects and the consequent speedup and scalability obtained.

Since the SPH methodology generally uses an explicit resolution scheme, their algorithms are easily parallelized on its minimum units (particles, cells). However, there are certain subroutines whose GPU parallelization is not immediate, in those cases different strategies will be implemented. These strategies will be focused on the obtention of the maximum parallelization implementation or the increase of the CPU versatility. Although it is always possible to use the CPU in those subroutines whose parallelization is problematic, this should be avoided due to the relatively high latencies associated with data transfers between CPU and GPU and the consequent reductions in computational performance.

2 GP-GPU: CUDA IMPLEMENTATION.

Traditionally, the GP-GPU has been developed using special languages (shaders) as GLSL, CG, or HLSL incorporated as extensions of the OpenGL and Direct3D APIs.

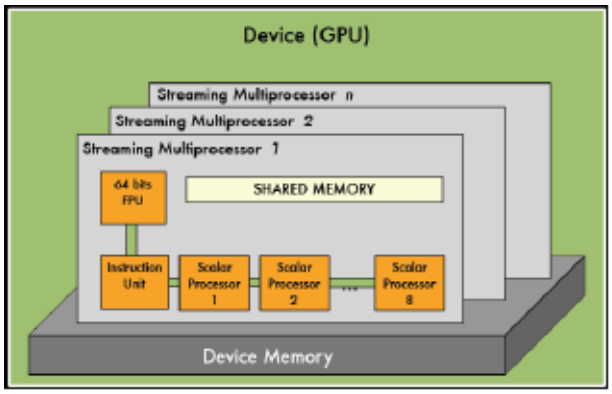

Figure 1: Basic architecture of a GPU card.

Learning GP-GPU programming not only required graphic programming as prerequisite, but also requires a considerable expertise on APIs and graphic languages. Currently, developing GP-GPU applications is done with the help of programming environments specifically designed to develop this type of codes as the CUDA (Computing Unified Device Architecture) or the OpenCL platforms.

2.1 Generalities.

Regarding GPU architecture, in Figure 1 a scheme of the distribution of the processors on the graphic cards used is shown. Basically, the GPU distributed in a set of multiprocessors. Each multiprocessor hosts typically 8 scalar processors in NVIDIA architectures. From the viewpoint of parallel codes, the first important concept to consider concerns the kernel functions (analogous to the shaders in the graphic computing context). When a kernel is called, throws a vector of N threads, each of which ends up being executed on a processor. In turn, every thread executes the instructions found in the kernel function sequentially. Once the kernel function is called, the N threads perform the instructions in parallel. Threads are grouped into blocks of threads. The ones associated in a specific block are executed in a common multiprocessor (8 single processors) where they can share variables and make use of the shared memory space associated with each multiprocessor.

When a kernel is called, blocks of threads are listed and distributed in the multiprocessors available. The threads of a block are executed simultaneously on a single multiprocessor, while multiple blocks could be executed concurrently in one multiprocessor. When all the threads of a block have finished, new blocks are launched in the vacant multiprocessors.

One multiprocessor can execute hundreds of threads concurrently. To manage this large number of threads, it uses a special architecture called SIMT (Single Instruction, Multiple Thread). This amount of threads are grouped into 32 unit packs called warps. In the SIMT architecture, the threads of a warp execute the same kernel instruction at the same time. Each multiprocessor is constantly alternating between warps, which can

decrease the latencies associated with the threads memory access and minimize the time when processors are inactive.

In order to make a parallel implementation, it is crucial to understand the GPU memory hierarchy to manage different memory spaces in order to achieve maximum transfer rates and avoid bottlenecks.

- **Registers**: Is a set of quick access memory banks used by the multiprocessors for local data allocation.
- **Shared memory**: Each multiprocessor has a shared reading and writing memory space only visible by the processors. This type of memory could be as quick as the registers, but the speed strongly depends on the way the code prevents access conflicts between processors.
- **Global memory**: is the read/write memory space visible for all GPU processors. This memory is relatively slow compared to reports the former ones. As the shared memory type, the amount of flow data depends on the way threads access to it.
- **Texture memory**: is a read only memory and can improve noticeably the performance if used properly.

2.2 High performance tips.

In [1] a series of essential strategies to obtain a maximum GPU performance are shown. Although following these guidelines is desirable, the implementation on the some of them depends on the nature of algorithm to parallelize.

- When consecutive half-warp threads read or write consecutive elements of a 32, 64 or 128 bytes memory segment of an array, the memory transfer of such items is done on a transaction, reaching huge memory transfer rates. Coalescent access to memory may occur either in the global memory or in the shared memory.
- Using shared memory to store those data that can be used by multiple threads of a block instead of global memory could achieve transfer rates 100 to 150 times higher.
- Minimize differences between threads of a warp is essential, because the SIMT architecture requires that warp threads are implicitly synchronized.
- Each multiprocessor is designed to manage hundreds of thread blocks and hundreds of thousands of threads simultaneously. As the number of processors is limited to 8, each multiprocessor is constantly alternating services among warps. When the occupation is very high, processors should not inactive waiting some threads to complete memory transactions, and they should give a temporary control to other threads from other warps that are ready to execute instructions. This decreases the latency associated to memory transfer efficiently.

3 SPH Parallel codes.

3.1 Introduction.

Smoothed particle hydrodynamics (SPH) is a Lagrangian method, with no computational mesh and has been widely employed to study free-surface flows. Originally invented for astrophysics during 1970s [7, 8], it has been applied to many different fields of the fluid dynamics and solid mechanics during the last decades. Instead of using a mesh, the SPH methods use a set of interpolation nodes placed arbitrarily within the fluid. This gives several advantages in comparison to mesh-based methods when simulating nonlinear flow phenomena. The method uses discrete approximations to interpolation integrals to change partial differential equations of fluid dynamics into summations. More complete reviews on standard SPH can be found at [9]. The SPH method is capable of dealing with free-surface problems, deformable boundaries, moving interfaces, wave propagation and solid simulation [2, 3]. In contrast, traditional numerical methods have difficulties to compute large deformations on the computational domain and also require special treatment (VOF, Level Set, etc...) to deal with free surface flows. Due to the large number of interactions for each particle at each time step, when SPH codes are computed on a single CPU they take a large computational time. When millions of particles are necessary to compute accurately a physical process, only parallel computing can guarantee efficient computational times. Graphics Processing Units (GPUs) thanks to their multi-threading capability can treat large data flows efficiently. Due to the inexorable development of the market of video games and multimedia, their computing power with streaming multiprocessor technology has increased much faster than of CPUs.Thus, GPUs appear as an accessible alternative to accelerate SPH models using a powerful parallel programming model where the graphics cards are used as the execution devices. Their performance can be compared with large cluster machines. A huge advantage is the price and the easy maintenance GPUs need in comparison with large multi-core systems.

The capability of GPUs to handle with SPH was shown by the pioneer work of Harada [4], where they were able to simulate 60,000 particles in real time and they obtained runs over 28 times faster using a GPU than a CPU with tests of 260,000 particles. The method proposed was implemented using a GeForce 8800GTX GPU and developed before the appearance of CUDA, which means that Haradas work supposed a significant advance even when most of its limitations could be fixed using the advanced GPU programming features introduced by CUDA. Recent works concerning SPH computing on GPU can be found in [5].

3.2 Formulation.

In the SPH technique, the fluid domain is represented by a set of particles scattered in space where different physical properties are known (mass, density, pressure, position, velocity). These mesh-free particles move according to the governing fluid dynamics (Navier-Stokes) equations. The properties can change throughout a simulation due to the

interaction of neighboring points. Herein, the SPH formulation implemented on the GPU architecture is described briefly.

3.3 Interpolation.

As mentioned in the introduction, our goal is to simulate newtonian and incompressible viscous flow in laminar regime. These flows are well described in the continuous formulation by the Navier-Stokes equations:

$$\nabla \cdot \mathbf{v} = \mathbf{0} \tag{1}$$

$$\rho \frac{d\,\mathbf{v}}{dt} = -\nabla P + \mu \nabla^2 \mathbf{v} \tag{2}$$

in which P is the pressure, ρ the density, $\mathbf{v}$ the velocity and t stands for time.

Equation (1) and the pressure term in equation (2) play a combined role. The pressure acts as a Lagrange multiplier that produces a zero divergence velocity field. If the incompressibility condition is imposed, either a penalty formulation or a pressure Poisson equation must be solved to calculate the pressure values consequently increasing the computational cost substantially. In the WCSPH context this hypothesis is relaxed by assuming a weakly compressible fluid with a large sound speed, where the equation (1) is replaced by

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \mathbf{v} \tag{3}$$

and a stiff equation of state $P = P(\rho)$ is added to the system.

Finally, assuming the Lagrangian description of the fluid, the fluid particles move according to the kinematic law:

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} \tag{4}$$

where $\mathbf{r}$ is the position vector.

The interpolation method in SPH is based on the following integral:

$$< f(\mathbf{r}) >= \int_V f(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h) d^3\mathbf{r}' \tag{5}$$

where the integral extends over the domain V, $d^3\mathbf{r}'$ is the element of volume dependant on the dimensions of the problem, $\mathbf{r}$ and $\mathbf{r}'$ are position vector and $W(\mathbf{r} - \mathbf{r}', h)$ is the interpolation or kernel function. The parameter h determines the size of the kernel, which means the distance of influence around $\mathbf{r}'$. The integrals are replaced numerically by summations of the contributions of nearby particles.

The complete SPH formulation [9] considered will be the following:

$$\frac{d\rho_a}{dt} = \sum_{b\in\mathcal{N}_a} m_b\, \mathbf{v}_{ab}\, \nabla_a W_{ab} \tag{6}$$

$$\frac{d\,\mathbf{v}_a}{dt} = -\sum_{b\in\mathcal{N}_a} m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2}\right) \nabla_a W_{ab} + \mathbf{\Pi}_a \tag{7}$$

$$P = \frac{\rho_0\, c_s^2}{\gamma}\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right) \tag{8}$$

in which m is the mass, $\mathbf{r}$ is the position vector, and the subscripts a or b refers to the particle that carries over the considered property.

The notation $\mathbf{v}_{ab}$ means $\mathbf{v}_{ab} = \mathbf{v}_a - \mathbf{v}_b$, $\nabla_a W_{ab}$ is the gradient of the b-centered kernel with respect to the coordinates of particle a and $\mathbf{\Pi}_a$ is the viscous term at particle a [9], $\mathcal{N}_a$ is the index set of particle a neighbors, regarding the kernel support, ρ_0 the reference density, c_s is the numerical sound speed and $\gamma = 7$.

The kernel used is a normalized Gaussian kernel, see [14], with a support of $3h$ and $h = 1.33\,dx$ where h is the smoothing length and dx is the typical initial separation among particles.

The integration in time has been performed using a Leap-frog second order scheme[12]. The selection of the time step has been based on the viscous diffusion, convective, acceleration, and sound waves propagation terms[12]. The CFL factor used was 1/8 using h as a reference length. Depending on the case a special initialization or stabilization technique has been used.

Within SPH techniques, WCSPH is the usual way of modeling incompressible free surface flows [10, 15]. It is easy to programme because the pressure is obtained from a separate equation of state (8) that is chosen so that the speed of sound is large enough to keep the relative density fluctuations small [9]. However, when dealing with highly distorted flows the need for an explicit definition of a boundary at the free surface is a major drawback.

4 THE DAM-BREAK PROBLEM

We implemented the weakly compressible SPH method to simulate a 2D dam break problem with the geometric parameters shown in the figure 2. This geometric configuration is based on [6].

The walls of the tank are three particles thick and are modeled with boundary static particles, treated the same way as the fluid particles in the computation of the forces. At $t = 0$ the fluid is completely still. The physical and numerical parameters were the following: $h_{fac} = 1.33$, smoothing length $h = 2.69 \cdot 10^{-2}$, initial distance between particles $dx = h/h_{fac}$, reference density $\rho_{ref} = 1000.0$, $\gamma = 7.0$, numerical sound speed $c_s = 15.0$, $\alpha = 8$, dynamic viscosity $\mu = 0.744$, gravity $g = 9.81$ and Gaussian kernel type were used

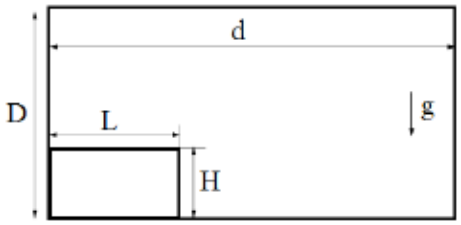

Figure 2: Initial configuration for the fluid and the tank. $L/H = 2$, $D/H = 3$, $d/H = 5.366$

N particles	CPU	GPU		
		k=1	*k=5*	*k=10*
7547	26.97	1.67	0.98	0.89
15047	57.19	3.10	1.78	1.61
25057	98.09	5.58	3.22	2.95
52559	227.75	10.55	6.07	5.51
90079	400.69	18.32	10.40	9.65
112550	494.37	23.95	13.60	12.38
137584	596.70	31.23	18.87	17.51

Table 1: CPU and GPU average time per step in milliseconds. Sequential executions were done on a CPU Intel T8200 - 2.33GHz and parallel executions were done on one device of a GPU NVIDIA Tesla C1060.

in the simulation. For each particle i, the initial pressure has an hydrostatic distribution $p_i = \rho_{ref} \cdot gravf \cdot (H - y_i)$ and the initial density is calculated according to equation (8).

4.1 CPU vs. GPU implementations

The table 1 shows the computational timings of seven executions corresponding to different values of the scale parameter H and keeping constant the distance between particles dx. The smallest value of the scale parameter was $H = 0.6$. Due to the relatively high computational cost of the neighbor list construction stage in the parallel implementation, different GPU executions were done varying the frequency of the execution of this stage, so that the neighbor list construction is performed every k steps of the algorithm in the GPU implementation.

In spite of the CPU sequential algorithm computes the neighbor list construction every step, this stage represents approximately 1% of the total computational time of the whole time step, consequently if the neighbor list construction were performed every k steps the speed up factor would not be noticeably reduced in this implementation.

A maximum speed-up of $x41.5$ was obtained when 90079 particles were simulated. It should be noticed that the GPU implementation lacks of certain scalability when the algorithm deals with more than this amount of particles, possibly because in this GPU approach the CUDA kernels involved in the neighbor list construction stage are memory bounded.

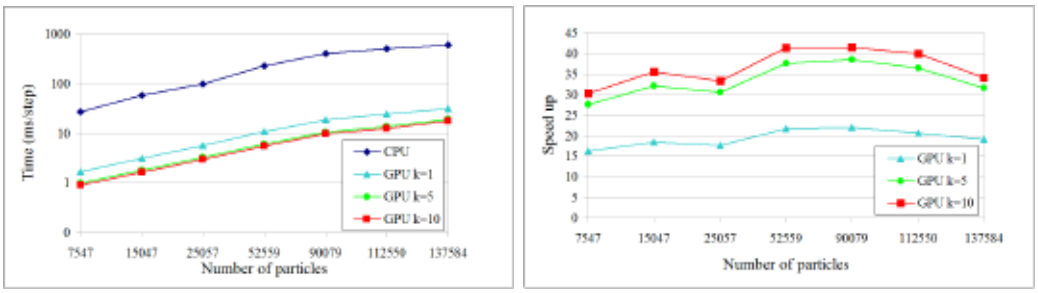

Figure 3: Average time and speed-up plots.

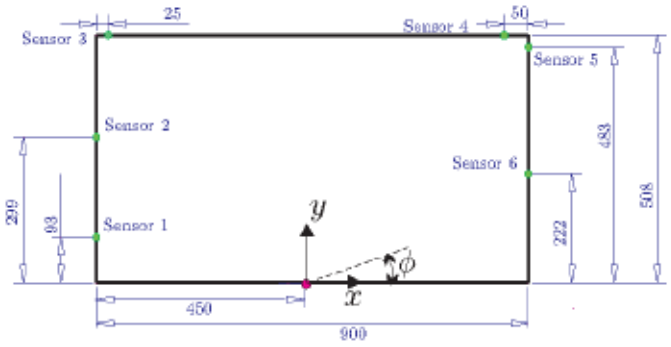

Figure 4: Tank geometry and position of the sensors. Units in mm. Rotation center at mid bottom.

5 SLOSHING FLUID CONTAINED IN LNG VESSEL TANKS

In order to provide a practical application case, the sloshing in a rectangular LNG vessel tank has been simulated. Sloshing moment amplitudes in a rectangular tank for a wide range of rolling frequencies have been investigated experimentally and numerically, see [11] or [12].

5.1 Problem description

The experimental data were measured in a tank (62 mm thickness) similar to the one shown in Figure 4, where only a lateral water impact has been considered.

Fixed particles (also refereed as dummy particles) have been used to impose the boundary condition. This type of boundary condition has a simple implementation in a massive parallel oriented code, but the consequences of this solution could be improved if more sophisticated boundary conditions are implemented.

The physical time simulated was 3 seconds (First impact time scale) and the number of particles N was around 100000. The computational time using a NVIDIA GeForce GTX 295 was less than 2 hours. This device has 480 cores, but excepting Multi-GPU implementation, only 240 cores can be used simultaneously.

To take into account the action of the engine that causes the tank movement, which

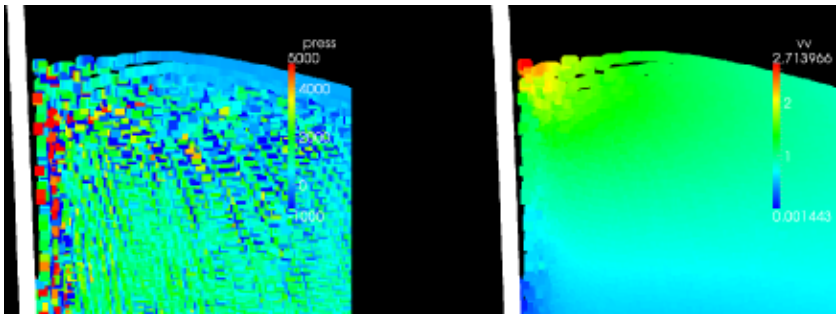

Figure 5: Wave impact frame, pressure field at left, velocity field at right.

is not a pure sinusoidal signal, and include all this effects in the simulation, the moving parameters of the tank (angle, angular velocity and angular acceleration) were directly obtained from the benchmark. This methodology tries to include the initializing phase caused by engine start in the fluid simulation.

5.2 Results

In figure 5, the pressure(left) and velocity (right) fields are represented in a zoomed area where the pressure sensor measures the first wave impact. Although the velocity field is nicely smoothed, the pressure field still has the typical noise effect present in weakly compressible SPH simulations.

Looking at figure 5 two interesting effects can be observed, firstly, the distance between particles is not kept constant and density instabilities are present at the free surface, secondly the particles are unable to reach the wall boundary and an undesired gap appears between the wall and the closest particles.

In Figure 4 the complete distribution of sensors in the tank is shown. In this work only the sensor 1 has been used for the validation.

In order to improve the pressure measurement, Shepard correction [13] has been implemented to improve the consistency of the interpolation, getting at least a 0th order consistency.

During the simulation, two different parts can be distinguished, in the first one the sensor 1 records the result of a nearly hydrostatic pressure distribution, in the second part the violent impact is registered and the flow dynamics is extremely complex. Comparing the results obtained, see Figure 6, when fixed particles are taken into account in sensor pressure interpolation the hydrostatic part of the simulation has a good agreement with the experimental data. This fact contrasts with the second part where the impact pressure measurement is better represented without fixed particles in the interpolation. When no fixed particles are taken into account for the pressure interpolation, the error in the maximum pressure impact value is around 870 Pa/m (24%). It was also noticed that the maximum pressure impact simulated was delayed 0.06 seconds compared to the experimental wave impact.

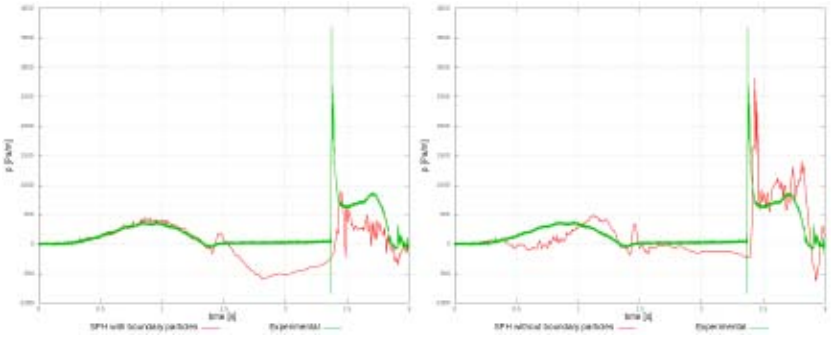

Figure 6: Pressure comparison, using fix particles (left), without fix particles (right).

6 CONCLUSIONS

A CPU-GPU solver has been developed to deal with 2D free surface flow problems requiring high computational cost. This code can be run as either a CPU code or a GPU code depending on the availability of hardware. The model has been demonstrated to be both accurate and efficient when dealing with a gravity-dominated flow problem. The code was tested in two different problems: first, the classical dam-break problem and also, the wave impact of the sloshing fluid contained in LNG vessel tanks as a practical example of particle methods. Simulations carried out for different resolutions showed a close agreement between numerical and experimental results. In addition, the numerical results were observed to converge to the experimental ones when increasing the resolution (the number of particles), both for free-surface elevations and pressures. In terms of efficiency, we have demonstrated that simulations with a large number of particles can be simulated on a personal computer equipped with a CUDA-enabled GPU card taking advantage of the performance and memory space provided by the new GPU technology. This means that research can be conducted with available cheap technology for problems that previously required high-performance computing (HPC). The speedups obtained in this work reveal the possibility to study real-life engineering problems at a reasonable computational cost. For the validation case chosen here, the GPU parallel computing can accelerate serial SPH codes by almost two orders of magnitude. Experience has shown that the speedup varies from one test to another with even greater speedup achievable than found here. The achieved performance can be compared to the large cluster machines, which are expensive and hard to maintain. Finally, for simulations requiring several million particles the immediate future for GPU computing should focus upon multi GPU implementations, since the memory requirements are still a limitation for a single GPU.

REFERENCES

[1] NVIDIA Corporation. *NVIDIA CUDA Best Practices Guide. Version 3.0*

[2] Libersky LD, Petschek AG, Carny TC, Hipp JR and 623 Allahdady FA (1993) *High-*

strain Lagrangian hydrodynamics a three-dimensional (SPH) code for dynamic material response. J Comput Phys. **109**: 67-75.

[3] Randles PW, Libersky LD (1996) *Smoothed particle hydrodynamics: Some recent improvements and applications. Comput Meth Appl Mech Engrg* **138**: 375-408.

[4] Harada T. and Koshizuka S. and Kawaguchi Y. *Smotthed Particle Hydrodynamics on GPUs. Proc of Comp Graph Inter* (2007), 63–70.

[5] Herault A. and Bilotta G. and Dalrymple R.A. (2010) *SPH on GPU with CUDA. Journal of Hydraulic Research*, **48 Extra Issue**: 74-79, doi:10.3826/jhr.2010.0005

[6] Colagrossi A. and Landrini M.. *Numerical simulation of interfacial flows by smoothed particle hydrodynamics. Journal of Computational Physics*, (2003), pp. 454.

[7] Lucy, L.B. . *A numerical approach to the testing of the fission hypothesis. Astron. J.* (1977) **82(12)**: 1013–1024.

[8] Gingold RA and Monaghan JJ (1977) *Smoothed particle hydrodynamics: theory and application to non- spherical stars. Mon Not R Astr Soc* **181**: 375–389.

[9] Monaghan, JJ. (2005) *Smoothed Particle Hydrodynamics. Rep Prog Phys* **68**: 1703–1759.

[10] Monaghan, JJ. (1994) *Simulating free surface flows with SPH. Journal of Computational Physics* **110(2)**: 399-406.

[11] Souto-Iglesias A, Pérez-Rojas L, Zamora-Rodríguez R. *Simulation of anti-roll tanks and sloshing type problems with smoothed particle hydrodynamics. Ocean Engineering* (2004)**31**::1169.

[12] Souto-Iglesias A, Delorme L, Pérez Rojas L, Abril S. *Liquid moment amplitude assessment in sloshing type problems with SPH. Ocean Engineering* (2006)**33**::1112.

[13] Xu F, Kikuchi M. *The Shepard Correction of Kernel Function in SPH Method. Conference on Computational Engineering and Science* (2004)**3**::123.

[14] Molteni D. and Colagrossi A. *A simple procedure to improve the pressure evaluation in hydrodynamic context using the SPH. Computer Physics Communications* (2009) **180**:861872.

[15] Lee E.S. and Moulinec C. and Xu R. and Violeau D. and Laurence D. and Stansby P. *Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method. Journal of Computational Physics* (2008) **227(18)**:8417-8436.