Improvement of Fluid Simulation Runtime of Smoothed Particle Hydrodynamics by Using Graphics Processing Unit (GPU)

Article *in* Journal of ICT Research and Applications · December 2017

DOI: 10.5614/itbj.ict.res.appl.2017.11.3.2

CITATIONS
0

READS
219

3 authors:

Wahyu Srigutomo
Bandung Institute of Technology
128 PUBLICATIONS **190** CITATIONS

SEE PROFILE

Ruddy Kurnia
University of Twente
13 PUBLICATIONS **61** CITATIONS

SEE PROFILE

Suprijadi Suprijadi
Bandung Institute of Technology
101 PUBLICATIONS **243** CITATIONS

SEE PROFILE

Improvement of Fluid Simulation Runtime of Smoothed Particle Hydrodynamics by Using Graphics Processing Unit (GPU)

Wahyu Srigutomo[1]*, Ruddy Kurnia[1] & Suprijadi[2]

[1]Physics of Earth and Complex System, Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Jalan Ganesa No. 10, Bandung 40132, Indonesia
[2]Theoretical Physics and Instrumentation, Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Jalan Ganesa No. 10, Bandung 40132, Indonesia
*E-mail: wahyu@fi.itb.ac.id

Abstract. This study concerns an implementation of smoothed particle hydrodynamics (SPH) fluid simulation on a graphics processing unit (GPU) using the Compute Unified Device Architecture's (CUDA) parallel programming. A bookkeeping method for the neighbor search algorithm was incorporated to accelerate calculations. Based on sequence code profiling of the SPH method, particle interaction computation – which comprises the calculation of the continuity equation and the momentum conservation equation – consumes 95.2% of the calculation time. In this paper, an improvement of the calculation is proposed by calculating the particle interaction part on the GPU and by using a bookkeeping algorithm to restrict the calculation only to contributed particles. Three aspects are addressed in this paper: firstly, speed-up of the CUDA parallel programming computation as a function of the number of particles used in the simulation; secondly, the influence of double precision and single precision schemes on the computational acceleration; and thirdly, calculation accuracy with respect to the number of particles. Scott Russell's wave generator was implemented for a 2D case and a 3D dam-break. The results show that the proposed method was succesfull in accelerating the SPH simulation on the GPU.

Keywords: *Compute Unified Device Architecture (CUDA); fluid simulation runtime; graphic processing unit (GPU); Scott Russell's wave generator; smoothed particle hydrodynamics (SPH).*

1 Introduction

The smoothed particle hydrodynamics (SPH) method was introduced as a Lagrangian formulation in which the modeling does not depend on a mesh geometry but particles move following the motion of the flow. Grid-based methods such as finite differences and finite elements have difficulties in many aspects, limiting their application in a wide range of problems. Large deformations, inhomogeneities, moving material interfaces, deformable boundaries and free surfaces are difficult to simulate using grid-based methods.

Received January 9th, 2017, Revised March 5th, 2017, Accepted for publication July, 31st, 2017.
 ISSN: 2337-5787, DOI: 10.5614/itbj.ict.res.appl.2017.11.3.2

Particle-based methods can deal with these difficulties. The main idea behind the SPH method is to provide accurate and stable numerical solutions for solving integral equations or PDEs with all kinds of possible boundary conditions that are discretized using a set of arbitrarily distributed particles without using a mesh. However, a weakness of the method is numerical dissipation, which leads to energy loss. This problem has been overcome by introducing small and weak perturbations on each particle stochastically as proposed in [1].

This work concerns optimizing the code that was developed in [1] to reduce computation time and be able to simulate with a large number of particles to increase the accuracy of the simulation. Since runtime simulation is proportional to the total number of particles, it is difficult to simulate fluid flows using SPH without an additional strategy to speed up calculation. Monaghan and Gingold [2] proposed a bookkeeping method to save computational time. This method can accelerate the simulation process by restricting the calculation of particle interaction to a box containing the concerned particle and neighboring boxes containing the corresponding neighboring particles. Nevertheless, the bookkeeping method remains inadequate for enabling fast simulation with higher particle resolutions. A possible technique to increase the computational speed is parallel computation. Parallel computation on graphics processing units (GPU) is extensively used for scientific computing. The GPU is specialized for performing intensive and highly parallel computation as well as interactive visualization and rendering of SPH [3].

The objective of this paper is to discuss the acceleration of the SPH method for fluid simulation by using parallel GPU programming. Based on profiling a sequential code in SPH – i.e. the particle interaction part, which consists of calculating the continuity equations and the momentum conservation equation – consumes about 95.2% of total calculation time. Since runtime simulation is proportional to the total number of particles, the particle interaction part is computed on the GPU using CUDA (Compute Unified Device Architecture) NVDIA® parallel programming. The bookkeeping algorithm is implemented with a grid construction and box calculation, which are calculated on the host or CPU, after which all the data are copied to the GPU. All particle interaction calculations are then performed on the GPU only for neighboring particles in the neighbor boxes and those in the box of the concerned particle. In a previous work, executing the SPH physics on a GPU accelerated the simulation in comparison to CPU implementation [4]. In the present work, the method is implemented in such a way that neighbor search is computed on the CPU while the standard SPH physics computation is done on the GPU. The speed-up of computation for two different GPU cards (GTX 560Ti and GT 220) was

compared, each with different computation capability and precision. In addition, the bookkeeping algorithm was used for the neighbor search step.

In this paper, first, the general concept of the SPH method and its standard implementation are briefly discussed. A CUDA parallel approach for SPH is subsequently introduced and described in more detail. Afterwards, the Scott Russell wave generator problem is used to compare runtime simulations and speed-up of the two different GPU cards for single precision and double precision simulation, respectively. Finally, we observe that the accuracy of simulation depends on the total number of particles. Full-time simulation for a 2D Scott Russell wave generator problem and a 3D dam-break problem with substantial numbers of particles are shown as test cases.

2 Fundamental Theory

2.1 Basic Theory of SPH

SPH is an interpolation method that allows any function to be expressed in terms of its values at a set of disordered points [5]. It represents a continuous fluid using a set of particles. Each *i*-th particle carries information of physical quantities such as position $\mathbf{r}_i$, velocity $\mathbf{u}_i$, mass m_i, density ρ_i and pressure P_i. Generally, the value of a function representing a physical quantity can be approximated by a number of neighboring particles, each contribution of which is determined by a kernel function W, written in Eq. (1)as follows:

$$f(r) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(r_j) W(r - r_j, h) \tag{1}$$

and the gradient discretization of function f in Eq. (2) becomes:

$$\nabla f(r) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(r_j) \nabla W(r - r_j, h) \tag{2}$$

where h is the smoothing length that determines the support of the kernel function.

The kernel function W is usually chosen to be an even function of finite range and it should be normalized to unity when integrated over space. The neighboring particles within a support domain of radius κh of the kernel function for the *i*-th particle contribute to the summation process of the *i*-th particle, as displayed in Figure 1.

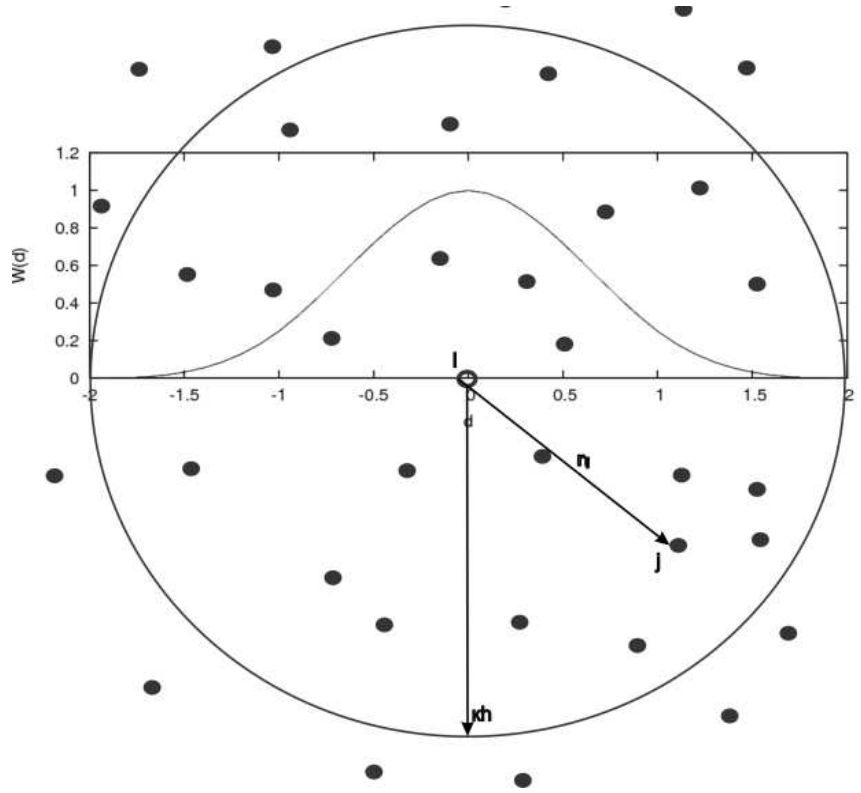

Figure 1 Sphere of influence for the *i*-th particle in 2D space.

The following kernel function in Eq. (3) based on cubic spline functions is used [5]:

$$W(d)=\begin{cases}\left(1-1.5d^2+0.75d^3\right)/\beta, & \text{if } 0\le d<1;\\ 0.25\left(2-d\right)^3/\beta, & \text{if } 1\le d<2;\\ 0, & \text{if } 2\le d\end{cases} \tag{3}$$

where $d=\left|r_i-r_j\right|/h$, $\beta=0.7\pi h^2$ and $\kappa = 2$.

2.2 Governing Equations

The governing equations for a viscous fluid that comprises the mass continuity and momentum conservation equations are in Eqs. (4) and (5) as follows:

$$\frac{1}{\rho}\frac{\partial \rho}{\partial t} + \nabla.\mathbf{u} = 0, \tag{4}$$

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla P + \mathbf{g} + \Theta \tag{5}$$

where ρ is density, $\mathbf{u}$ is velocity vector, P is pressure, $\mathbf{g}$ is gravity acceleration, and Θ refers to the diffusion terms.

In SPH, the fluid field is represented as a set of particles interacting with each other through evolution equations. The discretization of the continuity and momentum conservation equations in terms of Eq. (1) gives the following Eqs. (6) and (7) [6]:

$$\frac{d\rho_i}{dt} = \rho_i \sum_j \left(\mathbf{u}_i - \mathbf{u}_j\right).\nabla W_{ij}\frac{m_j}{\rho_j} \tag{6}$$

$$\frac{d\mathbf{u}_i}{dt} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_j \rho_i} + \Pi_{ij}\right)\nabla W_{ij} + \mathbf{g} \tag{7}$$

where ρ_k, P_k, and $\mathbf{u}_k$ are density, pressure and velocity of particle k (evaluated at $k = i$ or $k = j$), respectively, m_j is the mass of the j-th particle, $\mathbf{g}$ is body force, in this case the gravity, and $W_{ij} = W(\mathbf{r}_i\text{-}\mathbf{r}_j, h)$. Π_{ij} in Eqs. (8) and (9) is an artificial viscous term aimed at increasing the stability of the numerical algorithm [2]:

$$\Pi_{ij} = \begin{cases} -\tau\mu_{ij}\dfrac{c_{s,j} + c_{s,i}}{\rho_j + \rho_i}, & \text{if } \left(\mathbf{u}_i - \mathbf{u}_j\right).\left(\mathbf{r}_i - \mathbf{r}_j\right) < 0; \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$\mu_{ij} = h\frac{\left(\mathbf{u}_i - \mathbf{u}_j\right).\left(\mathbf{r}_i - \mathbf{r}_j\right)}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^2 + \varepsilon h^2}, \tag{9}$$

where c_s is the speed of sound following the equation of state, $\varepsilon = 0.01$ and τ is an empirical parameter that represents the shear viscosity. The value of $\tau = 0.01$ is used for most computations [5].

2.3 Equation of State

In the present implementation, the equation of state is used to avoid expensive resolution of the Poisson equation. The equation of state relates pressure to density as shown in Eq. (10):

$$P(\rho) = P_0\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right) + \chi \tag{10}$$

and the corresponding speed of sound is in Eq. (11) as follows:

$$c = \sqrt{\frac{P_0\gamma}{P_0}} \tag{11}$$

where γ is the polytrophic constant, typical adopted value is $\gamma = 7$ for water and similar fluids, c is the speed of sound at reference density $\rho_0 = 1000$ kg/m^3, χ is the background pressure; the reference pressure P_0 is usually chosen to achieve a weakly compressible fluid. To approximate real fluid with an artificial compressible fluid, the actual speed of sound cannot be used. However, the speed of sound should be set adequately low so that the time stepping remains reasonable. Monaghan [7] has found by experience that the speed of sound can be artificially slowed significantly for fluids without affecting the fluid motion and that the minimum speed of sound should be about ten times faster than the maximum fluid velocity in order to maintain the changes in fluid density at less than 1%. The optimal value of the speed of sound is determined by considering the balance of time stepping and the incompressible behavior of the artificial compressible fluid. This artificial compressibility considers that every theoretically incompressible fluid is actually compressible.

2.4 Boundary Conditions

The simulation domain is surrounded by fixed boundaries, or, in the case of water waves, a free surface. For fixed surfaces, the fluid particles next to the surface cannot pass through the surface, which is specified by a no-flow condition: $\mathbf{u}.\mathbf{n} = 0$ where $\mathbf{n}$ is the normal vector. In this work, the calculations were made with boundaries defined by lines of particles that exert repulsive forces on the fluid particles [8].

2.5 Bookkeeping Method

The conventional SPH method is highly time-consuming because the particle approximation is calculated for all particles. To accelerate the runtime of SPH, calculating only the nearest neighboring particles for the concerned particle should be considered. The smoothing function of SPH has a compact support domain, thus only a finite number of particles within the support domain of dimension κh of the concerned particle are used in the particle approximations. The process of finding the nearest particles is commonly referred to as nearest neighboring particle searching (NNPS). According to Monaghan and Gingold

[2], a substantial saving in computational time can be achieved by using cells as a bookkeeping device. Monaghan [9] describes the procedure for carrying out nearest neighboring particle searching using the linked-list algorithm.

In the implementation of the linked-list algorithm, a temporary mesh is overlaid on the problem domain, as illustrated in Figure 2. The mesh spacing is selected to be the same as the dimension of the support domain. Then, for the *i*-th particle its nearest neighboring particles can only be in the same grid cell or the immediately adjoining cells. Therefore, the search is confined to 3, 9 or 27 cells for one-, two-, or three-dimensional space, respectively. The linked-list algorithm allows each particle to be assigned to a cell and for all the particles in a cell to be chained together to ensure easy access.

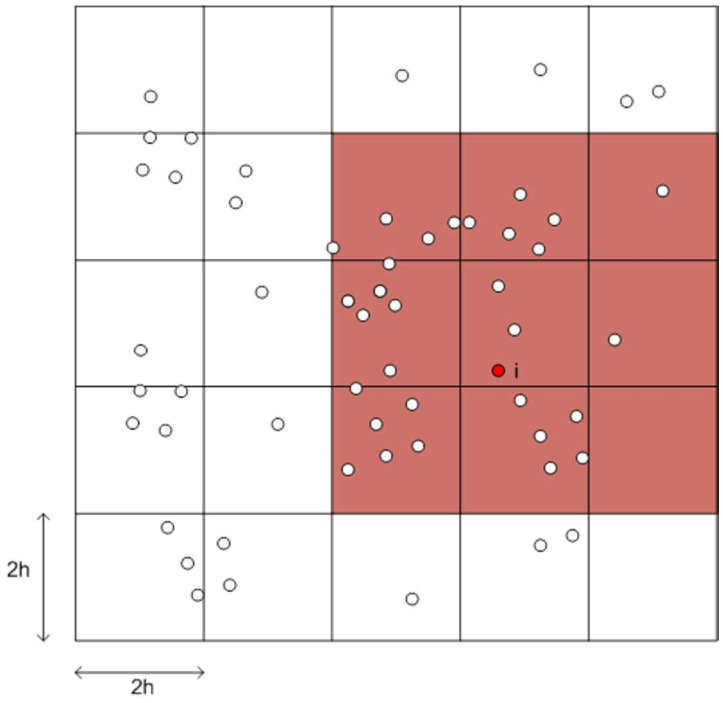

Figure 2 Illustration of the bookkeeping method.

The following pseudo-code computes *NB(K,J)*, the number of particle in the *(K,J)-th* bin, and *LIST(1 ,… ,NB(K, J),K,J)*, a list by index number of the particles in the *(K,J)-th* bin. For typical kernels $L = 2h$, where h is the 'interactions' parameter.

For 2D cases, the grid construction and bookkeeping algorithms are:

Algorithm 1: Grid construction algorithm

```
DO I=1,N
    K=Y(I)/L
    J=X(I)/L
    NB(K,J)=NB(K,J)+1
    LIST (NB(K,J),(K,J)=I
ENDDO
```

Algorithm 2: Bookkeeping algorithm

```
DOI=1,N
K=Y(I)/L
J=X(I)/L
DO JNDEX=J-1,J+1
DO KNDEX=K-1,K+1
DO INDEX=1,NB(KNDEX,JNDEX)
M=LIST(INDEX,KNDEX,JNDEX)
          W(I,M)=EVALUATION OF W
          ENDDO
ENDDO
ENDDO
ENDDO
```

In 2D cases, a particle in the *(K,J)-th* bin can only interact with particles in a total of 9 bins, while in 3D cases in a total of 27 bins.

3 CUDA Parallel Approach for SPH

In this section, CUDA parallel programming for fluid simulation of smoothed particle hydrodynamics is described, covering the profiling of the SPH code sequence steps, implementation of the bookkeeping method on the GPU, and computations of density and force.

3.1 Profiling the SPH Code Sequence

The main part of our SPH code is the time stepping function, which consists of the right-hand side (RHS) of the Navier-Stokes equations, grid calculation for neighbor particle search, position updating, equation of state calculation, and stepping time calculation based on the CFL condition. The second order of the Runge-Kutta time-stepping scheme is used.

Code profiling was conducted for 1000 iterations, as shown in Figure 3. The result shows that the most time-consuming part occurs in the calculation of particle interaction for the continuity equation and the momentum equation. The

computation time of the particle interaction part increases according to the number of particles. This makes it difficult to simulate fluid dynamics using SPH, where the precision of the method depends on the number of particles.

The neighbor search algorithm can accelerate the SPH code by increasing its efficiency owing to the fact that particle interaction calculations are only conducted for the nearest particles of the concerned particle. In addition to using neighbor search, calculation of the particle interaction part is conducted in several blocks on the GPU in a parallel way, as depicted in Figure 4.

The calculations of the continuity equation, the conservation of momentum, and equation of states are conducted on a device with a CUDA scheme. The grid calculation for neighboring particles is constructed and carried out on the host, from which the result is subsequently copied to the device.

3.2 Bookkeeping Method on GPU

Neighbor searching is usually the most computationally expensive step in particle simulation since it is applied to each particle at every step. In our bookkeeping code there are three main calculation parts. For 2D cases, each particle is located in a grid and stored in two 1D arrays. The first and the second 1D array represent the index of small boxes on the *x*-axis and the *y*-axis respectively. The size of the array is the same as the number of particles. The next step is listing the neighbor boxes of the concerned box. The list of neighbor boxes is stored in two 3D arrays, where the first 3D array and the second 3D array represent the index of neighbor boxes on the *x*-axis and the *y*-axis, respectively.

The size of the 3D array is the maximum number of boxes on the *x*-axis multiplied by the maximum number of boxes on the *y*-axis multiplied by the maximum number of neighbor boxes of the concerned box. Finally, the index of neighbor particles is calculated from the particles in each neighbor box. The index of neighbor particles is stored in one 3D array. This calculation scheme requires many arrays to be reserved and therefore has a large computational cost.

Algorithm 1 (grid construction) and Algorithm 2 (bookkeeping) are both implemented on the GPU. All 3D arrays (i_x, i_y, i_z) must be converted to 1D coordinates by

$$i_x + i_y \times grid_{width} + i_z \times grid_{width} \times grid_{height} .$$

Then, after all grids have been constructed and calculation of the boxes of particles in time-stepping has been done on the host, all data are copied to the device.

3.3 Density and Force Computations

The main calculation in the SPH method deals with the continuity equation (density) and the momentum conservation equation (velocity). CUDA parallel programming is implemented in this part to speed up calculation.

Algorithm 3: Parallel density and force calculation algorithm

```
define inttid=threadIdx.x+bloxkDim.x*blockIdx.x;
while (tid<n){
        K=Y(tid)/L
        J=X(tid)/L
        DO JNDEX=J-1,J+1
            DO KNDEX=K-1,K+1
                DO INDEX=1, NB(KNDEX, JNDEX)
                     M=LIST(INDEX, KNDEX, JNDEX)
                   dvdt[tid]=(tid, M) EVALUATION FOR VELOCITY
                     OR
                     drhodt[tid]=(tid, M) EVALUATION FOR VELOCITY
                ENDDO
            ENDDO
       ENDDO
      tid+=blockDim.x*gridDim.x;
}
```

Since our simulation is divided into blocks with a size equal to the global support radius, the neighbors of any particle in a box are particles in the neighboring box or in the box of the concerned particle. On devices, number of particles N is divided into $(N+511)/512$ blocks and 512 threads per block, which are launched in the kernel. In Algorithm 3, *threadIdx, blockDim,* and *blockIdx* are built-in variables in CUDA programming.

4 Results and Discussion

A two-dimensional Scott Russell wave generator problem and a three-dimensional dam-break problem were adopted as test cases. The objective of this work was to speed up the runtime simulation by using GPU-CUDA programming while simultaneously increasing the number of particles to achieve high accuracy in the simulation results. The accuracy of the simulation

was evaluated using an analytical solution of the solitary wave of Scott Russell's wave generator problem.

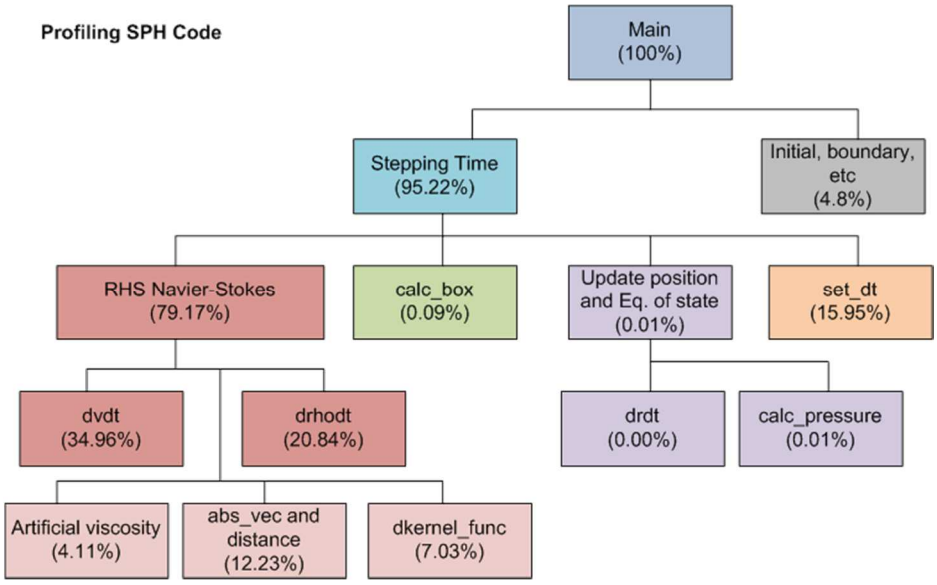

Figure 3 Profiling SPH code.

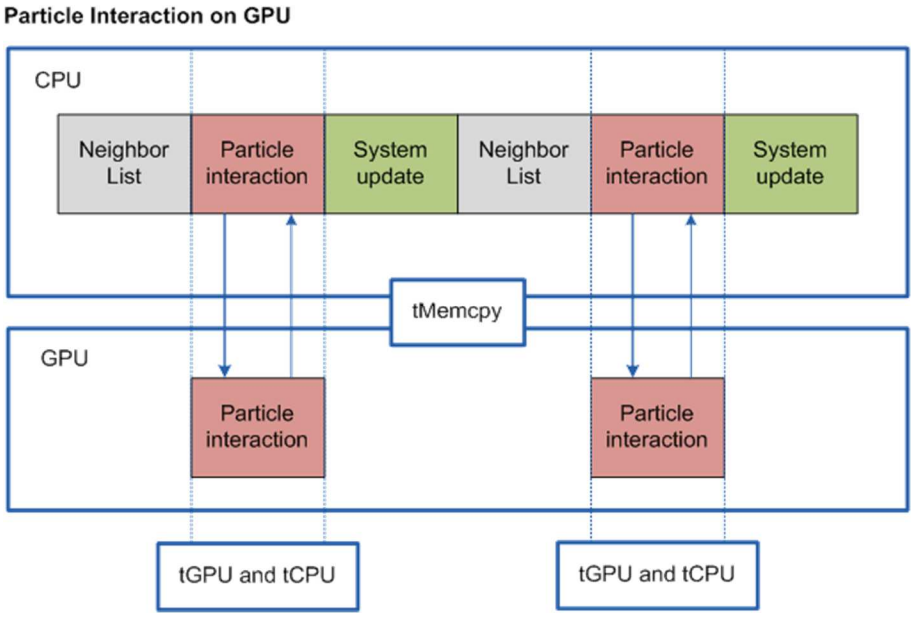

Figure 4 Particle interaction scheme on the GPU.

4.1 Precondition of Simulation

Scott Russell's wave generator was used to simulate a falling avalanche in a dam reservoir. In this case, a solitary wave generated by a heavy box falling vertically into the water was considered. The geometry of tank and weighted box were chosen according to the simulation configuration used by Ashtiani and Rezaei [10]. The simulation involved a weighted box (0.3 m x 0.4 m) dropping vertically into a 0.21 m deep wave tank filled with still water. The bottom of the box is initially placed 0.5 cm below the water surface to avoid splashing. The initial condition of the problem is shown in Figure 5. The vertical velocity of the box is computed using the following Eq. (12) [11]:

$$\frac{V}{\sqrt{gD}} = A\frac{Y}{d}\left(1-\frac{Y}{d}\right)^{1/2} \tag{12}$$

where d is the depth of the water, Y is the height of the bottom of the box above the bottom of the tank at time t, g is the acceleration of gravity, A is a constant value equal to 0.7, and V is the vertical falling velocity of the box at time t.

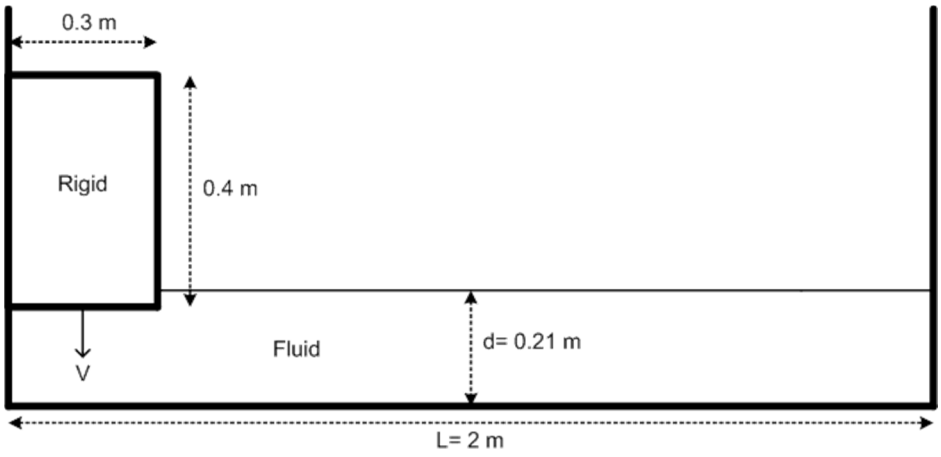

Figure 5 Sketch of Scott Rusell's problem considered in this paper.

The analytical form of the solitary wave generated by the falling box calculated by Lo and Shao [12] is in Eq. (13) as follow:

$$H(x,t) = a \times \sec h^2\left[\sqrt{\frac{3a}{4d^3}}(x-ct)\right] \tag{13}$$

where H is the water surface elevation, a is wave amplitude and $c = \sqrt{g(d+a)}$ is the solitary wave's velocity. The analytic solution was used to validate the simulation. In the simulation, the fluid surface was detected using the fast-free surface detection method proposed by Marrone and Colagrossi [13].

The concept of fast-free surface detection is based on the minimum eigenvalue of the normalization matrix of the kernel function and the scan region algorithm. The first step of the algorithm, which is based on the work of Doring [14], exploits the eigenvalues of the renormalization matrix, which is defined in Eq.(14) as follow:

$$\mathbf{B}(x_i) = \left[\sum_j \nabla W_j(\mathbf{x}_i) \otimes (\mathbf{x}_j - \mathbf{x}_i) \Delta V_j \right]^{-1} \tag{14}$$

where ΔV_j is the volume of j-th particle. Doring [14] showed that the value of the minimum eigenvalue λ of $\mathbf{B}^{-1}$ depends on the spatial organization of the particles in the neighborhood of the considered i-th point calculation. The minimum eigenvalue λ tends to change towards 0 when a particle is going away from the fluid domain and tends to switch to 1 when said particle is located inside the fluid domain. The second step of the algorithm detects particles that actually belong to the free surface by means of their geometric properties and evaluates their local normals.

A quantitative comparison between the simulation and the analytic solution is shown, using the relative error at each time t as expressed by the following Eq. (15):

$$\varepsilon = \frac{\sum_i^N \left| y_i - y_i' \right|}{\sum_i^N |y_i|} \times 100\% \tag{15}$$

where ε is the relative error (%), i is the index of particles detected on the surface, N is the number of particles detected, y_i is the height of the analytical solution at position x_i, and y'_i is the height of the fluid surface detected at position x_i.

SPH simulations were conducted on a GPU for single precision and double precision only up to 1000 iterations, but several full-time simulations were also performed. The speed-up was almost the same for both simulations, as can be seen in Tables 3 and 4, ensuring that all simulation results with 1000 iterations were acceptable. The speed-up value was defined as the ratio between GPU

time computation and CPU time computation. For the single precision simulations, two different computers equipped with different GPU cards and different CPU processors were used. The specifications of the first computer were: NVDIA GTX 560 Ti with intel Core i7 and RAM 16 GB, and for the second one: NVDIA GT 220 with intel Core i5 and RAM 4 GB. However, only the first computer was used for the double precision simulation. The details of the specification are listed in Tables 1 and 2.

Table 1 GPU card specification.

Gpu	Multiprocessor	CUDA Cores	Compute Capability
GTX 560 Ti	8	384	2.1
GT 220	6	48	1.2

Table 2 CPU Specification

Processor	Cores	Thread	Clock speed	Cache
Intel Core i7-2600K	4	8	3.4 GHz	8 M
Intel Core i5-2400	4	4	3.1 GHz	6 M

Table 3 Runtime for full-time simulation on GT 220.

Number of Particles	CPU (hours)	GPU (hours)	Speed up
1465	0.3832	0.1261	3.0394
3300	1.6857	0.5409	3.1165
5873	4.8352	1.5632	3.0931

Table 4 Runtime for 1000-iteration simulation on GT 220.

Number of Particles	CPU (hours)	GPU (hours)	Speed-up
1465	0.0419	0.0141	2.972
3300	0.1129	0.0365	3.0932
5873	0.2165	0.069	3.1376

4.2 Single Precision Simulation

Figures 6 and 7 depict the runtime speed-up for the simulation with single precision SPH, which has only float numbers. The implementation of GPU parallel code for SPH successfully reduced the computation time. The speed-up for the GT 220 was from 2.9 up to 3.2, whereas for the GTX 560Ti it was from 3.7 up to 4.15. Since the GTX 560 Ti has a higher computation capability (that is 2.1) compared to the GT 220 (1.2), the result also shows that the speed-up of the calculation with the GTX 560Ti was higher than with the GT 220.

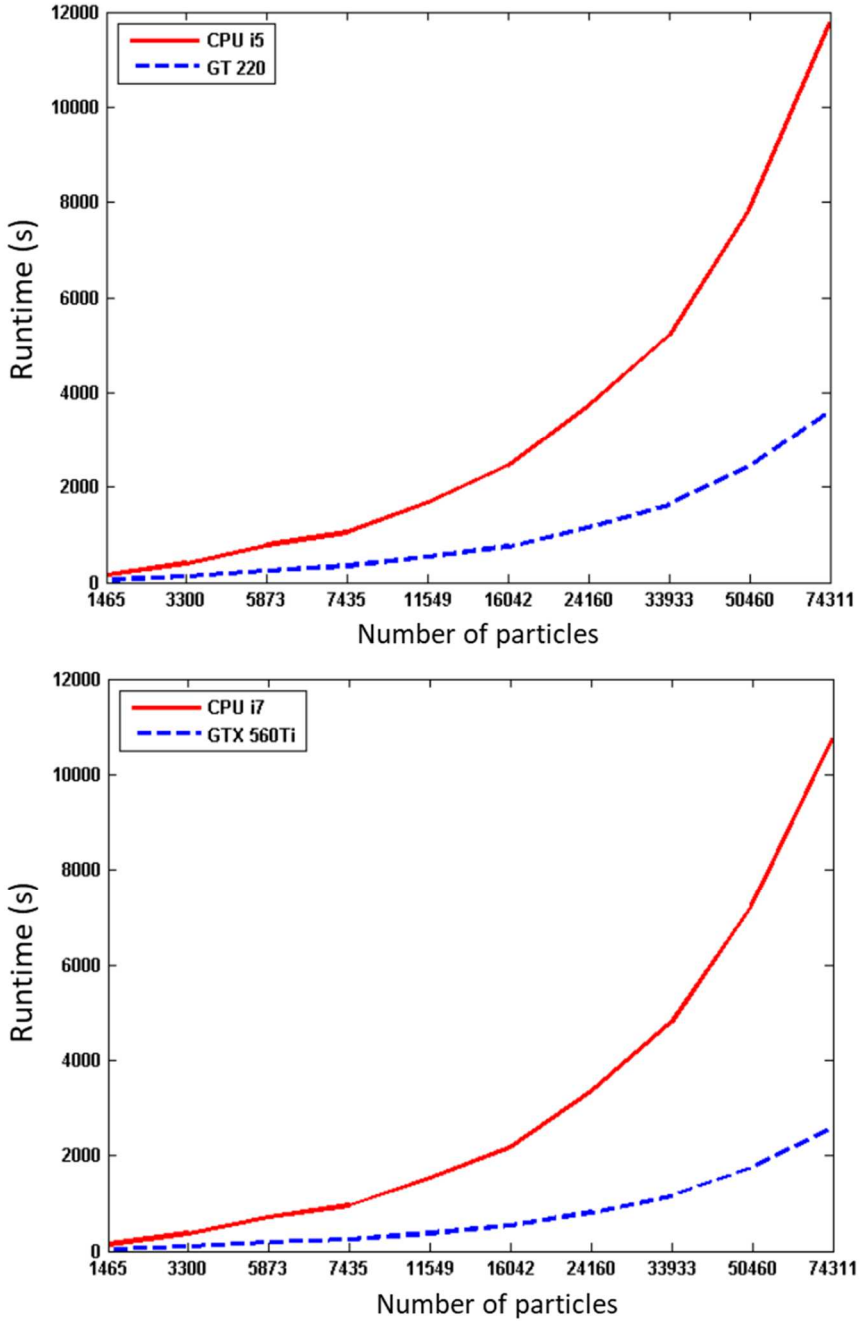

Figure 6 Computation time for single precision SPH using GT 220 and GTX 560 Ti.

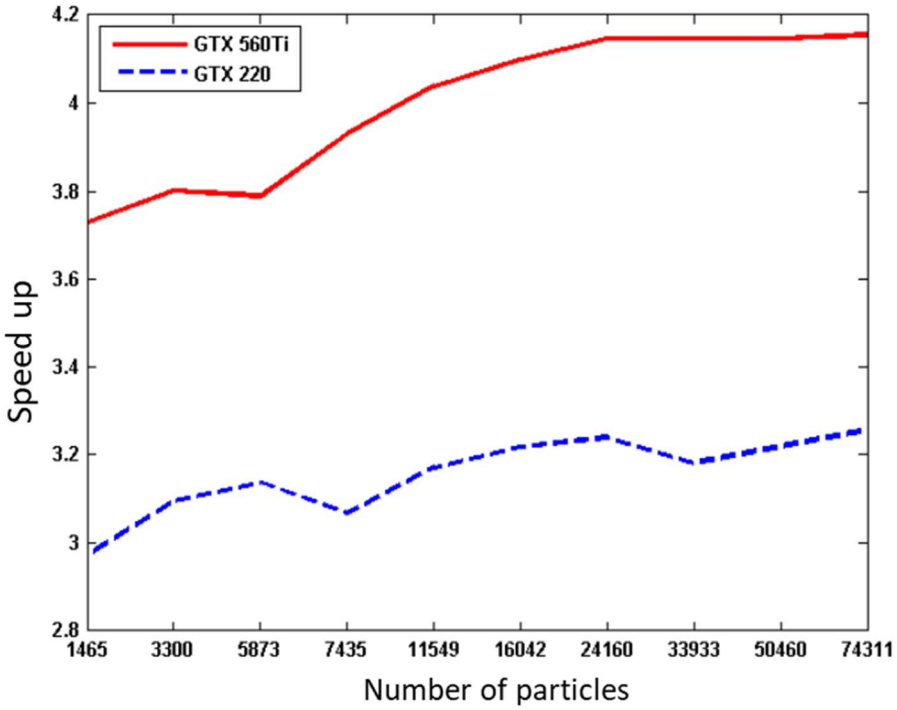

Figure 7 Speed-up by GT 220 and GTX 560 Ti for single precision SPH.

4.3 Effect of Single and Double Precision

Figures 8 and 9 depict a comparison between single precision and double precision with regard to the computational speed-up. The results show that the runtime for double precision code was faster than that for single precision for both CPU and GPU calculations. The GPU calculation with single precision had more speed-up than with double precision.

The GTX 560 Ti did not exhibit better performance in double precision than in single precision. This is due to the execution time that is expressed as floating point per second for single precision being higher than for double precision.

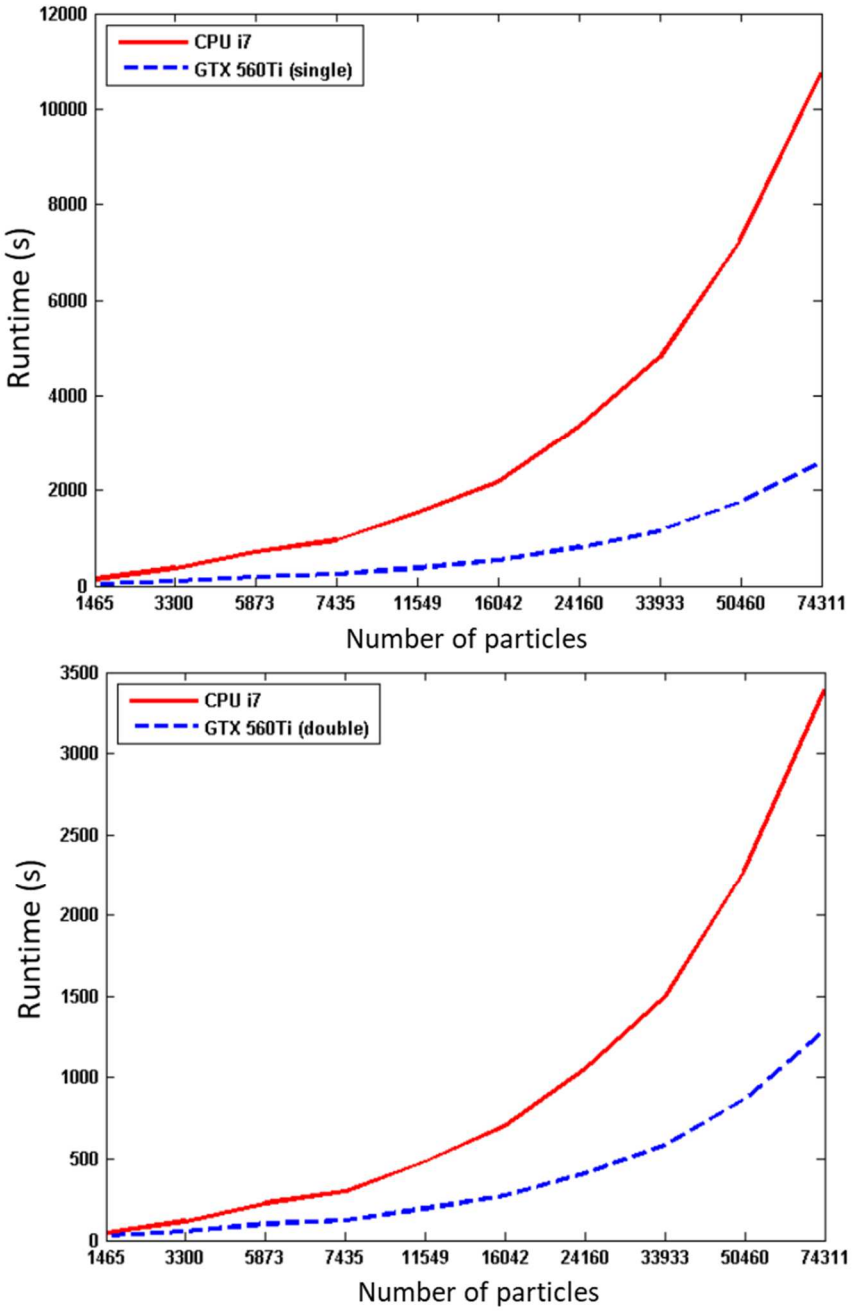

Figure 8 Computation time for single precision and double precision SPH.

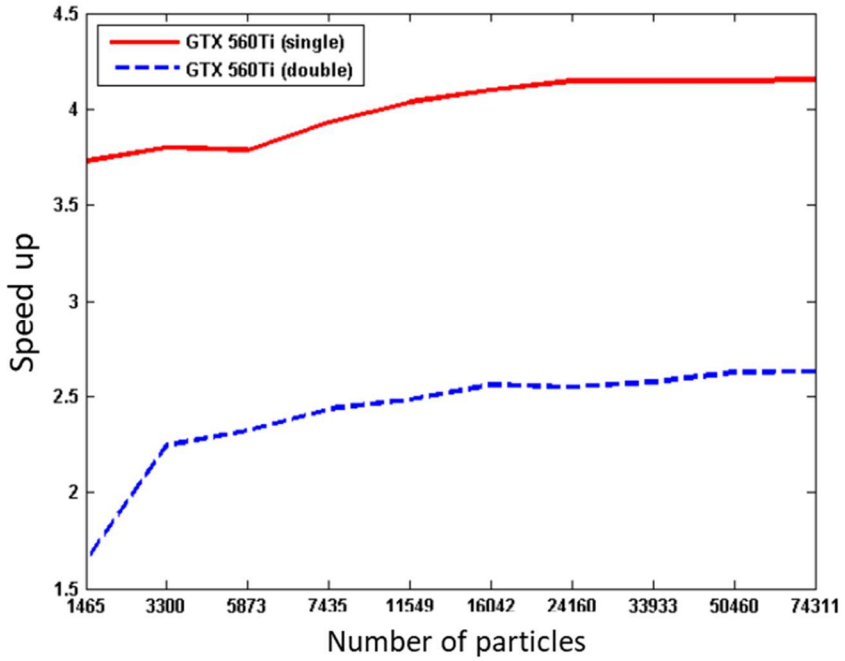

Figure 9 Speed for GTX 560 Ti with single and double precision SPH.

4.4 Accuracy of Simulation Depending on Number of Particles

The accuracy of the simulation depends on the number of particles. The particle-based fluid solver SPH requires a large number of particles to achieve a smooth surface. The Scott Russell wave generator was simulated, which with *21270* particles yielded a runtime of *15.977* hours for a full-time simulation (2.15 second simulation time). The simulation was carried out using an Intel i5 computer and a GT 220 graphic card. Figure 10 shows the free-surface profile for a time instance compared with the analytic solution (black solid-line).

Figure 11 depicts the relative error of the free surface compared to the analytic solution. The smallest error was achieved by the simulation with *21270* particles. It can be clearly seen that the effect on simulation accuracy with a large number of particles is that the larger the number of particles, the better the simulation. For the three-dimensional case, a dam-break problem was adopted (Figure 12). A simulation was performed using *30976* particles, which resulted in a runtime of *14.972* hours for a full-time simulation (5 second simulation time). The simulation was carried out using an Intel i7 computer with a GTX

560Ti graphic card. The results also demonstrate that a smooth fluid surface is achieved by using a large number of particles.

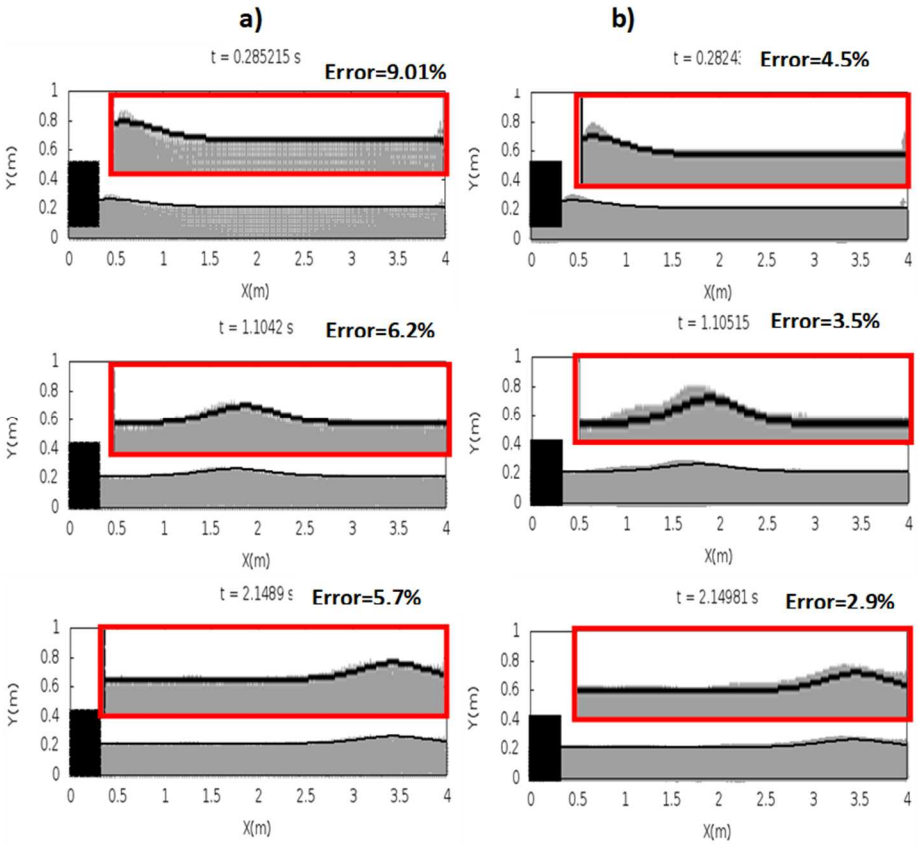

Figure 10 Simulation of Scott Russell's wave generator: a) number of particles = 5,873, b) number of particles = 21,270. The free surface in the simulations is compared with the analytic solution (solid line) and relative errors are indicated. The red boxes are close-up views of the free surface.

It can be seen from Figures 10 and 11 that the relative errors for both cases of Scott Russel's problem with smaller and larger number of particles tended to decrease with time. Furthermore, the overall relative errors with a larger number of particles were about one half times lower than with a smaller number of particles. The relative errors oscillated between 5.7% at later times and 10.2% at early times for cases where the number of particles was 5873 and 9210 and decreased to a level between 5.5% and 3% for the case where the number of particles was 21270. These results generally confirm the need of a sufficient number of particles to produce simulations with an acceptable resolution.

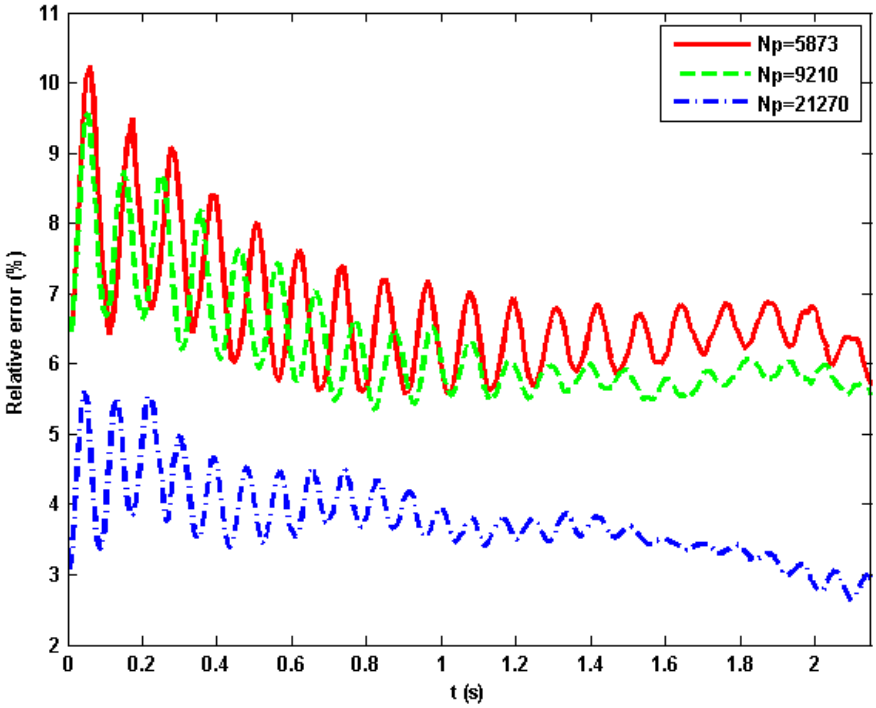

Figure 11 Relative error curve for Scott Russell's wave generator.

The results of the 3D dam-break flow simulation are depicted in Fig. 12. A water column of 0.3 m height is initially bounded by three vertical walls, a vertical gate (0.5 m from the back vertical wall) and a bottom wall whose area is 0.5 x 0.5 m^2. The gate is then instantaneously removed allowing the water to collapse and flow along a horizontal passage under the influence of gravity. The distance between back and front vertical wall is 2.0 m.

The figure shows the flow simulation in the form of normalized velocity, $v/(gh)^{1/2}$ at three successive times t = 1, 2.5 and 5 s. At early times, high velocity distribution appears at the vicinity of the gate's bottom. As time passes, the height of the water column decreases and higher velocity distribution grows forward and backward from the area near the removed gate. Longer distribution of high velocity now fills in the passage and continues until it collides with the front vertical wall, yielding a vertical jet rising along the wall, which tends to overturn backward. Within the scope of this study, the rapid drop of the water column's height also ignites a splash in the area near the back vertical wall, but nevertheless smooth velocity and surface distribution were achieved using 30976 particles.

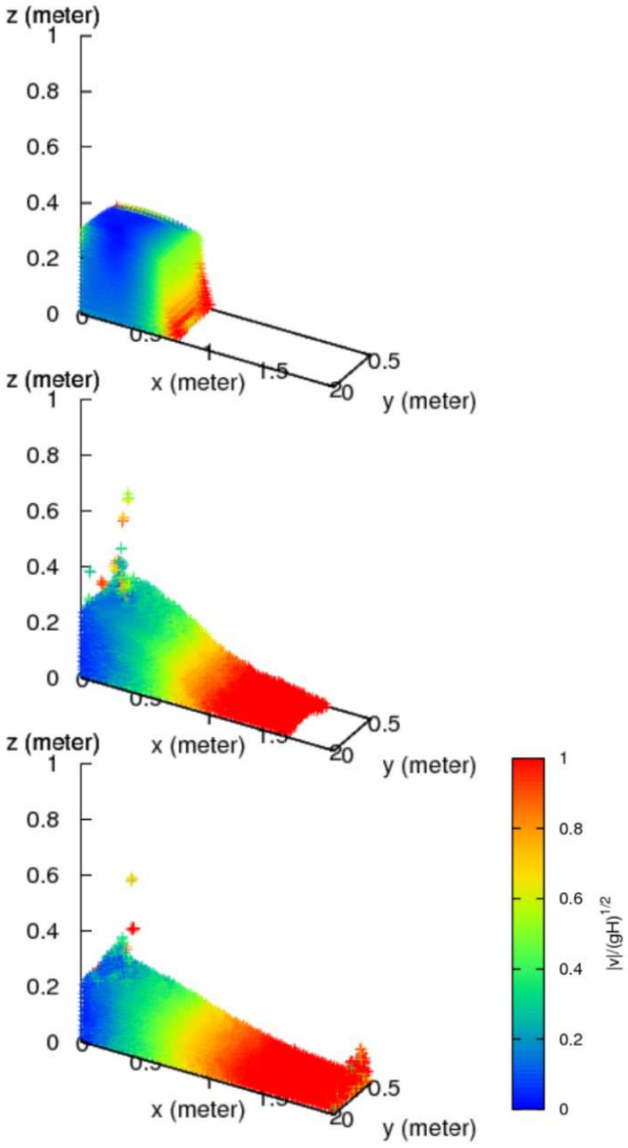

Figure 12 Normalized velocity distribution for 3D dam-break problem at three different times (number of particles = 30976).

5 Conclusions

In this work, the SPH method was implemented on a GPU with CUDA parallel programming. Single precision simulations were calculated on different GPU

cards (NVDIA GTX 560 Ti and GT 220), each with various total numbers of particles. The speed-up of the calculation with the GTX 560 Ti was 3.7 up to 4.15 while with the GT 220 it was 2.9 up to 3.2. The runtime for double precision was shorter than for single precision for both CPU and GPU calculations, but single precision had a higher speed-up than double precision. The last section showed that the accuracy of the method depends on the number of particles. Scott Russell's wave generator was simulated with 21270 particles, producing the smallest relative errors. The 3D case was represented by a dam-break problem with 30976 particles. The overall results show that a runtime improvement of fluid simulation using SPH was achieved using the method proposed in this paper.

References

[1] Kurnia, R., Omata, S. & Kazama, M., *The Stochastic Smoothed Particle Hydrodynamics to Overcome Energy Loss.* Gakuto International series, Mathematical sciences and applications, **34**, pp. 157-174, 2011.

[2] Monaghan, J.J. & Gingold, R.A., *Shock Simulation by the Particle Method SPH*, J. Comput. Phys., **52**, pp. 374-389, 1983.

[3] Goswami, P., Schlegel, P., Solenthales B. & Pajarola, R., *Interactive SPH Simulation and Rendering on the GPU*, ACM SIGGRAPH Symposium on Computer Animation, pp. 1-10, 2010.

[4] Amada, T., Imura, M., Yasumoro, Y., Wanabe, Y. & Chihara, K, *Particle-based Fluid Simulation on GPU*, ACM Workshop on General-Purpose Computing on Graphic Processors, 2004.

[5] Monaghan, J.J., *Smoothed Particle Hydrodynamics*, Annu. Rev. Astron. Astrophys., **30**, pp. 543-574, 1992.

[6] Colagrossi, A. & Landrini, M., *Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics*, J. Comput. Phys., **191**, pp. 448-475, 2003.

[7] Monaghan, J.J., *Simulating Free Surface Flow with SPH*, J. Comput. Phys., **110**, pp. 399-406, 1994.

[8] Monaghan, J.J. & Kos, A., *Solitary Waves on a Cretan Beach*, Journal of waterway, port, coastal, and ocean engineering, **125**, pp. 145-154, 1999.

[9] Monaghan, J.J., *Particle Methods for hydrodynamics*, Computer Physics Report, **3**, pp. 71-124, 1985.

[10] Ashtiani, B.A. & Rezaei, A.M., *Modification of Weakly Compressible Smoothed Particle Hydrodynamics for Preservation of Angular Momentum in Simulation of Impulse Wave Problems*, Coastal Engineering Journal, **51**(4), pp. 363-386, 2009.

[11] Monaghan, J.J. & Kos, A., *Scott Rusell's Wave Generator*, Physics of Fluids, **12**, pp. 622-630, 2000.

[12] Lo, E.Y.M. & Shao, S., *Simulation of Near-shore Solitary Wave Mechanis by an Incompressible SPH Method*, Applied Ocean Research, **24**, pp. 275-286, 2002.
[13] Marrone, S., Colagrossi, A., Le Touze, D. & Graziani, G., *Fast Free-surface Detection and Level Set Function Definition in SPH Solvers*, J. Comput. Phys., **229**, pp. 3652-3663, 2010.
[14] Doring, M., *Développement d'une Méthode SPH Pour Les Applications à Surface Libre en Hydrodynamique*, PhD thesis, École Centrale Nantes, 2005. (Text in French)