An architecture for real time fluid simulation using multiple GPUs

José Ricardo da S. Junior
Mark Joselli
Marcelo Zamith
Marcos Lage
Esteban Clua
Media Lab
Universidade Federal Fluminense

Eduardo Soluri
Nullpointer Tecnologia

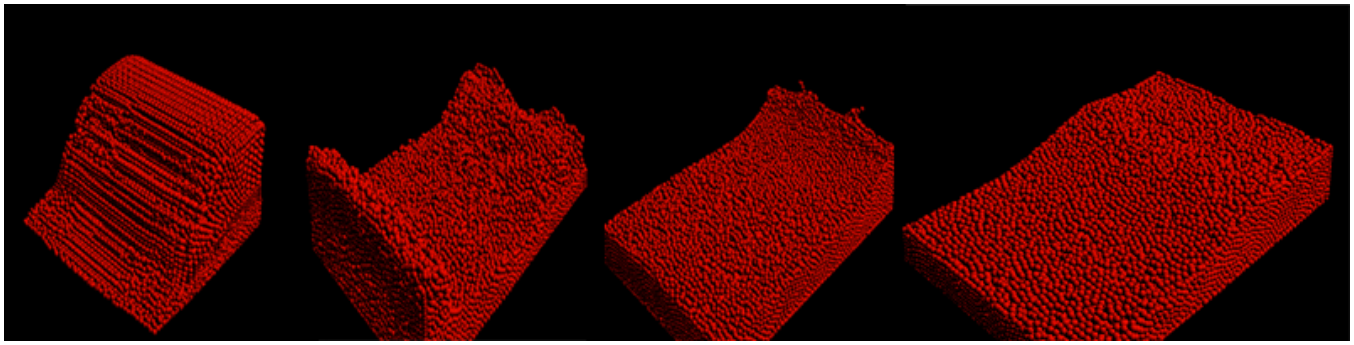

Figure 1: *Real time fluid simulation*

Abstract

Natural phenomena simulation, such as water and smoke, is a very important topic in order to increase real time scene realism in video-games and general real time simulations. However, this kind of simulation requires numerically solving the Navier-Stokes equations, which is a computationally expensive task. Additionally, to deal more immersing simulation, interaction between the flow and the objects in the scene are needed, which increases even more the computational work. The advent of GPU computing has enabled the development of many strategies for accelerating these simulations. In this paper we propose an scalable architecture for multiples GPUs for fluid simulation, allowing complex fluid behavior for real time applications like games. Also, the techniques explained in this paper could be adapted for others real-time simulations such as physics simulations.

Keywords:: CUDA, GPU Computing, Fluid Simulation, SPH, Multi GPU, rigid body

Author's Contact:

{jricardo,mjoselli,mzamith,mlage,esteban}@ic.uff.br
esoluri@nullpointer.com.br

1 Introduction

Realism in video-games is not only a matter of perfect graphics, but also includes the search for real behaviors and physics. While many improvements had been done for dynamic rigid bodies elements, there are still many lacks to be fulfilled in fluids simulation research. This is an important topic for the game industry, since real aerodynamics effects or liquids behaviors are present in almost any title that simulates real environments.

In order to achieve a real immersion, interaction between fluid and rigid bodies placed in the simulated environment must be considered. However, the collision between different objects with the fluid and the approximation of the physical forces coming from these interactions require time consuming computations. Most times those effects are neglected for video-games, since the physics simulation is only one of many tasks that must be handled by the engine.

The first real time fluid simulations were performed in CPU [Kass and Miller 1990]. Most of these works did not consider the interaction between fluid and scene objects in order to achieve interactive frame rates. Later, with the processing power provided by the GPUs, other phenomena could be coupled to real time fluid simulations. The reader can find methods that consider two-way interactions between fluid and its surrounding objects in the woks [Kipfer and Westermann 2006; Kurose and Takahashi 2009].

With the accessibility of modern GPUs by a wide range of people and its processing capability in relation to CPU, this device is being used for solving an enormous set of intensive tasks, such as rigid body [Joselli et al. 2009b], gravitational N-body [Bédorf et al. 2012], crowd simulation [Joselli et al. 2009a] and molecular simulations [Páll et al. 2011], to name a few.

Nowadays, many GPU Computing systems are starting to have multiple GPU devices to solve problems [Kim et al. 2011]. In order to distribute the workload across multiple GPUs, the developer must manage the data exchange between the main memory and these devices, guaranteeing consistency between the multiple copies of data, making the development for these architectures more difcult for the developer.

In this paper we present an architecture for fluid simulation using multiples GPUs, being it in the same machine or in different nodes across a network, allowing more realistic fluid simulation in a virtual scene. Additionally, this architecture is automatically scalable with the number of GPUs available during the simulation. In this work, the fluid simulation in performed through the Smoothed Particle Hydrodynamics (SPH) method, a meshfree particle Lagrangian approach. As far as the authors of this work knows, this is the first real time fluid simulation to use this type of architecture.

The remainder of this paper is organized as follows. After referring to related fluid simulation works, in Section 2, we describe the fluid governing equations in Section 3 and how we deal computationally with it in Section 4. Next, in Section 5, we present the developed acceleration data structure and in Section 6 our approach to deal with more than one GPU. The results are shown in section 7. Finally, in section 8 we present the conclusions of the paper.

2 Related Work

The first physical simulation using an Eulerian grid-based approach was originally proposed by Foster and Metaxas [Foster and Metaxas 1996; Foster and Metaxas 1997]. They proposed to solve the full 3D Navier-Stokes equations in order to recreate visual properties of dynamic fluids. In [Stam 1999], Stam simulated dynamic

gases using a semi-Lagrangian integration scheme that achieves unconditional stability using artificial viscosity and rotational damping. Foster and Fedkiw [Foster and Fedkiw 2001] extended the technique to liquids using both a level-set method and particles inside the liquid. Enright et al. [Enright et al. 2002] added particles outside the fluid for free surface tracking.

In order to allow two-way rigid body interaction, Takahashi et al. [Takahashi et al. 2002] presented a simple method to couple fluids and buoyant rigid bodies using regular grids and a combination between the volume of the fluid and Cubic Interpolated Propagation methods. Génevaux et al. [Arash et al. 2003] used marker particles for free surface representation and to perform interaction with deformable rigid bodies. In [Cohen et al. 2010], the authors used a moving grid to simulate fluid, which allowed to compute the fluids properties anywhere in space.

After the introduction of particle systems by Reeves [Reeves 1983], simulation of natural phenomena using meshless methods started to be proposed. Fluid simulations using the Smoothed Particle Hydrodynamics (SPH) method were proposed by Desbrun and Gascuel [Desbrun and paule Gascuel 1996] to reproduce deformable objects. This approach was latter extended in [Stora et al. 1999], allowing lava simulation through viscosity and temperature coupling.

Müller et al. [Müller et al. 2003] used the SPH method to perform fluid simulation in real time. Latter, the authors extend their work in [Müller et al. 2004] by including fluid interaction with rigid bodies to simulate virtual surgery using a Gaussian Quadrature to distribute ghost particles on rigid bodies surfaces, which are responsible for generating repulsive forces.

Kipfer and Westermann [Kipfer and Westermann 2006] used SPH to simulate fluid flows over deformable terrains. These simulations were performed in GPU using a shader based approach, without considering collision with rigid bodies. Kurose and Takahashi [Kurose and Takahashi 2009] proposed an approach to simulate fluids and rigid bodies with two-way interaction between them using SPH in GPU. The rigid bodies were represented by polygons and they solved a Linear Complementary Problem (LCP) to compute the collision forces between fluid and solids.

Nowadays, many workstations, servers and desktop computers are equipped with multiple GPU devices. The data transfer between the GPUs are normally realized by the host memory [Nukada et al. 2012]. Nowadays, with the direct computing directives of CUDA, the access of memory between the GPUs are made directly, without passing by the host. This fact increases the overall performance of such architecture. But still in this case, the data transfer speed is limited due to the contentions in PCI-Express interfaces.

In order to allows a faster simulation, some kinds of problems are being solved using more than one GPU architecture. Among some works, we can cite [Nukada et al. 2012], which solves a Fast Fourier Transform in 3D using multiples GPUs. Additionally, [Cevahir et al. 2009] computes a fast conjugate gradient in more than one GPU. Using more than one device to solve a problem requires a well established process in order to share and processing data among these GPUs.

The management of a multiple GPU architecture can be hard. There has been some works that deals exclusive with this management [Zamith et al. 2011; Huynh et al. 2012].

3 SPH Approach

In order to perform fluid simulations, we need to solve the system of differential equations:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}.\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}, \tag{1}$$

and

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho\mathbf{v}) = 0, \tag{2}$$

which are known as Navier-Stokes equations for Newtonian incompressible flows. In these equations, ρ represents the fluid's density, $\mathbf{v}$ the velocity field, p the pressure field, $\mathbf{g}$ the resultant of external forces and μ the fluid's viscosity.

Equation (1) is known as *equation of motion* and states that changes in the linear momentum must be equal to the sum of all forces acting in the system. The convective term $\mathbf{v}.\nabla\mathbf{v}$ represents the rate of change of the velocity field in a fluid element while it moves from one position to another. The convective term is null in Lagrangian methods since the particles used on the fluid discretization follows the flow.

Equation (2) is known as *continuity* or *mass conservation* equation and states that in the absence of sinks and sources the amount of mass in the system must be constant. For particle based methods this equation is unnecessary since each particle carries a constant quantity of mass [Müller et al. 2003].

In this paper, the Navier-Stokes equations are solved using the SPH method that was introduced by Lucy [Lucy 1977] and Gingold and Monaghan [Gingold and Monaghan 1977] to perform simulations of astrophysical problems. The SPH is a meshless Lagrangian method that evaluates (anywhere in space) the field quantities defined only at discrete set of particles using a compact support, radial and symmetrical smoothing kernel [Monaghan 1992].

The evaluation of a continuous scalar field $A(\mathbf{x})$ is achieved by calculating a weighted summation of contributions for all particles $i \in [1...N]$, with position $\mathbf{x}_i$, mass m_i and additional attributes A_i using

$$A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r}, h), \tag{3}$$

where ρ_i is the density of particle i, $\mathbf{r} = \mathbf{x} - \mathbf{x}_j$ and $W(\mathbf{r}, h)$ is the smoothing kernel. To compute the density ρ_i of a particle i, we rewrite equation (3) as follows:

$$\rho_i = \rho(\mathbf{x}_i) = \sum_j m_j W(\mathbf{r}, h). \tag{4}$$

The kernel function $W(\mathbf{r}, h)$ must have compact support, i.e. $\int W(\mathbf{r}, h)dr = 1$ and $W(\mathbf{r}, h) = 0$ for $|\mathbf{r}| > h$. According to [Liu and Liu 2003], the value of h must be chosen in order to maintain the number of neighbors inside a particle support approximately 5, 21 and 27 in one, two and three dimensions, respectively.

The gradient and Laplacian of a smoothed attribute function $A(\mathbf{x})$ is the gradient and Laplacian of the kernel function:

$$\bigtriangledown A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \bigtriangledown W(\mathbf{r}, h), \tag{5}$$

$$\bigtriangledown^2 A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \bigtriangledown^2 W(\mathbf{r}, h). \tag{6}$$

In SPH, the pressure field is computed using an equation of state that is a modification from the ideal gas law, as suggested by Desbrun [Desbrun and paule Gascuel 1996]

$$p_i = k(\rho_i - \rho_0), \tag{7}$$

where k is the stiffness constant of the fluid and ρ_0 is its rest density. Finally, the acceleration of particle i is computed as the sum of pressure, viscosity and external forces, being the last one the sum of rigid body interaction and gravitational forces.

In this work, as advised by [Müller et al. 2003], three smoothing kernels are used. For the density, the smoothing function used is

$$W(r, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \textit{otherwise} \end{cases} \tag{8}$$

where r represents the vector $\mathbf{r}$ magnitude.

Using this function for pressure force calculation tends to a grouping particles behavior [Müller et al. 2003]. In order to avoid this problem, this work uses the smoothing function

$$W^{pressure}(r,h) = \frac{15}{\pi h^6} \begin{cases} (h-r)^3 & 0 \leq r \leq h \\ 0 & otherwise. \end{cases} \quad (9)$$

Finally, the smoothing function used for viscosity calculation is

$$W^{viscosity}(r,h) = \frac{15}{2\pi h^3} \begin{cases} \frac{-r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & otherwise. \end{cases} \quad (10)$$

as equation 8 leads to small instabilities.

4 Fluid computation

Fluid simulation using the SPH approach requires the execution of some ordered tasks as presented in Figure 2. In this section, we describe the most important aspects of some tasks, absent the *particle's structuring*, which is not an intrinsic task for fluid processing and will be explained later.

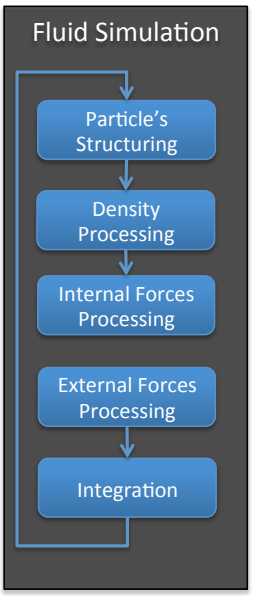

Figure 2: *Necessary stages for performing fluid simulation.*

4.1 Forces calculation

During fluid and rigid body simulation, external forces coming from collision as well as the ones caused by gravity are computed. Also, two-way coupling between rigid body and fluid are only computed for the rigid body's particles that had collided in the broad phase step, allowing the fluid to continue its own processing when no collision occurs.

These forces are calculated using the discrete elements method (DEM), which is used for simulating granular materials [Mishra 2003] like sands. The repulsive force f_{ij}, acting on particle i through interaction with particle j is computed using a spring force $f_{i,s}$

$$f_{i,s} = -k(t_{rad} - |r_{ij}|)\frac{r_{ij}}{|r_{ij}|}, \quad (11)$$

and a damping force $f_{i,d}$

$$f_{i,d} = \eta v_{ij}, \quad (12)$$

where k is the spring coefficient, η is the damping coefficient, r_{ij}, v_{ij} and t_{rad} represents the relative distance, relative velocity and i and j each particle radius sum, respectively.

4.2 Boundary conditions

Boundary conditions are performed in this paper using repulsive forces, which are added to particles, in order to push them away from the boundary. Although this problem can be solved by others methods, such as simply do a naive reflection of the particles's velocity, this solution presents a more stability simulation.

Müller et al [Müller et al. 2003] just implements collision with the particles position directly. In this case, when particles collides with a boundary, they are simply pushed away from the object and the velocity component that is perpendicular to the boundary is reflected.

On the other hand, Harada et al. [Harada et al. 2007] just implement collision by affecting the pressure and density of particles. This way, the pressure correction will essentially push away particles from the boundary surface.

The repulsion force used in this paper, is described by Amada [Amada 2006] and is solved by Equation 13

$$f_i^{repulsive} = \begin{cases} K_s d - ((\mathbf{v_i} \cdot \mathbf{n})K_d) & d > \varepsilon \\ 0 & otherwise \end{cases} \quad (13)$$

where d is the particle distance to boundary, ε is the collision accuracy, $\mathbf{v}_i$ is the velocity of particle i, $\mathbf{n}$ is the surface normal of the wall, K_{stiff} is a stiffness parameter and K_{damp} is a dampening parameter. As it is possible to see, this force acts as a spring, as the more a particle penetrates a boundary, more it is pushed away from the boundary, as is possible to see in Figure 3.

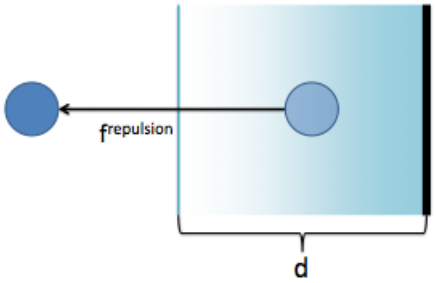

Figure 3: *Particles next to the boundary are pushed away through a repulsive force.*

4.3 Integration

After forces are computed for each particle, it is necessary to integrate them to compute the acceleration, velocity and, finally the position. Integration in this paper uses the explicit Eulerian approach. In this case, the internal and external forces previously computed are integrated for each particle, and its velocity and position variation is calculated.

5 Acceleration structure

Fluid is represented using a collection of particles that interact with each other using the SPH model. This interaction needs to be performed frequently for fluid simulation, as each particle needs to find its neighborhood particles for calculating variables such as pressure and density, according to the SPH method.

This operation has complexity of $O(n^2)$ for a collection of n particles using a brute force method, being an expensive operation, even for a small set of them. To avoid this time complexity, this paper employs an acceleration structure based on hash tables [Harris 2005] for locating nearby particles, which also allows the usage of

an unbounded world. This acceleration structure requires a predefined number of slots, called buckets. Each of these buckets has two variables, indicating the starting and ending offset in the array containing the index for a particle, as shown in Figure 4 for an eight bucket hash grid. A bucket that does not have any particle is set with a special flag, to avoid its wrong computation during simulation.

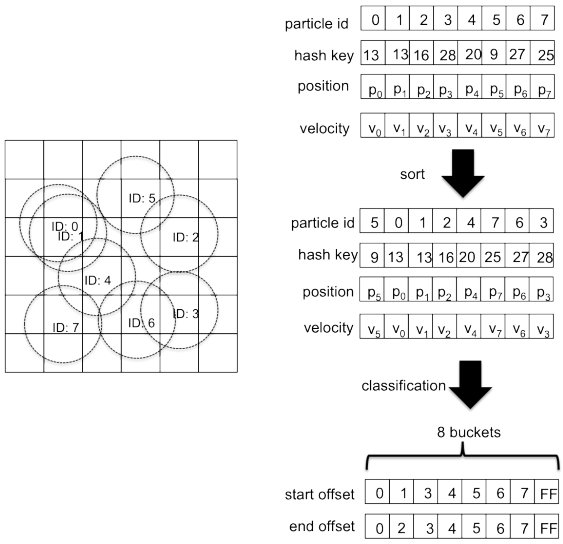

Figure 4: *Process of generating an acceleration data structure based on a hash table.*

Before the hash table processing, a preliminary operation produces a hash key for each particle by using the absolute position of the particle through the use of the algorithm proposed by Teschner et al. [Teschner et al. 2003]. This algorithm receives *p1, p2 and p3*, which are the greatest prime number used to minimize hash key conflicts (chosen in our tests as *73856093*, *19349663* and *83492791*, respectively). It also receives the parameter *cell_size*, that represents the imaginary grid's cell size.

We introduced a new parameter in the proposed algorithm in order to avoid an observed problem. Following the original algorithm the same hash key is produced for particles located at symmetrical positions in the world, causing unnecessary data processing. In our proposal a new parameter named *world_limits* is included in the hash key generation formula. It represents the world's bounding box, calculated at each time step. The algorithm with our modifications is presented in Algorithm 1.

Algorithm 1: Algorithm for hash key genenation.

Input: float3 pos, float3 cell_size, int num_buckets, float3 world_limits
Output: unsigned int hash_code
int x ← (int) ((pos.x + world_limits.x) / cell_size.x);
int y ← (int) ((pos.y + world_limits.y) / cell_size.y);
int z ← (int) ((pos.z + world_limits.z) / cell_size.z);
hash_code ← ((x * p1) xor (y * p2) xor (z * p3)) mod num_buckets;

Following, after each particle's hash key calculation, it is necessary to sort these particles based on its calculated hash key. This operation is done in GPU by using the radix sort algorithm [Huang et al. 2009], presented in CUDPP library [1]. This way, using a imaginary cell size equals the kernel radius K_r, only 27 buckets are processed during fluid's particle processing (three grid cells in each dimension).

6 Multiples GPUs architecture

According to SPH model, it is required for each particle in a cell to know its neighbor in order to process pressure and forces [Müller et al. 2003]. This fact leads to a dependency between them, as presented in Figure 5.

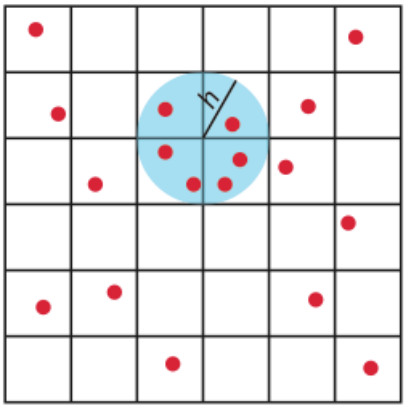

Figure 5: *Particles' data dependency located in different cells.*

The first strategy to allows more than one GPU to process fluid's particles is to split up the domain among the available GPUs, being particles located in the border processed differently from the ones located inside the grid. In this case, particles that are not in the edge of the grid can be processed normally, as all its dependency data is available on the same GPU. On the other hand, particles that are located in the edge cannot be processed, as half of its dependency data are located on another GPU's memory. Figure 6 shows this technique for 2 GPUs in a host. For the cases where GPUs are distributed over nodes in a network, this data can be transferred using NVidia GPU Compute Direct with CUDA language [Corporation 2012], which enables peer GPU to GPU memory communication over a network. For local GPUs, this data can be shared using Per to Per (P2P) communication, which enables a single view of the whole GPU's memory in the host.

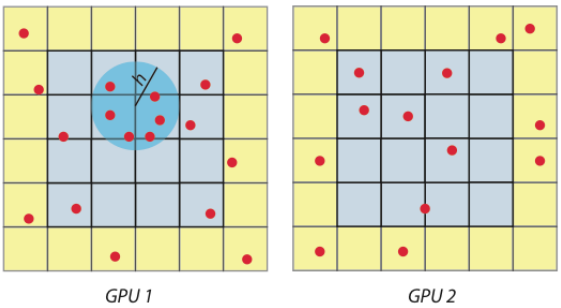

Figure 6: *Simulation domain distributed over 2 GPUs. Particles located at yellow cells need to be transferred between them before its computation, while particles at blue cells can be processed independently.*

Processing fluid's particles using a set of GPUs located at independent nodes in a network is done by a collection of ordered tasks, according to Algorithm 2. Some of the tasks that needs to be performed require more computational effort than others, as presented in Figure 7, which shows the distribution of time over all tasks that need to be performed for fluid simulation. In order to avoid the complexity of the code for minor performance, in a first attempt, only fluid's tasks that requires more computational effort are distributed among various GPUs.

As can be seen in Figure 2, the first task that needs to be done is called **particle's structuring**. Mainly this task calculates a hash code for each particle in order to group them in the same cell. This task is performed by only one GPU, as concurrency data writing is needed. It is important to state that this task is not an intrinsic part of fluid processing, used here only to avoid de $O(n^2)$ complexity. Additionally, beside this task, all subsequent computations are done in parallel by available GPUs.

For all subsequent tasks, processing particles requires data access to

[1] Available at http://gpgpu.org/developer/cudpp

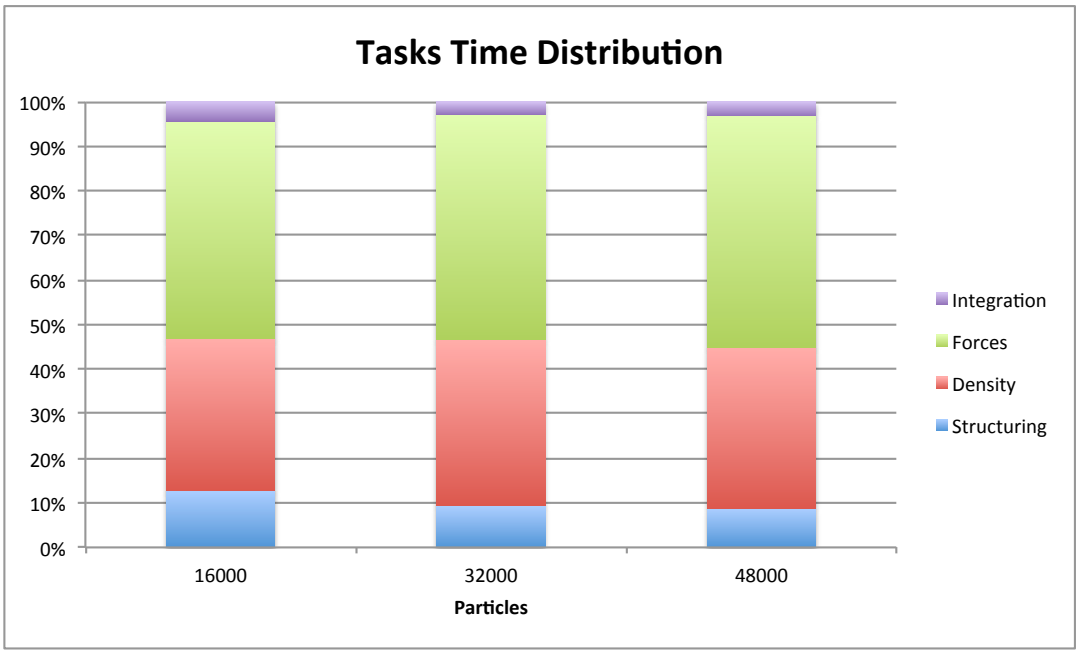

Figure 7: *Time distribution for fluid's tasks.*

Algorithm 2: Algorithm describing the high level steps during fluid simulation for various GPUs.

```
Input: particles, num gpus
particle structuring(GPU master);
synchronize();
amount particles per gpu = particles / num gpus;
for availabe GPU do
    share-data(edge-position-particles);
    calculate-density(inner-particles);
end
synchronize();
for availabe GPU do
    calculate-density(edge-particles);
    calculate-forces(inner-particles);
    integrate(inner-particles);
    share-data(edge-density-particles);
end
synchronize();
for availabe GPU do
    calculate-forces(edge-particles);
    integrate(edge-particles);
    share-data(all-particles-positions);
end
synchronize();
```

all neighborhood particles inside the *kernel radius*, such as position, velocity and density. To allow data sharing, they are done by a collection of synchronized steps, according to Figure 8.

In the first step, after particles have been grouped, the main GPU, which stores all the particles in the simulation, starts to divide, equally, all of them among the available GPUs and starts sending their respectives data for processing. As long as all GPUs receives its data, immediately it starts a kernel to perform the calculation of all the particle's properties, such as force and velocity, as explained in Section 4. During this calculation, it is important to notice that only particles that are not in the grid's edge can be processed, as data in the grid's edge are not yet available. In parallel, each GPU in the node starts sending particles located at its edge to it's neighborhood GPU. This information is made available for each GPU by the *master GPU*, the ones who manages all tasks performed by each GPU.

After the end of the previous step, each GPU is responsible to check if all required information is available. In this case, after finishing the previous step, each GPU starts the second step. In this second step, particles located in the edges are processed, using the same tasks that were performed in the particles in the previous step.

Finally, in the *integration* task, new velocities and position are calculated for all particles and sent back to *master GPU*. It is important to notice that, absent for the integration task, all the other one do not need to synchronize, allowing the GPU to stay busy all the time.

In the first version, our approach are able to deal with GPUs located on the same machine, using the P2P communication among the GPUs. Even with this restriction, our architecture leads to a better speedup, as communication with CPU memory is avoided. For future versions, we will allows fluid simulation to be performed over various GPUs located in different hosts over the network.

7 Results

This section presents the results obtained from our multiple GPU architecture for fluid simulation. For these tests, a PC equipped with an Intel Core i7 using 32 GB of RAM and two NVidia GeForce 580 GTX with 1 GB DDRAM was employed. Simulations tests with different configuration were performed. Fluid rendering is done in screen space through applying a bilateral filter in sphere's normals.

First, Table 1 shows the simulation of fluid made using both a single GPU and our multi GPU architecture and its graph in Figure 9. The column labeled FPS represents the *frames per second* which measure the time necessary to update and render the simulation, while the time represents the milliseconds spend for rendering a frame. *Speedup* is measured by the relation of column X^1 over Y^2.

Table 1: *Results of fluid simulation using both a single GPU and our multi GPU architecture.*

	Single GPU		Multi GPU		
Particles	FPS[1]	Time	FPS[2]	Time	Speedup
32,000	260	3.84	310	3.22	1.19
64,000	151	6.62	205	4.87	1.35
96,000	105	9.52	180	5.55	1.71
128,000	80	12.50	130	7.69	1.62
256,000	41	24.39	55	18.18	1.34
512,000	15	55.66	23	43.47	1.53
1,024,000	5	200.00	9	4	1.8

According to the presented result, it is possible to see that using more than one GPU has increased the overall performance of the

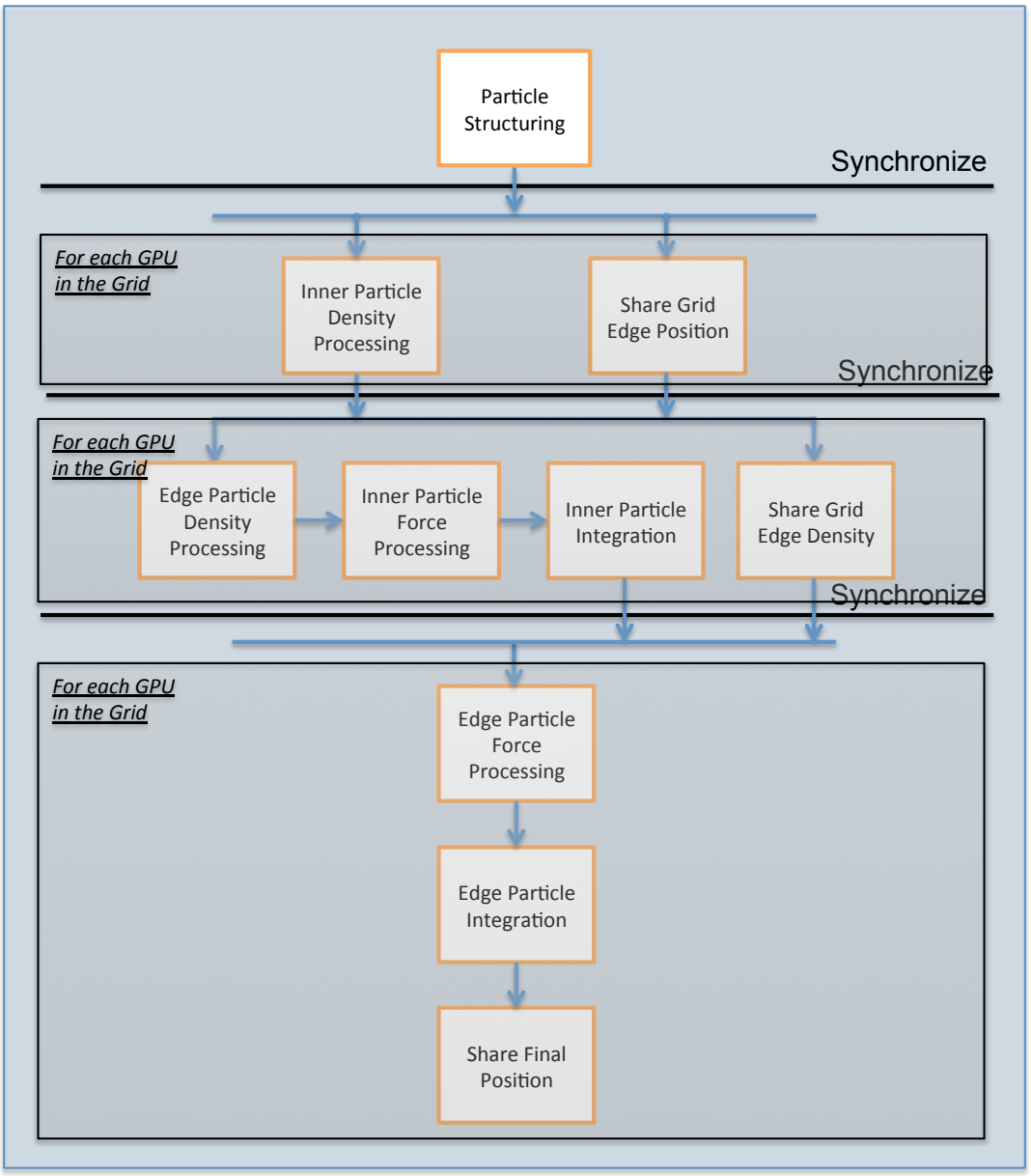

Figure 8: *Fluid tasks distribution over our multi GPU architecture.*

fluid simulation. In this case, growing the number of available GPUs decreases the time to perform this simulation, allowing more particles to be used in order to grow up the realism of the simulated fluid. Figure 10 a simulation screen is shown using a total of 512,000 particles.

8 Conclusions and future works

Our proposed simulation using a multi GPU architecture reached a speedup of almost twice the implementations of the most recent full GPU based approaches, allowing the simulation of more complex fluid's behavior in real time games and other kinds of simulation. The main acceleration factor comes from the fact that many complex and time consuming tasks can be split up among a collection of GPUs during the simulation processing. Additionally, it is important to notice that the same approach can be applied for a set of correlated problems, such as biological molecules simulation.

According to the results presented, the proposed architecture is being extended in order to enable the use Dynamic Parallelism of the newest Kepler GPUs architecture, which allows dispatch of CUDA kernel inside the kernel being processed. This way, we can avoid spend time processing empty cells during the simulation.

Additionally, due to the capacity of GPUs, an architecture that sends data for processing according to each device capability would be very beneficial for the whole simulation. This way, less fluid's particles are sent for GPUs with less capability, avoiding possible bottlenecks.

Rendering techniques allowing implicit surface reconstruction and shading are also being studied in order to allows its fully usage in the game industry.

Acknowledgements

The author thank all the Computation Institute at Federal Fluminense University for their support. Financial support from CAPES is acknowledged.

References

AMADA, T. 2006. *Real-time particle-based fluid simulation with rigid-body interaction*. Charles River Media, Singapore.

ARASH, O. E., GNEVAUX, O., HABIBI, A., AND MICHEL DISCHLER, J. 2003. Simulating fluid-solid interaction. In *in Graphics Interface*, 31–38.

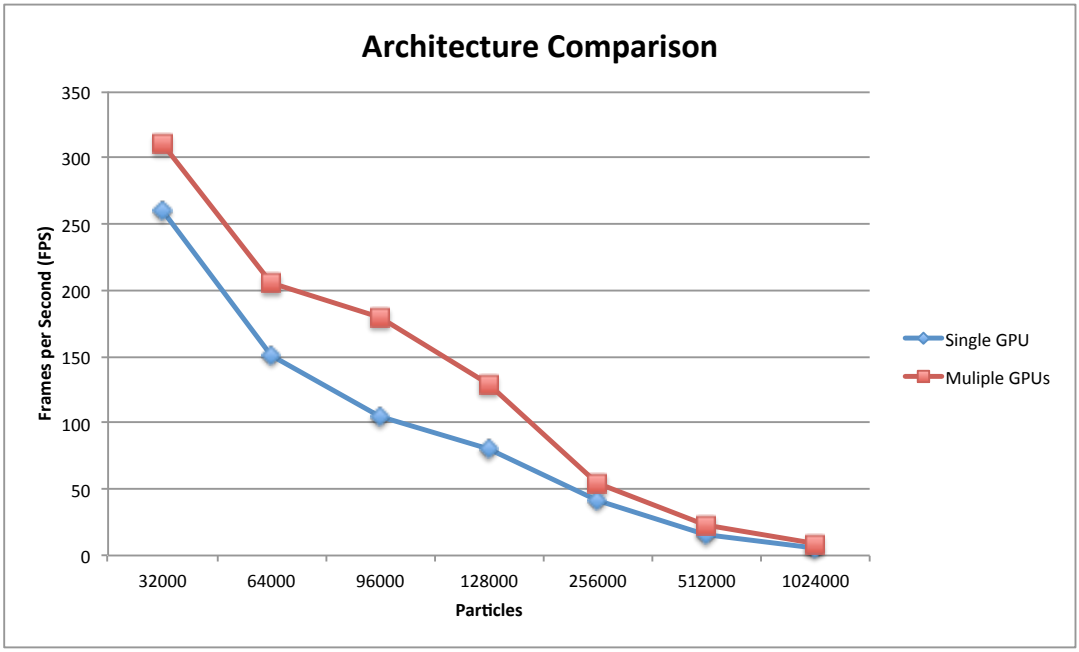

Figure 9: *Fluid simulation comparison using single and multi GPU architecture.*

BÉDORF, J., GABUROV, E., AND PORTEGIES ZWART, S. 2012. A sparse octree gravitational n-body code that runs entirely on the gpu processor. *J. Comput. Phys. 231*, 7 (Apr.), 2825–2839.

CEVAHIR, A., NUKADA, A., AND MATSUOKA, S. 2009. Fast conjugate gradients with multiple gpus. In *Proceedings of the 9th International Conference on Computational Science: Part I*, Springer-Verlag, Berlin, Heidelberg, ICCS '09, 893–903.

COHEN, J. M., TARIQ, S., AND GREEN, S. 2010. Interactive fluid-particle simulation using translating eulerian grids. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 15–22.

CORPORATION, N. 2012. Nvidia cuda programming guide.

DESBRUN, M., AND PAULE GASCUEL, M. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *In Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation*, Springer-Verlag, 61–76.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3, 736–744.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58*, 5, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 181–188.

GINGOLD, R. A., AND MONAGHAN, J. J. 1977. Smoothed particle hydrodynamics - theory and application to non-spherical stars. In *Royal Astronomical Society, Monthly Notices, vol. 181*, 375–389.

HARADA, T., KOSHIZUKA, S., AND KAWAGUCHI, Y. 2007. Smoothed particle hydrodynamics on GPUs. In *Computer Graphics International*, 63–70.

HARRIS, M. 2005. Fast fluid dynamics simulation on the gpu. In *ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, SIGGRAPH '05.

HUANG, B., GAO, J., AND LI, X. 2009. An empirically optimized radix sort for gpu. *Parallel and Distributed Processing with Applications, International Symposium on 0*, 234–241.

HUYNH, H. P., HAGIESCU, A., WONG, W.-F., AND GOH, R. S. M. 2012. Scalable framework for mapping streaming applications onto multi-gpu systems. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, ACM, New York, NY, USA, PPoPP '12, 1–10.

JOSELLI, M., PASSOS, E. B., ZAMITH, M., CLUA, E., MONTENEGRO, A., AND FEIJO, B. 2009. A neighborhood grid data structure for massive 3d crowd simulation on gpu. *Games and Digital Entertainment, Brazilian Symposium on 0*, 121–131.

JOSELLI, M., ZAMITH, M., CLUA, E., MONTENEGRO, A., LEAL-TOLEDO, R., CONCI, A., PAGLIOSA, P., VALENTE, L., AND FEIJÓ, B. 2009. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Comput. Entertain. 7*, 4, 1–15.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 49–57.

KIM, J., KIM, H., LEE, J. H., AND LEE, J. 2011. Achieving a single compute device image in opencl for multiple gpus. In *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, ACM, New York, NY, USA, PPoPP '11, 277–288.

KIPFER, P., AND WESTERMANN, R. 2006. Realistic and interactive simulation of rivers. In *GI '06: Proceedings of Graphics Interface 2006*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 41–48.

KUROSE, S., AND TAKAHASHI, S. 2009. Constraint-based simulation of interactions between fluids and unconstrained rigid bodies. In *Proceedings of Spring Conference on Computer Graphics*, 197–204.

LIU, G. R., AND LIU, M. B. 2003. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method; electronic version*. World Scientific, Singapore.

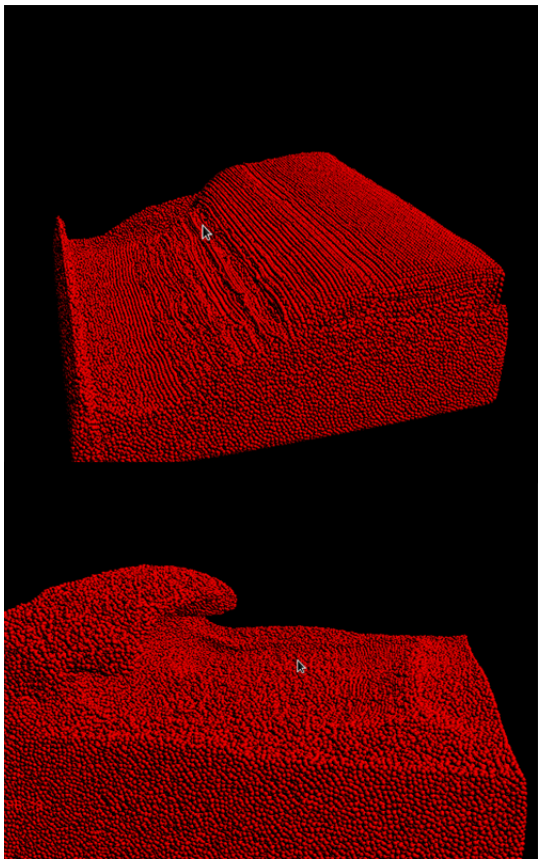

Figure 10: *Fluid simulation performed with 512,000 particles using our multi GPU architecture.*

LUCY, L. B. 1977. A numerical approach to the testing of the fission hypothesis. In *Astronomical Journal, vol. 82*, 1013–1024.

MISHRA, B. K. 2003. A review of computer simulation of tumbling mills by dem part i - contact mechanics. In *International Journal of Mineral Processing, Vol. 71(1-4)*, 73–93.

MONAGHAN, J. J. 1992. Smoothed particle hydrodynamics. In *Annual review of astronomy and astrophysics. Vol. 30*, 543–574.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.

MÜLLER, M., SCHIRM, S., TESCHNER, M., HEIDELBERGER, B., AND GROSS, M. 2004. Interaction of fluids with deformable solids. *Comput. Animat. Virtual Worlds 15*, 3-4, 159–171.

NUKADA, A., MARUYAMA, Y., AND MATSUOKA, S. 2012. High performance 3-d fft using multiple cuda gpus. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, ACM, New York, NY, USA, GPGPU-5, 57–63.

PÁLL, S., HESS, B., AND LINDAHL, E. 2011. Poster: 3d tixels: a highly efficient algorithm for gpu/cpu-acceleration of molecular dynamics on heterogeneous parallel architectures. In *Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion*, ACM, New York, NY, USA, SC '11 Companion, 71–72.

REEVES, W. T. 1983. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2, 91–108.

STAM, J. 1999. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128.

STORA, D., AGLIATI, P.-O., CANI, M.-P., NEYRET, F., AND GASCUEL, J.-D. 1999. Animating lava flows. In *Proceedings of the 1999 conference on Graphics interface '99*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 203–210.

TAKAHASHI, T., UEKI, H., KUNIMATSU, A., AND FUJII, H. 2002. The simulation of fluid-rigid body interaction. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*, ACM, New York, NY, USA, 266–266.

TESCHNER, M., HEIDELBERGER, B., MUELLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. In *In Proceedings of VMV'03*, 47–54.

ZAMITH, M., VALENTE, L., JOSELLI, M., CLUA, E., TOLEDO, R., MONTENEGRO, A., AND FEIJ, B. 2011. Digital games based on cloud computing. In *SBGames 2011 - X Simpsio Brasileiro de Jogos para Computador e Entretenimento Digital*.