A heterogeneous multicore CPU and GPU architecture for two-way real time fluid simulation

José Ricardo da S. Junior
Esteban Clua
Mark Joselli
Marcelo Zamith
Federal Fluminense University

Eduardo Soluri
Nullpointer Technology

Abstract

Natural phenomena simulation, such as water and smoke, is a very important topic to increase real time scene realism in video-games. However, the computational fluid simulation is an expensive task since we must numerically solve the Navier-Stokes equations. Additionally, an immersing simulation requires interaction between the flow and the objects in the scene, increasing even more the computational work.

In this paper we propose an heterogeneous multicore CPU and GPU scalable architecture for fluid simulation with two-ways interaction with solid objects. We also show the impact of this architecture over GPU and CPU bounded simulations and present results that can reproduce complex fluid behavior in real time applications like games.

Keywords:: Fluid simulation; CUDA; GPU; load balance

Author's Contact:

jricardo,esteban,mjoselli,mzamith@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.

Most works in games develop strategies in which the steps of simulation are performed exclusively by the CPU or GPU. In cases where the GPU is used, the CPU remains idle on the simulation computation, being responsible only for coordinating the graphic processor tasks. Good engine architectures uses this idle time to other tasks, such as artificial intelligence, culling or data structures manipulations. However, as modern CPUs have many cores, most of them remain stranded even with these tasks distribution. In many cases, this lack of time is due to the long time required to transfer data between CPU and graphic device.

In this paper we present a new and efficient architecture for simulating fluid and rigid bodies using a heterogeneous multicore CPU and GPU system, which allows two-way interaction between them.

The remainder of this paper is organized as follows. After referring to related fluid simulation works, in section 2, we describe our simulation strategy in section 3. Next, in section 4, we present the developed acceleration data structure, in section 5, and we introduce our heterogeneous architecture for multicore CPU and GPU. The results are shown in section 6. Finally, in section 7 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]. 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.

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.

3 Simulation Strategy

In order for solving fluid governing equations, we use the Smoothed Particle Hydrodynamics (SPH), adopting the model proposed by Müller [Müller et al. 2003]. The reader who wants more information about it is advisable to look at this reference.

For rigid body, we discretize its polygons into a set of equally radius spheres. This way, performing collision detection between rigid bodies and fluids is a matter of checking collision between spheres, as fluid is also represented in the simulation by a collection of equally radius spheres. For this discretization, we use a modified version of the depth peeling algorithm [Everitt 2001], originally used for rendering transparent polygons. The results are presented in Figure 1. It is important to notice that these particles are used only during collision detection with fluid and others rigid bodies in the scene.

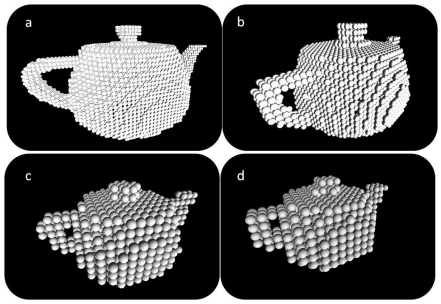

Figure 1: *Teapot processing using variable radius parameter: (a) 0.3 radius with com 7337 spheres; (b) 0.5 radius with 2674 spheres; (c) 0.9 radius with 600 spheres; (d) 1 radius with 668 spheres.*

4 Acceleration structure

Fluids and rigid bodies are represented using a set of particles that interact with each other. This interaction needs to be performed frequently for fluid simulation, as each particle needs to find its neighborhood particles for calculating variables like pressure and density, according to the SPH method. It's also necessary to retrieve the particles neighborhood relation during the computation of fluid-rigid body and rigid body collisions.

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 for locating nearby particles, which also allows the usage of an unbounded world.

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.

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].

In our proposal, fluid's and rigid body's particles do not have to have the same radius. To achieve this possible radius difference, our work employs more than one hash table, each localized in independent memory spaces. For the fluid's particles case, the kernel radius K_r is used for the size of the imaginary cell size. Derived from the uniform grid adopted in our proposal, only 27 buckets are processed during fluid's particle processing (three grid cells in each dimension).

For the dynamic rigid body case, as previously detailed, the simulated object is discretized into a set of particles that are also manipulated using a hash table. This hash table is independent of the one used by the fluid's particles. Thus, it is located at a different memory space from the former. Knowing that all the rigid bodies are discretized into a set of particles of R_r radius, the same algorithm is used for calculating each particles hash key (using a cell_size of R_r).

Finally, a third hash table is used for sorting static rigid body's particles. Our decision of splitting the simulated rigid bodies into separated hashes is motivated by the fact these particles do not move nor rotate during the simulation, allowing our solution to process them offline and only once, before the simulation actually starts. Hence, the particles of static rigid bodies are inserted into this hash table at the initialization step with its absolute position in space.

In order to compute the fluids and rigid body interaction, for instance, to process collisions between them, it is necessary to access data located at different hash tables. Our solution to this problem was to create a mapping function between those data structures. The function is responsible for mapping a given particle's position from one hash table into a different one.

The mapping function is based on a relation between the properties of the two hash table we want to map. Supposing that X_{to} is the hash table's cell size where a particle is mapping to; and X_{from} is the particle's hash table cell size, we define the transform ratio as $T_r = X_{to}/X_{from}$.

5 Heterogeneous architecture

In this section we present a new heterogeneous architecture that uses multiple CPU cores and GPU for fluid and rigid body simulation, at same time allowing two-way fluid and rigid body interactions. In this heterogeneous architecture we also focus on minimizing data exchange and concurrent data access during the simulation.

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

5.1 CPU-GPU Load balance strategy

Simulating fluid and rigid bodies requires the execution of some ordered tasks and data interchange between them for achieving two-way interaction. This dependency may cause overall system performance degradation in the case when one of these steps takes too long to process data necessary by a dependent stage. To minimize this bottleneck, these tasks need to be well distributed between the GPU and CPU cores in an heterogeneous system, considering the parallel GPU's power compared to the CPUs.

To avoid unnecessary processing, we developed an architecture that process rigid body's particles only when they collide with each other or with the fluid's particles. Thus, when no collision between rigid bodies and fluid occurs, their simulation can proceed independently of each other without any data exchange. On the other hand, in case of collision, these exactly particles must take further processing for a more accurate collision and response calculation.

In our proposed heterogeneous architecture, tasks are distributed between CPU and GPU according to Figure 2. As it can be seen in this figure, the GPU is responsible for processing all fluid simulation stages as well as rigid bodies that have collided with other rigid bodies or the fluid. Our decision is driven by the fact that during collision, a high number of particles need to be processed in the narrow phase for accurate collision detection. In the architecture, the CPU is responsible for the broad phase and the integration step of rigid bodies that had no collision with other rigid bodies nor with the fluid, being processed in parallel using multiple CPU cores during fluid's particles spatial subdivision.

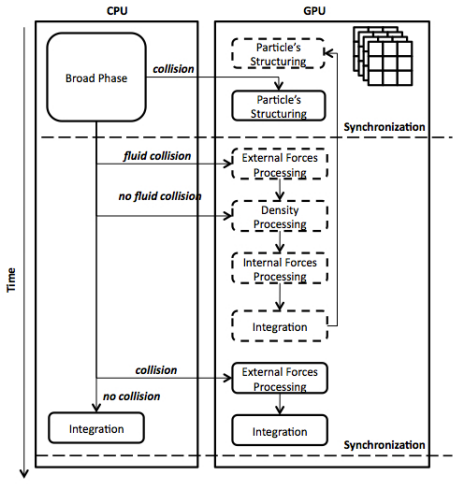

Figure 2: *GPU-CPU task distribution. Dashed blocks represent fluid's tasks while full blocks represent rigid body's tasks.*

5.2 Broad phase

The effectiveness of this heterogeneous architecture and simulation relies on processing only particles that need to be processed for accurate collision and two-way interaction with the fluid. During simulation, the following collision possibilities can be observed: rigid body with fluid, rigid body with other rigid body only, and rigid body with rigid body and also fluid. By detecting these types of collision, the two-way interaction between rigid body and fluid and its associated tasks can be performed only when they are necessary.

The broad phase collision detection is made between fluid and rigid bodies by using an *axis aligned bounding box (AABB)* approach. For this, all the available CPU cores are used through the *OpenMP* library, with allows CPU multithread programming. During this, each CPU core is responsible for performing the collision detection in a subset of all rigid bodies in the scene and the fluid's volume using its bounding box. In case of collision, a flag is checked in a

flag array, indicating if this collision was with a rigid body, fluid or both. For more clarity, this process is shown in Figure 3, where the dashed blocks store the number of collisions detected on each CPU core.

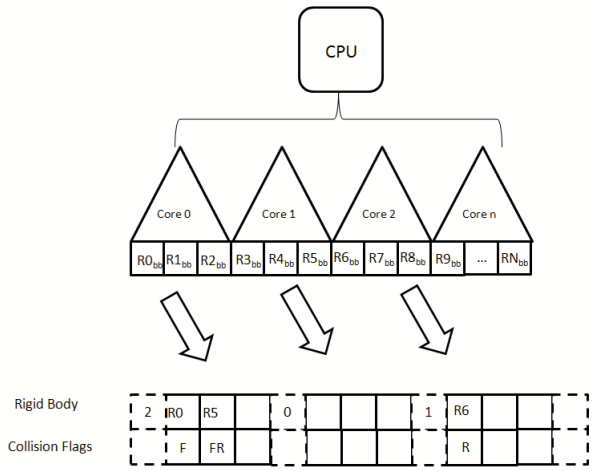

Figure 3: *Performing the broad phase using multiple CPU's cores. Rk_{bb} represents the k-th rigid body's bounding box. F flag represents fluid collision while R flag represents rigid body collision.*

At the end of this process an array containing the potential rigid body colliders whose bounding volume overlapped and a flag array indicating the type of collision is generated. Using these data, a position, an offset and a count array is created independently for the ones that collided with fluid and rigid body. These arrays are responsible for storing the rigid body's relative particles position, the offset index for the starting particle set of each rigid body's in the position array and the particle number of each rigid body, respectively, as presented in Figure 4. Processing interaction between rigid bodies and fluid is only necessary in this array set instead of all elements presented in the simulation, as is commonly done. Rigid bodies that had any collision are processed by CPU's cores just by integrating its center of mass.

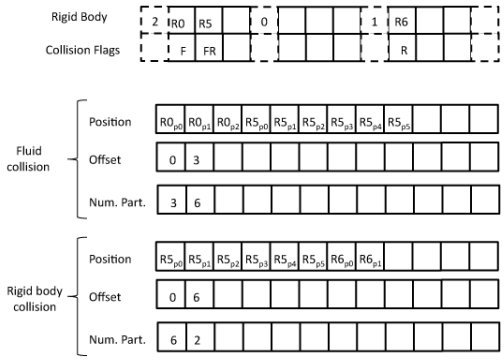

Figure 4: *Data array generated after the broad phase stage in CPU. Rigid body's particles that collide with fluids are stored independently of the ones that collided with fluid.*

5.3 Fluid and rigid bodies collision

After the broad phase step, rigid bodies' particles that collided with the fluid and other rigid bodies are stored in two distinct arrays for the narrow phase processing. Hash key generation and sorting are made only for these particles using its own regular grid, being the narrow phase only processed in these selected particles. In the narrow phase, collision detection is made by using the mathematical sphere equation which informs if collision occurs and the depth of interpenetration.

5.4 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 is 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 forces generated during particles interaction need to obey third Newton's law, which states that for every action there is a reaction of the same magnitude but with opposite direction. So, the same approach presented here is used for fluid/rigid body and rigid body/rigid body interaction, storing each force in its respective particle.

5.5 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.

For fluid's particles, the internal and external forces previously computed are integrated for each particle, and its velocity and position variation is calculated.

For rigid bodies, another approach must be taken due its discretization. During collision, each particle stores its external force and torque amount coming from interaction with other fluid or rigid bodies' particles. At the end, particle's forces and torque must be added and the resulting force and torque applied directly in the rigid body's center of mass. Our approach uses a scan operation [Harris et al. 2007] in GPU for this task. However as rigid body's particles forces and torques are stored in a unique array, performing a scan would add all forces from all rigid body's particles that had collided. To solve this, a segmented scan operation [Sengupta et al. 2007] is used instead of the scan defined by the CUDPP library. The segmented scan employs an auxiliary array indicating the start of each segment in the array to be processed, doing the addition over all particles from different subsets of the whole array.

6 Results

This section presents the results obtained from our heterogeneous multi-core CPU and GPU architecture. For these tests, a PC equipped with an Intel Core 2 Quad Q6600 using 4 GB of RAM and a NVidia Tesla C1060 for processing and a NVidia 8400 GS with 512 DDRAM for visualization was used. Simulations tests with different configuration were performed. During all simulation tests, the *teapot* model was discretized into *76* particles. Fluid rendering is done in screen space through applying a bilateral filter in sphere's normals.

First, Table 1 shows the simulation of rigid body and fluid made entirely in CPU and GPU. The column labeled FPS represents the *frames per second* which measure a time necessary to update and render the simulation. Additionally, it also presents an IPS column that represents the *interaction per second*, which measures the time of updating the simulation, without considering rendering time. *Speedup* is measured by the relation of column X^1 over Y^2.

Table 1: *Results of fluid and rigid body simulation with two-way interaction between them. RB: number of rigid bodies; CF: total of fluid's particles; TPS: total of particle's system.*

			GPU		CPU		
CF	RB	TPS	FPS	IPS[1]	FPS	IPS[2]	Speedup
4096	200	19296	23.3	82.0	0.6	0.6	136.66
4096	400	34496	13.7	36.5	0.3	0.3	121.66
8192	200	23392	14.5	40.1	0.5	0.5	80.20
8192	400	38592	9.7	22.2	0.3	0.3	74.00

The result using the new architecture of heterogeneous multi-core CPU and GPU for processing fluid and rigid body simulation with two-way interaction between them is presented in Table 2 and its

graph in Figure 5. The increased performance of our heterogeneous architecture over GPU bounded simulation is shown in Table 3.

Table 2: *Result of fluid and rigid body simulation in the heterogeneous architecture with two-way interaction between them. RB: number of rigid bodies; CRP: total of rigid body's particles; CF: total of fluid's particles; TPS: total of particle's system.*

				Heterogeneous	
CF	RB	CRP	TPS	FPS	IPS
4096	200	15200	19296	28.2	290.2
4096	400	30400	34496	17.9	238.6
8192	200	15200	23392	17.9	120.5
8192	400	30400	38592	12.9	105.6

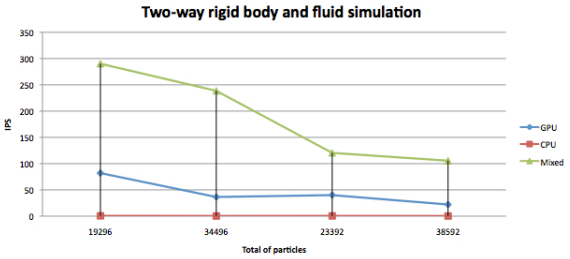

Figure 5: *Graph of rigid body and fluid simulation with two-way interaction in CPU, GPU and heterogeneous (GPU e CPU) system.*

Table 3: *Overall system performance increase using the heterogeneous architecture of multiple CPU cores and GPU over GPU bound. RB: number of rigid bodies; CRP: total of rigid body's particles; CF: total of fluid's particles; TPS: total of system's particles.*

CF	RB	CRP	TPS	GPU IPS[2]	Het. IPS[1]	Speedup
4096	200	15200	19296	82.0	290.2	3.54
4096	400	30400	34496	36.5	238.6	6.54
8192	200	15200	23392	40.1	120.5	3.0
8192	400	30400	38592	22.2	105.6	4.7

In Figure 6 a simulation screen is shown using a total of 80.000 particles, including the fluid's and rigid body's particles.

7 Conclusions and future works

Our proposed simulation using an heterogeneous system based on multicore CPU and GPU reached a speedup of almost four times the implementations of the most recent full GPU based approaches, allowing the simulation of more complex fluid's behavior in real time in games. The main acceleration factor comes from the fact that many complex and time consuming tasks can be avoided during the simulation processing using a broad phase step. In this case, only rigid bodies that potentially collided with fluid and/or other rigid bodies need further and detailed processing. In most cases these number is very low when compared with the amount of rigid bodies in the scene.

Using the rigid body's discretization in this work and the multiple regular grids, level of detail for processing rigid body collision is being developed. In this case, rigid bodies which are in the simulation focus uses a large number of small particles to better perform collision detection and response, while rigid bodies that are not in the simulation focus can use fewer number of big particles to increase the simulation performance during the narrow phase processing. We also state that this could be dynamically adjusted in future works.

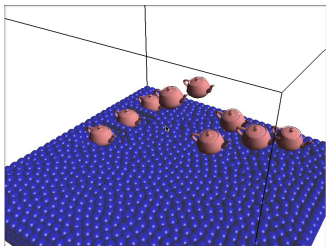

Figure 6: *Fluid and rigid body simulation with two way interaction using our heterogeneous GPU and multicore CPU architecture.*

References

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

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.

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

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., NVidia Corporation.

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.

HARRIS, M., SENGUPTA, S., AND OWENS, J. D. 2007. Parallel prefix sum (scan) with cuda. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, Aug.

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.

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.

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.

SENGUPTA, S., HARRIS, M., ZHANG, Y., AND OWENS, J. D. 2007. Scan primitives for gpu computing. In *Graphics Hardware 2007*, ACM, 97–106.

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.

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.