Journal of Mathematics and System Science 5 (2015) 225-231
doi: 10.17265/2159-5291/2015.06.002

A Research on Particle-Based Parallel Methods for Fluid Animation

Fengquan Zhang and Junli Qin
College of Computer Science, North China University of Technology, Beijing,100144, China.

Received: March 25, 2015 / Accepted: April 22, 2015 / Published: June 25, 2015.

Abstract: In this paper, we present an acceleration strategy for Smoothed Particle Hydrodynamics (SPH) on multi-GPU platform. For single-GPU, we first use a neighborhood search algorithm of compacting cell index combined with spatial domain characteristics. For multi-GPU, we focus on the changing patterns of SPH's computational time. Simple dynamic load balancing algorithm works well because the computational time of each time step changes slowly compared to previous time step. By further optimizing dynamic load balancing algorithm and the communication strategy among GPUs, a nearly linear speedup is achieved in different scenarios with a scale of millions particles. The quality and efficiency of our methods are demonstrated using multiple scenes with different particle numbers.

Key words: CFD, Particle-based, Fluid, GPU

1. Introduction

Although finite elements can create attractive fluid animation, they cannot produce the realistic appearance and behavior when it is used for problems involving large deformations and material fracture. Smoothed Particle Hydrodynamics (SPH) has strong ability in modeling large deformations application. However, a high computational burden makes it impossible for it to meet the requirement of generating and rendering realistic fluid using interactive frame rates.

In this paper, our simulation framework is based on Smoothed Particle Hydrodynamics [1]. The fluid volume is discretized with particles that are used as simulation objects to calculate Navier-Stokes equations. Computational complexity is a challenge of fluid simulation, which can result the simulation non-real-time. SPH method is inherently parallelism, less data dependent, which can be realized through the appropriate parallel modification. In order to accelerate fluid simulation velocity, we implement SPH algorithm on the Multi-GPU. We present a new Multi-GPU-friendly technique for the SPH simulation. The entire SPH simulation and rendering are on the Multi-GPU platform, including neighborhood search, force calculation and surface extracting. According to our knowledge, this is the first Multi-GPU-based system which can accomplish all above mentioned works.

All process of the simulation and rendering are entire implemented on the GPU. We show our experiments using different scale scene. The performance data of our method demonstrates that our system is faster than previous GPU implementations having the same simulation effect and stability as ours. Our approach focuses on three main aspects that are platform performance, simulation speed and rendering quality, which can enforce more large-scale particle simulations and produce better simulation results and with an inspiring image quality.

2. Related Works

A scalar quantity $f(x_i)$ is interpolated at location **r** by a weighted sum of contributions from a finite set

Corresponding author: Fengquan Zhang, College of Computer Science, North China University of Technology. E-mail: fqzhang@ncut.edu.cn.

of sampling points x_j as [2]:

$$< f(x_i) >= \sum_j \frac{m_j}{\rho_j} f(x_j) W(x_i - x_j, h) \quad (1)$$

Where ρ_i is the density of a particle j, m_i is the mass of a particle, and W is a smoothing kernel function with influence radius h. The gradient and Laplacian of the scalar quantity are calculated by with gradient and Laplacian of the kernel respectively:

$$\nabla f(x) = \sum_j \frac{m_j}{\rho_j} f(x_j) \nabla W(x_i - x_j, h) \quad (2)$$

$$\nabla^2 f(x) = \sum_j \frac{m_j}{\rho_j} f(x_j) \nabla^2 W(x_i - x_j, h) \quad (3)$$

The density ρ_i can be computed as

$$\rho_i = \sum_j m_j W(x_i - x_{j,h}) \quad (4)$$

The simplified of the Navier-Stokes equation can be denoted by Equation (5), where $\mathbf{a}$ is the acceleration and integrated with the Leap-frog scheme in our experiment. The particle pressure forces and viscosity forces can be expressed as shown in Equation (6) and (7).

$$\rho \mathbf{a} = f^{pressure} + f^{viscosity} + f^{external} \quad (5)$$

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(x_i - x_j, h) \quad (6)$$

$$f_i^{viscosity} = \mu \sum_j m_j \frac{u_j - u_i}{\rho_j} \nabla^2 W(x_i - x_j, h) \quad (7)$$

Neighbor search is a key step in the SPH method, which costs the most time in simulation. According to the statistics by massive experiments, neighbor search costs 50%-90% of the total time for the simulation. Along with the increase of the number of particles, the more computations cost for neighboring particles search. Therefore, a good neighbor search method is the powerful guarantee in real-time simulation. Harada [3] firstly enforced neighbor search on the GPU which divided the simulation into regular grids. It saves time and memory by creating grid index and using texture memory. Amada [4] proposed a neighbor search method using by dynamic quadtree structure. Serkan [5] first sorted the particles, and then put them into grids, finally searched the neighbor particles with a grid particle mapping method. In paper [6], it presented optimizations for two practical instances of uniform grids, i.e. index sort and spatial hashing.

3. Physical Simulation

3.1 Parallel Analysis

SPH is an interpolation method for particle systems. The field quantities that are only defined by discrete particle locations can be evaluated anywhere in simulation area. In order to get vivid visual effect, it needs much more particles participation that result in simulation slowly. In the CPU program, the simulation cannot reach real-time for larger scale scene simulation. Simulation is limited by particle size seriously. Furthermore it is difficult to ensure the speed and effect of simulation.

As showing in the Fig.1, SPH defines particle i how to compute the value of any attribute value A at an arbitrary position r in space by smooth interpolation over the set of all nearby particles j in the support radius h. When the distance d of particle i and j is less than h, particle j is with contribution to i, and vice versa.

We can come to the conclusion that SPH exists data dependence from Fig.1. To divide serial neighborhood search algorithm into two steps: particle space subdividing and neighborhood search. In each time

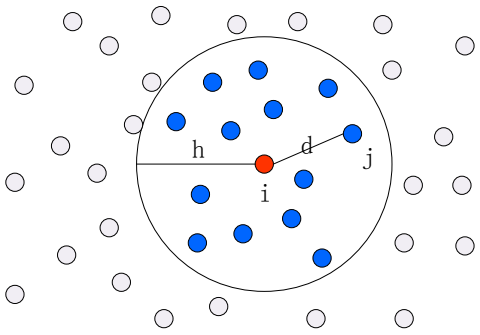

Fig.1 The data dependence in 2D space.

step, subdividing particle into space firstly and mapping it onto grid that reduce the scope of neighborhood search. We use neighborhood search in the process of calculation of density and force respectively. It can save a lot of memory and support more particles simulation. The two different grained parallel schemes, one is a fine-grained that a particle corresponds to a thread, and another is coarse-grained that each thread corresponds to a group particles in the same grid as in Fig.2.

Between of Multi-GPU data transfer is a key factor of decision overall simulation performance. If the amount of data exchange on GPU-GPU is large, the simulation time will extend using the traditional method.

3.2 The scheme of parallel

To avoid it, we use a distributed load balancing schema for a parallel implementation of SPH fluid simulation. Our approach to load balancing is designed to be lightweight and totally distributed. We split the simulation domain into slices based on the number of particle and SPH computing time, which divide different simulation domain into different GPUs. At a beginning of a new iteration step, each processing unit has a task to execute, which is decided by last iteration in terms of computing time. We can balance the load of per GPU by he predicted computing time. According to each GPU load to partition boundary, ensure for dynamically load balancing between four GPUs as shown in Fig.3.

In the process of the dynamic load balancing, we use the GPU consumption of last time step to predict the GPU boundary of next time step, then exchange particles between GPUs in terms of the boundary, which ensure load balancing of system effectively.

3.3 Compacting cell

With the increase of the number of particles, the memory consumption will also increase. In order to reduce memory consumption, it has to compact the empty memory. As suggested in [6], it proposed to use a secondary data structure which stores a compact list of non-empty cells. To compact empty cells, we introduce a method that based on stream compaction [7]. The particle-based methods depend only on the size of the fluid domain rather than the size of the simulation domain. So we only store the non-empty cells, that compact cells and save the large amounts of memory. By constructing cell, each cell is assigned to a sorted index. Memory for a used cell is allocated if it contains particles and deallocated if the cell gets empty. Compared with the basic uniform grid, the memory consumption scales with the number of particles and not with the simulation domain. Compacted cells can also ensure that the particles of

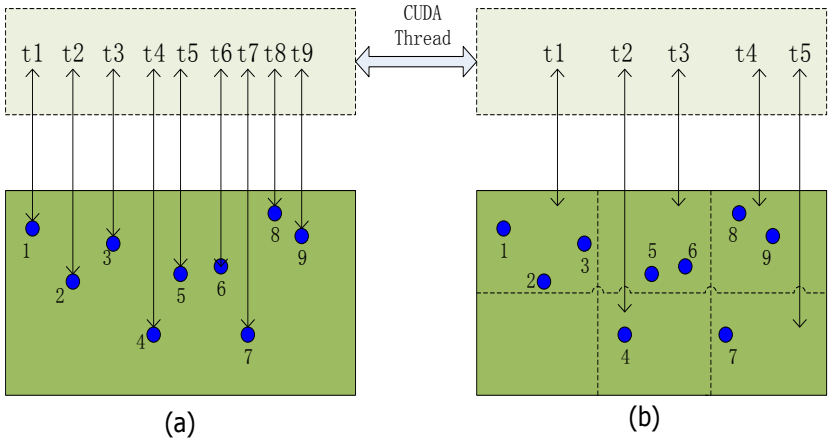

Fig.2 (a) denotes a particle corresponding to thread, which is a fine-grained parallel method. (b) denotes a cell corresponding to a thread, which is a a coarse-grained parallel method.

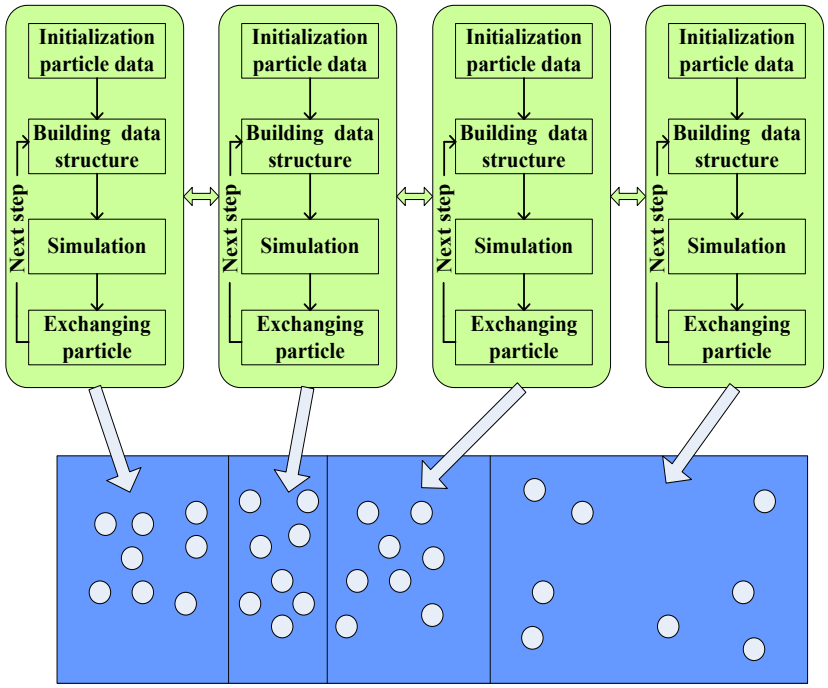

Fig.3 The simulation carried out on 4 GPUs: upper part the phases each GPU executes per step; low pert the region divided into 4 slices of different sizes, associated to its load.

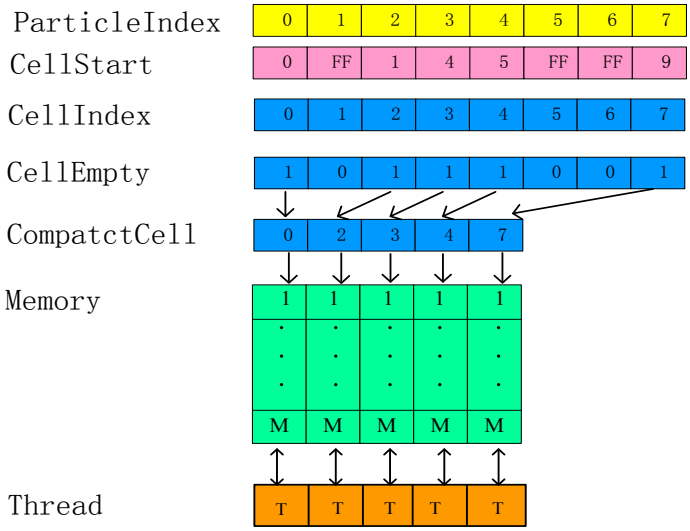

Fig.4 Compacting cell. In the CellStart, FF denotes the empty cell. If the cell is empty, then its value in CellEmpty is set for 0 and the opposite is 1. CellEmtpy store The non-empty cell index is stored in CompactCell that is allocated memory M. Supposing before compacting the number of cell is N and after compacting the number of cell is K. The memory consumption is reduced to $O(K*M+N)$. The neighborhood search only the K used cells.

neighborhood cells and memory location are related, and improve the cache-hit rate. In contrast to hash table, the method can avoid the collision caused by short table. But hash cells may include many non-neighborhood particles when hash table is long, which also bring difficult for neighborhood search and waste the time of access memory, reducing the cache-hit rate. The stream compaction method is illustrated in Fig.4.

Z-indexing [8] is a tracking block method that exactly copes with each particle into a block of shared memory, and assigns each block a constant number of

threads. If the particles are less than the number of threads, the thread will be wasted. As the particles need to be transferred from main memory to shared memory, the cost of transfer is relatively large for Multi-GPU. The paper method increases the time of sorting and judging of neighborhood particles, but avoids the shortcomings of GPU dynamic allocating memory, and saves the memory, using the concept of replacing space with time and fully advantage of GPU acceleration. After constructing particle cells, we will compress the cell to reduce the memory, then sort the cell and particles by cell indexing. Sorted particles have great relativity on the spatial location and memory, which ensure the cache-hit rate.

3. Results and Discussion

3.1 Results

In this section, we show some properties of our SPH fluid simulation. We have enforced and tested both SPH simulation and rendering using OpenGL, CUDA 3.2 and GLSL on the platforms: Windows7 OS, 2x 32-Bit Intel Xeon Eight Core E5620 @2.40GHZ, 6GB RAM and 4x GeForce GTX 480 with 1.5GB video memory.

Serial program needs to store the neighborhood information to avoid their the overhead search. For each particle, the information of searching its neighborhood particles is stored in a separate neighbor list on the CPU, so it wastes storage and seriously restricts fluid simulation. We use parallel SPH method to avoid the problem with CUDA as shown in Table 1.

Our analysis of the memory consumption of the implementation as is shown in the Table 1, which particles denote the number of particle, Cells denote the number of particles sampling cell, and nbx, nby, nbz denote the number of distance field in 3 axes. If adopt this way to simulation, the video memory is occupied 30MB when the size of particle is 298k. For GTX480, the video memory is 1536MB, then it can hold about 1,400k particles. It grows by about 30 times.

During the implementation, we find that texture cache, L1 cache and data type have a very significant impact on performance, so we have also tested them on the GTX480 and GTX280. As shown in the Table 2, where 48KB_L1/16KB_L1 denotes the performance

Table 1 The memory usage of different size of particle.

Particles	Cells	nbx,nby,nbz	Video memory (Byte)
52k	59,319	50,50,50	5,787,532
102k	125,000	50,50,50	10,409,944
152k	185,193	50,50,50	15,527,920
190k	226,981	50,50,50	19,363,284
242k	287,496	50,50,50	24,618,340
298k	357,911	50,50,50	30,310,384

Table 2 Performance ratio of different hardware setting.

Particles	48KB_L1/16KB_L1	GT_TEX/ GT	GT4/GT3(480)	GT4/GT3 (280)
6K	2.845%	-9.339%	1.757%	2.963%
10K	4.053%	-0.965%	1.342%	5.335%
30K	9.802%	3.151%	1.968%	9.969%
52K	9.852%	4.676%	2.870%	10.394%
71K	11.270%	7.228%	4.742%	12.741%
102K	11.112%	8.367%	4.495%	12.335%
152K	12.196%	10.629%	4.753%	14.690%
190K	13.103%	12.606%	4.733%	14.827%
242K	13.524%	13.304%	4.963%	16.252%
298K	14.345%	13.917%	5.204%	17.082%

percentage of different L1 cache setting; GT_TEX/GT denotes performance percentage of using texture memory ratio not using texture memory; GT3 denotes float3, fine-grained parallel, 48k L1 cache, useless texture memory. GT4 denotes float4, fine-grained parallel, 48k L1 cache, useless texture memory. That is, using float 4 is better than float3 in data structure, especially in much more the number of particles. Although float4 wastes some memory, it is good at byte alignment and coalesced access.

Above base test can be looked upon as some performance setting. Now we will experiment on the Multi-GPU using the bunny scene with 3,049k particles as shown in Fig.5.

We find that our implementation is significantly faster than earlier GPU implementations. As shown in **Table 3**, we method is faster than in [8] and [9] in the same hardware platform. We are respectively statistics of the time-consuming of physical simulation and rendering in our SPH simulation.

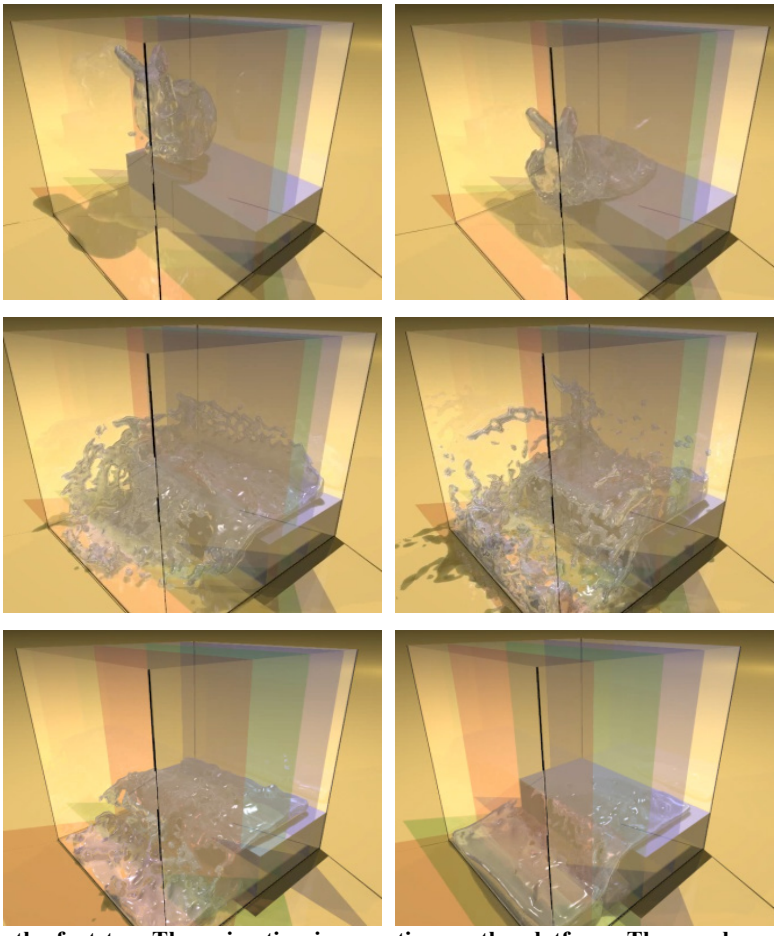

Fig. 5 Bunny falling on the footstep. The animation is execution on the platform. The number of particle is 3,049k. The simulation speed is 9.86FPS. Different color boards denote simulation domain on different GPUs.

Table 3 Simulation and rendering performance results on our platform.

Particle Count	Physical simulation	Rendering	Overall	Real-time interaction
298K	192.34fps	64.15fps	53.15fps	Yes
504K	112.85fps	32.18fps	28.36fps	Yes
1,940K	44.18fps	17.12fps	14.28fps	No
3,049k	29.78fps	12.55fps	9.86fps	No

4. Conclusions

Our experiments demonstrate that using the Multi-GPU can obtain very large performance implements that make it possible to do real-time simulations which previously required costly specialized hardware or took minutes or hours to run. Using our highly optimized framework, we demonstrate a large improvement in performance for the SPH model, and obtain high performance compared to other state-of-the-art implementations. In the future, we will deeply research on CPU-GPU platform architecture for optimization overall performance.

Acknowledgment

This research work is supported by fund of PXM2014_014212_000097 and Y3SJ6600CX.

References

[1] J. Monaghan, Smoothed Particle Hydrodynamics. Report on Progress in Physics, vol. 68, 2005, pp. 1703-1759.

[2] L.B. Lucy, A Numerical Approach to the Testing of the Fission Hypothesis. The Astronomical Journal. Vol. 82(12), 1977, pp. 1013-1024.

[3] T. Harada, S. koshizuka, Y. kawaguchi, Sliced Data Structure for Particle-Based Simulations on GPUs. In Proceedings 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, 2007, pp. 55–62.

[4] T. Amada, and T. Masataka, Particle-based Fluid Simulation on the GPU. International Conference on Computational Science, 2006, pp. 228–235.

[5] B. Serkan, G. Ugur, GPU-Based Neighbor-Search Algorithm for Particle Simulations. Journal of graphics, 2009, pp. 31-42.

[6] I. Markus, A. Nadir, and B. Markus, A Parallel SPH Implementation on Multi-Core CPUs. Computer Graphics Forum, vol. 30, 2010, pp. 99-112.

[7] NVIDIA: CUDA SDK Samples http：//developer.download.v-Vidia.com/compute/cuda/sdk/website/samples.html, 2009.

[8] P. Goswami, P, P. Schlege, B. Solenthaler, and R. Pajarola, Interactive SPH Simulation and Rendering on the GPU, Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, ACM, 2010, pp. 55-64.

[9] Y. Zhang, B. Solenthaler, and R. Pajarola, Adaptive Sampling and Rendering of Fluids on the GPU, Proceedings of IEEE/EG Symposium on Volume and Point-Based Graphics, 2008, pp. 137-146.