IF : 3.62 | IC Value 70.36

Volume-5, Issue-9, September - 2016 • ISSN No 2277 - 8160

Original Research Paper

Engineering

GPU Accelerated Smoothed Particle Hydrodynamics Simulation

Deepa S. Kadam Computer Engineering Department, ICEM, Savitribai Phule Pune University, Pune

Prof. Ram B. Joshi Computer Engineering Department, ICEM, Savitribai Phule Pune University, Pune

ABSTRACT

Smoothed Particle Hydrodynamics is a powerful tool for simulating fluid dynamics. Moreover, it is easily parallelizable, as the interaction between two particles is independent of the others. However, with a large number of particles there would be a significant amount of computation involved for calculating the interaction between the particles. So its necessity of an algorithm that is suitable for such parallelization using GPUs. An analysis of the implementation of smoothed particle hydrodynamics (SPH) simulation in a parallelized manner is presented here. In normal implementation, there is very much data transfer overhead. It is because, after performing physical computation every time it copies all data to the main memory for displaying them. But constantly sending this data makes this computation extremely slow. So to overcome this problem, a parallel implementation of SPH simulation using shared memory is used in proposed work. Speaking simply, proposed work performs all the physical computations at GPU and updated data is sent to the main memory only when it needs to be displayed. This will help to minimize the CPU-GPU data transfer overhead and speeding-up the performance.

KEYWORDS : SPH, parallelization, thread, simulation, particle interaction

INTRODUCTION

Smoothed Particle Hydrodynamics is a commanding mean for simulating fluid dynamics, with an enormous realm of applications. The traditional grid-based numerical methods have troubles in handling some complex phenomena. The reason to use SPH over other numerical methods such as Particle in Cell (PIC), FDM and FEM is that SPH does not depend on the boundary conditions. It means, SPH does not need a grid to calculate any spatial derivatives. Instead, it applies analytical differentiation of interpolation formulae. The equations of momentum and energy become sets of ordinary differential equations which are easy to understand mechanically as well as thermodynamically.

The system focuses on particles themselves represent mass, SPH promises safeguarding of mass without additional computation. It calculates pressure from weighted contributions of neighboring particles reasonably than by solving linear systems of equations. As SPH is a gridless numerical method, it can be used to simulate fluid flows with one or more free surfaces.

The fluid simulation varies in computer graphics, from extremely time consuming quality animations for film and visual effects to simple real-time simulation used in video games. Several other techniques have been developed by researchers since the introduction of CFD. The common techniques among them are Eulerian grid-based methods, Lattice Boltzmann methods, Lagrangian grid-less SPH methods and Vorticity-based methods. Navier-Stokes equations describe the motion of a fluid at any point within a range by a set of non-linear equations. The CFD focuses on studying fluid objects by means of computer graphic techniques and approaches. The fundamentals of any CFD equation are the Navier-Stoke equation.

$$\frac{\partial \bar{u}}{\partial t} + \bar{u}.\nabla\bar{u} + \frac{1}{p}\nabla p = \bar{g} + v\nabla.\nabla\bar{u} \qquad \text{Eq. (1)}$$

The equation is called 'momentum equation' which describes how fluid moves due to external forces. The variables u, ρ, g, v represent velocity of fluid, density, acceleration due to gravity and kinematics viscosity respectively. The equation also describes the 'incompressibility condition'. The earliest attempt in applying computer graphics to solve the Navier-Stoke equations was done by Foster and Dimitri (1996) who described the solutions to simulate liquids. According to Foster and Dimitri, realism is provided through a finite difference approximation to the incompressible Navier-Stoke equations and is coupled with the Lagrangian equations.

According to the research, even the simplest animation exhibits slight realistic behavior not available in previous computer graphics for fluid simulation. This realistic behavior then expanded by Stam. Stam and Chen mapped the surface onto 3D using pressures in fluid flow. This method achieved realistic real-time fluid surface behaviors by applying the physical laws of fluid and avoiding extensive 3D fluid dynamics computations. Their model allows multiple fluid sources to be placed interactively in a dynamic virtual environment. Other efforts by researchers (Chen et. al. 1997) solved the 2D Navier Stoke equations using a computational fluid dynamics method. Cell Indexing approach for searching approximate neighbor particles necessary for efficient fluid simulation using SPH is used by Onderik & Roman for efficient neighbor search of particles. This approach encoded coordinates and index into a key instead of storing particles into a fixed 3D grid or hash map. But it required large amount of memory full 3D grids.

Takahiro Harada achieved accelerated a simulation method for free surface flow by using graphics processing units (GPUs). It became a good solution in sequential manner than the traditional algorithms for satisfying all the requirements of real-time simulations algorithms.

The sequential implementation of SPH is having high overhead of CPU to GPU memory transfer. Additionally, it is having drawback of memory space wastage. The parallel implementation can overcome these drawbacks. This can be achieved at the cost of multiple physics frames per screen frame. In simple implementation, all particles data are copied to GPU. Then all the physics computations are performed, all data is copied back to the main memory, displayed and sent back to GPU. But constantly sending this data make this computation extremely slow. But if we copy the data to GPU only once, perform multiple physics frames per screen frame and when it needs to displayed then only send the particle data back to the main memory.

The subsequent sections describe procedure, strategies, diagrams, results, discussion conclusion and references.

PROCEDURE

This implementation performs the entire SPH computation on the GPU using CUDA by optimizing the neighborhood search, in terms of space and time overhead. In SPH simulation, the neighborhood search, performed on each particle for every time step is computationally most expensive part for implementation. GPU computation will help in accelerating the expensive SPH computations. The simulation domain is distributed into a virtual indexing grid and the grid location of a particle. Virtual indexing grid is divided in X, Y, Z along each of the dimensions. The grid location of a particle is used to determine bit-interleaved Z-index of this virtual indexing grid The Z-index can be computed very efficiently using a table lookup approach.

All particles lying within any power-of-two sized aligned block have contiguous Z-indices while using the z-indexing technique.

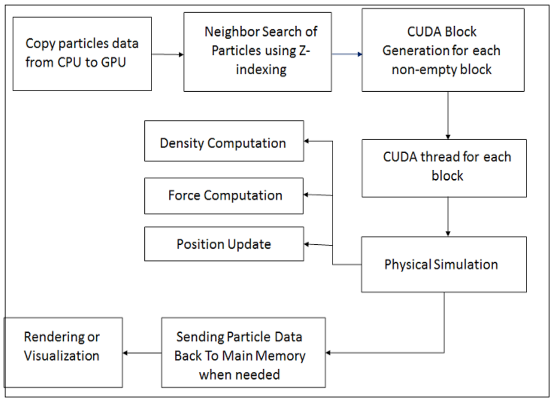

Copy particles data from CPU to GPU
Neighbor Search of Particles using Z-indexing
CUDA Block Generation for each non-empty block
Density Computation
Force Computation
Position Update
CUDA thread for each block
Physical Simulation
Rendering or Visualization
Sending Particle Data Back To Main Memory when needed

Fig. (1) System Architecture for SPH Simulation

The nearest power-of-two block size S in the indexing grid domain can be determined for range queries with given radius R and global support radius of the SPH simulation. Z-indices of all particles are calculated at the start of each time step in parallel. Then these particles are sorted in CUDA using parallel radix-sort. In this way, for each block the index of its first particle and number of particles it contain are determined. With the support of atomicMin operation in CUDA, the first particle in the block can be determined. By incrementing the particle count using atomicInc, the number of particles can be found. Like this, each particle updates both the starting index and particle count of its block in the list B, which is of size $|B| = (X_{max}/S)^3$, where X_{max} is assumed as simulation domain grid dimension.

Density Computation

Density computation of following equation (2) is the initial step to begin SPH simulation.

$$\rho_i = \sum m_j W(r_i - r_j, h) \quad \text{Eq. (2)}$$

where, is mass of particle at position and is smoothing kernel with core radius . These computed densities from last step will be made available for force computation as CUDA texture. Same procedure for neighbor searching and density computation will be followed for force computation using equation (3).

$$f_i^{pressure} = -\sum_j \frac{m_j}{\rho_j}\frac{(p_i+p_j)}{2}\nabla W(r - r_j, h) \quad \text{Eq. (3)}$$

As the updated density values considered necessary for force kernels different kernels need may be there as density and force computation cannot be clubbed together in a single CUDA block. After updating the particle positions and copying them to the position array, radix sort array can be used for block computations so that for block maintenance, there will be no need of extra space.

Surface Particle Extraction

In this method, a particle *i* is considered to be a surface particle if its distance to the center of mass of its neighbourhood is larger than a certain threshold. By summing up the positions $\mathbf{r}_j$ in *i*'s neighborhood weighed by their mass m_j, the center of mass can be calculated as-

$$r_{CM_i} = \sum_j \frac{m_j r_j}{m_j} \qquad \text{Eq. (4)}$$

If very few particles are there, above equation (4) fails to detect surface particles. For this reason, one more constraint is added that is, *i* will always consider being a surface particle if for a particle *i* the number of neighbours lies below a user defined threshold. This condition can be achieved since the computation of neighborhood has already to be calculated in physics simulation part. The particle that belongs to the surface it is written to an output array, otherwise omitted.

The basic stages in the implementation can be understood with following fig. (2).

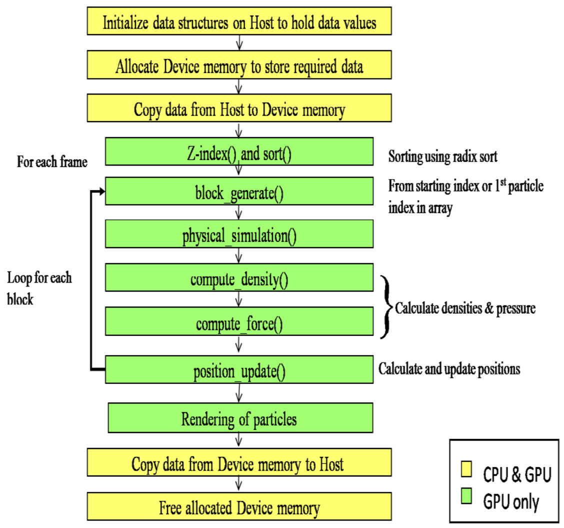

Initialize data structures on Host to hold data values
Allocate Device memory to store required data
Copy data from Host to Device memory
For each frame
Z-index() and sort()
Sorting using radix sort
block_generate()
From starting index or 1st particle index in array
Loop for each block
physical_simulation()
compute_density()
compute_force()
Calculate densities & pressure
position_update()
Calculate and update positions
Rendering of particles
Copy data from Device memory to Host
Free allocated Device memory
CPU & GPU
GPU only

Fig. (2) Phases of GPU accelerated SPH Simulation

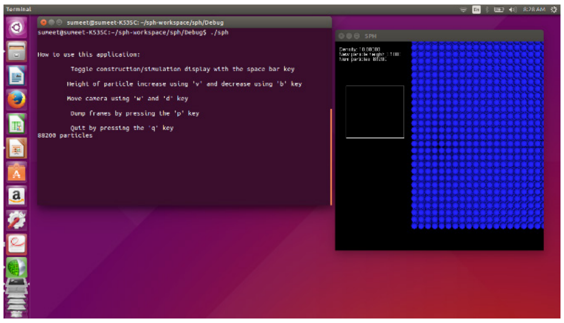
All above phases start assuming particles data has been copied to GPU. As the system is going to use the shared memory and the considered data is particle data, there is need of grid for shared memory access. It has been achieved with help of some lines of code.

RESULT DISCUSSION

The primary goal of the work was to bring out parallel programming in SPH applications. In order to reduce data traffic to the main memory, it is required to place all the neighboring particle's data in GPU's shared memory let the particles in the center grid's particles access the shared-memory. Additionally, it is important to keep the memory offset in such a way that each center grid particle could access the correct shared memory offset.

The data in shared memory is compared to the second particle data in the global memory. So the global memory access is reduced. In proposed work, this has been achieved by minimizing CUDAmalloc(), CUDAmemcopy reducing or removing the rendering copy process. To reduce the computation workload, some parameters has changed such as using smaller smooth-radius. Considering all these points, the concept of proposed work has been implemented.

Fig (3) shows the application window after execution. It provides options to user to add more particles, increase or decrease height of particle. Clicking on white window spawn new particles to the grid. User can also increase or decrease elevation of camera as shown in Fig (4).



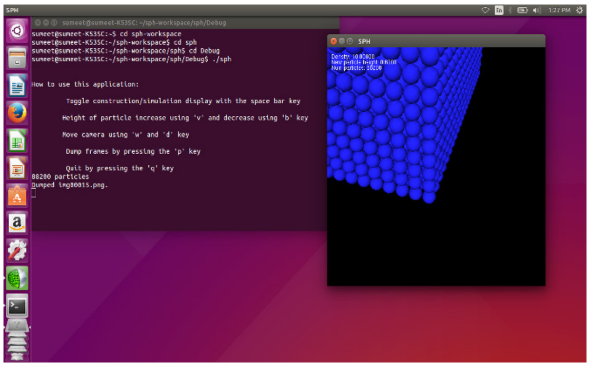
```
sumeet@sumeet-K53SC:~/sph-workspace/sph/Debug$ ./sph


How to use this application:

        Toggle construction/simulation display with the space bar key

       Height of particle increase using 'v' and decrease using 'b' key

       Move camera using 'w' and 'd' key

        Dump frames by pressing the 'p' key

        Quit by pressing the 'q' key
88208 particles
```

Fig. (3) Output window

The keys u, h, j, k have provided some functions like moving camera, increasing or decreasing the elevation. This is useful because for testing accuracy of particles all the constraints should be taken into consideration. The particles density can be increased using v key and decreased by using b key.

Volume-5, Issue-9, September - 2016 • ISSN No 2277 - 8160

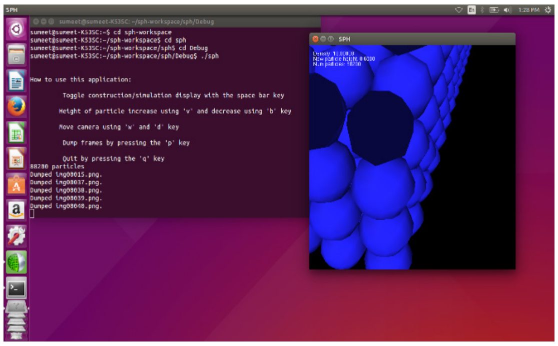
The dumped frames of simulation can be stored for offline reading. The default path of storing these frames is Debug. The dumping of frames can be started by pressing p key. The use of shared memory and multiple neighbor finding algorithm used in this work applicably provided the speedy routine.



```
sumeet@sumeet-K53SC:~$ cd sph-workspace
sumeet@sumeet-K53SC:~/sph-workspace$ cd sph
sumeet@sumeet-K53SC:~/sph-workspace/sph$ cd Debug
sumeet@sumeet-K53SC:~/sph-workspace/sph/Debug$ ./sph


How to use this application:

         Toggle construction/simulation display with the space bar key

        Height of particle increase using 'v' and decrease using 'b' key

        Move camera using 'w' and 'd' key

         Dump frames by pressing the 'p' key

         Quit by pressing the 'q' key
08200 particles
Dumped img00015.png.
```

Fig. (4) Camera and Particle movement after increasing elevation

Some of the tests performed on ISPC (Intel's SPMD Program Compiler). These tests vary with CUDA and ISPC implementation.



```
sumeet@sumeet-K53SC:~$ cd sph-workspace
sumeet@sumeet-K53SC:~/sph-workspace$ cd sph
sumeet@sumeet-K53SC:~/sph-workspace/sph$ cd Debug
sumeet@sumeet-K53SC:~/sph-workspace/sph/Debug$ ./sph


How to use this application:

         Toggle construction/simulation display with the space bar key

        Height of particle increase using 'v' and decrease using 'b' key

        Move camera using 'w' and 'd' key

         Dump frames by pressing the 'p' key

         Quit by pressing the 'q' key
88200 particles
Dumped img08015.png.
Dumped img08037.png.
Dumped img08038.png.
Dumped img08039.png.
Dumped img08040.png.
```

Fig. (5) Particles after increasing width

Taking the particle count and considering pressure and density as first constraints, the graph is plotted in Fig (6). It shows the pressure and density of particles as the number increases.

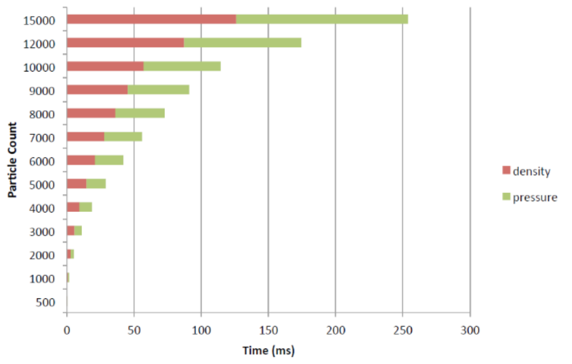


Fig. (6) Pressure and density change depending upon number of particles

CONCLUSIONS

The sequential implementation of SPH is having high overhead of CPU to GPU memory transfer. This requires much time for computation. The parallel implementation used in this work can overcome this drawback. Additionally, sequential implementation is having drawback of memory space wastage. This system gives the parallel implementation of SPH using shared memory at the cost of multiple physics frames per screen frame. In simple implementation, all particles data are copied to GPU. Then all the physics computations are performed, all data is copied back to the main memory, displayed and sent back to GPU. The use of shared memory saves much time of data transfer. Because the data from shared memory is sent to the main memory, only when it is needed.

The main constraint of this work was to bring out parallel programming in the SPH simulation using CUDA and OpenGL. This work gives a very good quality dumped images those can be used for offline reading. This work can be applicable in simulation based applications such as in video games, animations, movies and in Dualsphysics

REFERENCES:

[1] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler and Renato Pajarola: "Interactive SPH simulation and Rendering on the GPU", Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010), pp. 110M. Otaduy and Z. Popovic (Editors).

[2] Harada T., Koshizuka S., Kawaguchi Y.: "Smoothed particle hydrodynamics on GPUs", In Proceedings Computer Graphics International (2007), pp. 6370.

[3] Amenta N., Kil Y. J.: "Defining point-set surfaces",. ACM Transactions on Graphics 23, 3 (2004), 264270.

[4] Adams B., Pauly M., Keiser R., Guibas L. J.: "Adaptively sampled particle fluids". ACM Transactions on Graphics 26, 3 (July 2007), 4854.

[5] Dyken C., Ziegler G., Theobalt C., Seidel H.-P.: "GPU marching cubes on shader model 3.0 and 4.0", Research Report MPI-I-2007-4-006, Max-Planck-Institut fr Informatik, August 2007.

[6] Guennebaud G., Gross M.: "Algebraic point set surfaces", ACM Transactions on Graphics 26, 3 (2007),23.

[7] Harada T., Koshizuka S., Kawaguchi Y.: "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. 5562.

[8] Iwasaki K., Dobashi Y., Yoshimoto F., Nishita T.: "GPU-based rendering of point-sampled water surfaces", The Visual Computer 24, 2 (2008), 7784.

[9] Muller M., Charypar D., Gross M.: "Particle based fluid simulation for interactive applications", In Proceedings Eurographics/ACM Symposium on Computer Animation (2003), pp. 154159.

[10] Monaghan J.: "Smoothed particle hydrodynamics", Annu. Rev. Astron. Astrophys. 30 (1992), 543574.

[11] Zhang Y., Solenthaler B., Pajarola R.: "Adaptive sampling and rendering of fluids on the GPU", In Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics (2008), pp. 137146.

[12] Dr.Sauro Maneti, "A Smoothed Particle Hydrodynamics:Basics and Applications", Gioved 12, Novembre,2009.