Simulations of Large Particle Systems in Real Time

Witold Alda[1] **and Krzysztof Boryczko**[1]

[1]*AGH University of Science and Technology*
al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail:{alda, boryczko}@agh.edu.pl

Abstract

Simulation of interacting particle systems has been a well established method for many years now. Such systems can span different scales, including microscopic (where particles represent atoms, as in Molecular Dynamics simulations) as well as macroscopic. In the latter case, growing interest is put into Smoothed Particle Hydrodynamics approach. Traditionally, over many years, simulation of particle systems of a size big enough to give reasonable physical results, have been treated as the most time consuming calculations. As such, they have been run in an off-line mode, even in largest computer centers, where the batch mode work was a general rule. Introduction of massively parallel graphics processors (GPUs) gave an incredible impact not only to rendering performance, but also to the speed of numerical calculations, available also on a desktop computer or a local cluster. This enabled very fast (real-time or near real-time) simulations on a personal or near personal hardware for relatively large systems. It also encouraged to make the simulations interactive, with parameters adjusted on-line as well as to use on-line interactive visualization. The price for the speed is the necessity of designing new data structures and new algorithms which are vital for efficient use of massively parallel GPUs. Thanks to available software: Nvidia's CUDA and more general OpenCL, GPU programming became easier, however still needs a significant effort.

Keywords*: Smoothed Particle Hydrodynamics (SPH), Molecular Dynamics (MD), GPU architecture, GPU programming, CUDA, OpenCL, Scientific Visualization*

1. Introduction

Putting 'real-time simulation' expression in the title is somehow provocative. Obviously in atomistic simulations using molecular dynamics (MD), where the entire simulation time does not exceed 10^{-6} sec, one cannot speak about real time in a literary sense. Even in macroscopic simulations, such as Smooth Particle Hydrodynamics (SPH), exact real-time can be hardly achieved, depending on the number of particles and available computer power. Our understanding is more figurative. We think more about the possibility of making simulations using large number of particles (10^6 or more) with computers available locally (powerful desktops, local clusters) in an iterative rate. We think that having the possibility of observing the simulation on-line, changing the parameters and stepping backward, when it's needed, is of great importance to the idea of computer experiments. Recently we can achieve it much easier thanks to the extreme progress in the available power of Graphics Processing Units (GPUs). In the paper we underline the benefits we gather from the usage of GPUs, as well as the algorithmic problems we have to cope with.

2. Scope range of particle simulations

Among particle methods, SPH seems to evolve most rapidly in many fields of engineering. It is also the only method suitable for macro-scale and near real-time simulations. Originally developed for astrophysical problems, recently covered a vast area of applications. Most common is simulation of fluids in complex environment e.g. in porous media [1]. We can also point a fluid-solid modeling e.g. in cardiovascular vessels [2], simulation of solids [3] and especially multi-physics simulations [4] with fluid flows, sedimentation processes and seabed erosion coupled together.

3. Smoothed Particle Hydrodynamics Model Formulation

SPH is a meshless method, with particles used to carry the simulated field quantities. In SPH, any field quantity can be approximated using a weighted sum:

$$A(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h) \ , \quad (1)$$

where A is the scalar quantity calculated at point r. For all neighbor particles j within a certain distance, the mass m_j , density ρ_j , field value Aj and weight function W (usually denoted a smoothing kernel) are used to calculate the value. The distance between two points decides how much influence the neighbor particle gives. Over a certain cut-off value the interaction is neglected. There are different kernel functions for certain cases and one can design their own. A kernel should be normalized:

$$\int_r W(r,h)dr = 1 \quad . \quad (2)$$

Each particle represents a small part of the volume of the simulated fluid. The mass of a particle is constant during the simulation, but the densities need to be calculated at every time step, according to equation (1):

$$\rho(r) = \sum_j m_j \frac{\rho_j}{\rho_j} W(r - r_j, h) \ . \quad (3)$$

4. GPU implementation of particle algorithms

4.1. Neighbor search implementation

The crucial part of every particle-based simulation method is the determination of interaction between particles. In case of SPH, a model with a fixed r_{cut} value often used. An effective approach is to distribute particles into a regular cell grid. Then, a set of neighbors for each particles has to be found to compute interactions. With the cell grid approach finding neighbor particles is a matter of scanning all adjacent cells to the one where a particle is located (total of 27 for a 3D simulation).

There are two general approaches to store particles in data structures: linked-lists or arrays. In the case of linked lists [5], for each cell there is a list of particles belonging to that cell. This is implemented as two arrays, *head* and *next*, each storing identifiers of particles. The second way is putting particles in a global sorted array, allowing for linear memory access while scanning a cell for interactions. The former approach has shown to be suitable only for smaller environments (up to about 10^5 particles), but doesn't scale well further. In most cases the latter one was more efficient.

4.2. Bucket sort

A useful feature that NVIDIA introduced in CUDA since GeForce 8800 GT, 8600 series and newer, are atomic operations on global memory. Such operations allow for concurrent updates to the global memory by several threads. In case of collision these requests are processed sequentially and each is guaranteed to be executed. One of the basic operations is *atomicExch* which puts a new value at a memory location, returning the previous one. It makes bucket sorting implementation straightforward, with the following computation kernel, executed for each element:

```
prev = atomicExch(head+mapping[id], id)
next[id] = prev
```

where *mapping* array maps particle *id* to a cell. Updating the *head* array needs to be synchronized as several particles may belong to the same cell. On the other hand *next* is modified only by a single particle so this assignment is safe.

The problem with this solution is that atomic operations are quite costly and decrease overall performance. However shared memory may be leveraged to compute as much as possible locally and then update the global memory. The bucket sort algorithm described here has been modified so that each thread blocks computes lists of elements for their data and then merges it with the remaining results:

```
__shared__ int s_mapping[THREAD_COUNT]
__shared__ int s_next[THREAD_COUNT]
bucket = mapping[id]
s_mapping[threadIdx.x] = bucket
s_next[threadIdx.x] = -1
__syncthreads()
is_head = true
for(int i = threadIdx.x - 1; i>= 0; i--)
  if(s_mapping[i] == bucket)
    other_id = blockIdx.x * blockDim.x+i
    next[other_id] = id
    s_next[i] = threadIdx.x
    is_head = false
    break
__syncthreads()
if(is_head)
  prev_id = atomicExch(head +bucket, id)
  first = threadIdx.x
  while(s_next[first] != -1)
    first = s_next[first]

next[blockIdx.x*blockDim.x+first]=prev_id
```

The main idea behind the algorithm is that for the local array each element finds it's preceding one, within the same cell. Also *s_head* flag is set for those element, that were the first in each cell. Such partial lists are added to the global memory as in the previous implementation. Additionally *s_mapping* is used to cache *mapping* array on local multi-processor.

4.3. Converting lists to arrays

To convert a set of linked lists the following steps need to be executed:

- compute the number of elements in each cell, by walking each linked list,
- finding the position of each cell's first particle, done by executing prefix-sum algorithm on the array with cell sizes,
- computing position of each particle by visiting elements of each lists and adding the position of the cell

As a result, for each particle a position is obtained. Then rearranging the particle array is a trivial operation. For comparision we have also used the radix sort method [xx] to produce particles arrays. Additionally, for each cell, the first particle has to be found.

GPUs appeared to be extremely effective for particle calculations. With new NVIDIA 500 series it became possible to simulate 10^6 particles in interactive mode.

5. Conclusions

Using modern graphics processors and carefully tuned algorithms, it is possible to perform interactive simulations with real-time visualization for at least 10^6 particles at a frame rate around 15 fps.

Acknowledgments:

Partial support by Ministry of Science and High Education grant N N519 443039 is kindly acknowledged.

References

[1] Holmes, D.W. , Williams J.R., Tilke, P., *Smooth particle hydrodynamics simulations of low Reynolds number flows through porous media The Finite Element Method, Int. J. Numer. Anal. Meth. Geomech.* 2011; **35**:419–437.

[2] Farahani, M.H., Amanifard, N., Hosseini, S.M, *A Fluid-Structure Interaction Simulation by Smoothed Particle Hydrodynamics*, Engineering Letters, ISSN 1816-093X, 16:3, 2009.

[3] Fuller, M.D. *The Application Of Smooth Particle Hydrodynamics To The Modelling Of Solid Materials*, PhD Thesis, University of Leicester, April 2010.

[4] Ulrich, C. and Rung, T., Multiphysics SPH for Harbour and Ocean Engineering Hydrodynamics, V European Conference on Computational Fluid Dynamics ECCOMAS CFD 2010 J. C. F. Pereira and A. Sequeira (Eds) Lisbon, Portugal,14-17 June 2010.

[5] Rożen, T., Boryczko, K. and Alda, W., *GPU bucket sort algorithm with applications to nearest-neighbour serach*, Journal of WSCG, Volume 16, Number 1-3, pages 161-167, February 2008.