1 Abstract

There is no study using multiple GPUs for particle-based simulation to the best of our knowledge although several researchers have been using a GPU. In this study, a particle-based simulation is parallelized on multiple GPUs. There are several challenges to accomplish it. For example, the simulation should not have serial computations that can be a bottleneck of the simulation. Particle methods cannot assign a fixed computation data to each GPU but the data has to be reassigned each iteration because particles not having any fixed connectivity can move freely. It is difficult for a particle-based simulation to scale the performance to the number of GPUs because the overhead of the parallelization can be high. We overcame these hurdles by employing not a server-client computation model but a peer-to-peer model among GPUs. Each GPU dynamically manages their own computation data without a server. A sliced-grid was used to lower the traffic among GPUs. As a result, the simulation speed scales well to the number of GPUs and the method brings it possible to simulate millions of particles in real-time. Of course, the proposed method is effective not only for a simulation on GPUs but also one on CPUs. The contribution of this study also includes a sorting technique utilizing the coherency between the time steps which was also introduced to increase the performance on a GPU.

2 Methods

When a particle-based simulation is parallelized on multiple GPUs, it is not obvious how to manage the data on them. The simplest computation model would be to put a processor managing the data of all the GPUs. However, we did not employ the computation model because the managing processor can be a bottleneck of parallelization. Instead, we employed a computation model in which all the GPUs manage their own data, respectively. The computational domain is divided into subdomains of the number of GPUs. Each GPU is responsible for the particles in the subdomain. Compared to grid-based simulations, particle-based simulations have to manage the data of each GPU dynamically because simulation entities move freely in the computational domain. Two kinds of particles have to be sent to neighboring GPUs. One is the particles get out from the subdomain of a GPU. The other is the particles located at the edge of the subdomain. The later is necessary to compute physical values of particles locating at the edge of the subdomain. We can send these particles when they are required but the send is not efficient because it accompanies with a lot of data transfer of small granularity. Thus, ghost regions were introduced to improve the efficiency of the transfer. By introducing it, these particles are packed in a buffer and sent once in iteration. Here, we have to care about the method to select these particles to be sent because it is an overhead of parallelization. We used the grid constructed to make the neighboring particle search efficient was used to select the data to be sent dynamically. The sliced grid was introduced to make the neighboring particle search efficient [Harada et al. 2007]. The sliced grid improves not only the memory efficiency but also computational efficiency by increasing the locality of the data stored in the grid. The sliced grid is also effective to our problem because it reduces the empty voxels and this results in smaller data to be transferred between GPUs. This paper also presents a block transition sort utilizing the frame coherency and thus suited for almost sorted lists. The block transition sort was introduced to a particle-based simulation to improve the computation speed further by increasing locality of the data.

3 Results

The program was written in C++ and CUDA[NVIDIA ] on a PC equipped with a GeForce 8800 GT and Tesla S870 (four C870 GPUs). Fig.1 shows a screenshot from a real-time simulation using 500,000 particles. They were simulated on 4 GPUs and the other GPU was used for rendering. The host invoked five threads on the CPU and each of them controls a GPU. An iteration of the simulation took about 30 ms. Several scenes were simulated on 1, 2 and 4 GPUs and the measured times are shown in Fig.1. The simulation times scale well to the number of GPUs. However, the efficiency slightly decreases on the total computational time which also includes times for communication between GPUs although the data size to be sent is optimized in our method. This is because of the data transfer. When using 2 GPUs, a GPU has to send data to another GPU. However, when 4 GPUs were used, 2 of 4 GPUs have to send data to two neighboring GPUs and it results in longer transfer time than simulations on 2 GPUs.

References
