Dynamic Load Balancing of Granular Particle Simulations on Multi-GPU Systems

Sascha Sauermann

18R50523

Germany

Prof. Dr. Takayuki Aoki

Department of Mechanical Engineering

1 Background

Granular materials not only occur in nature in form of sands, grains or powders, but they are also a part of many machines like printer toners or industrial processes in chemical plants or ore processing facilities. Thus, there is a large demand for computer simulations of those materials to understand certain material properties or improve manufacturing processes.

There is a need to simulate multiple millions of particles, which requires an enormous computational effort. As the percentage of particle that are simultaneously in contact is relatively small, memory access costs are greater than the costs of the floating-point operations required for computing those interactions.

This is one of the main differences between *Granular Particle Simulations* and other n-body problems like *Molecular Dynamics Simulations* where the computation of the interactions, which range further than physical contact, dominate the computational costs.

Utilizing current generation supercomputers that consists of multiple interconnected nodes with few CPUs (central processing unit) each, it is possible to simulate granular materials with realistic particle numbers. Installing additional accelerator cards like GPUs (graphics processing unit) into each node allow for great performance improvements, because they have a higher floating-point performance and wider memory bandwidth than CPUs. [4]

2 DEM for Granular Materials

Simulations of granular materials are done by utilizing the *Discrete Element Method* (DEM). The material is represented by single particles (e.g. spheres) whose movement is defined by Newton's equations of motion. Those ordinary differential equations (ODE) are then solved at discrete time steps by an integrator and forces between the particles are calculated in each step.

2.1 Time Integration

We use the second order Leapfrog scheme for the time integration, as it has a good numeric stability, is a symplectic integrator and has a minimal memory footprint. The new velocity v and position x of a particle are calculated with an offset of half a time step Δt from the old velocity, position and acceleration as follows: [3]

$$v^{t+\Delta t/2} = v^{t-\Delta t/2} + \Delta t \frac{F^t}{m} \tag{1a}$$

$$x^{t+\Delta t} = x^t + \Delta t \cdot v^{t+\Delta t/2} \tag{1b}$$

2.2 Force Model

We model the interaction of two granular particles by a linear spring and a damping force that is dependent on the penetration depth and the relative velocity between the particles, which is then applied in the normal direction. Additionally, friction in tangential direction can be taken into account. [1][4]

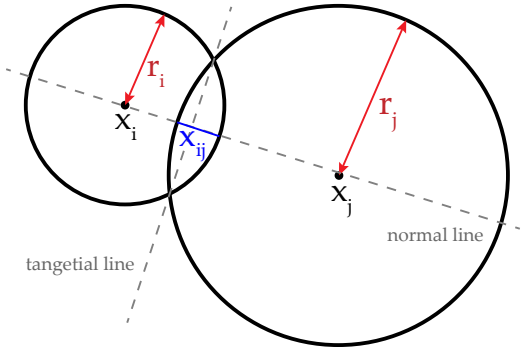

Figure 1: Spherical particle model for granular materials

The force between two intersecting particles i and j with positions x, velocities v and radii r is thus defined as

$$F_{i,j} = \begin{cases} N_{ij}(-k \cdot x_{ij} - \gamma \cdot \dot{x}_{ij}), & \text{if } x_{ij} > 0 \wedge i \neq j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

with the *spring coefficient* k, the *damping coefficient* γ and penetration depth x_{ij}. The relative velocity $\dot{x}_{ij}$ can be calculated from the absolute velocities v and the unit normal vector N_{ij}.

$$x_{ij} = r_i + r_j - \|x_j - x_i\| \quad \dot{x}_{ij} = (v_i - v_j) \cdot N_{ij} \quad N_{ij} = \frac{x_j - x_i}{\|x_j - x_i\|} \tag{3}$$

The total force on a particle i is then defined as the sum of all individual forces $F_i = \sum_j F_{ij}$.

2.3 Linked Cell Method

In our model two particles only apply forces to each other when they intersect. Thus, we would have to check this condition for every possible particle pair, leading to an computational complexity of $\mathcal{O}(n^2)$ for n particles.

To improve upon this, we divide the domain into cells with side lengths of at least r, where r is the radius (or circumsphere for non-spherical particles) of the largest particle. Then checking for intersection of particles in the same and in neighboring cells is enough (see Figure 2). As only a small finite number of particles fit in each cell and there are only 27 cells (in 3D) to check for each particle, the total complexity is reduced to $\mathcal{O}(n)$.

3 Dynamic Load Balancing

To utilize multiple GPUs, we want to distribute the particles evenly over all of them. For this we divide our simulation domain into subdomains. As the particles move over time, just applying a static domain decomposition at the start is not sufficient. Particles that leave one subdomain must be sent to the GPU that handles the area they now reside in. Over time, this can lead to an unbalanced particle distribution.

As the GPU memory is quite small, the number of particles that we can store on each GPU is limited. An imbalance does not only lead to idling GPUs and wasted resources, but the particle numbers on a GPU can exceed the available memory which poses huge problems. Shifting particles around between host and device memory greatly decreases the performance and should be avoided whenever possible. Therefore, it is important to have a dynamic domain decomposition that is updated whenever reasonable.

The approaches for dynamic load balancing can be divided into three different groups: spatial decomposition, load diffusion and graph-based methods [2]. We will now look at a selection of first group, because we already have a spatial decomposition in place with the linked cells algorithm we can reuse.

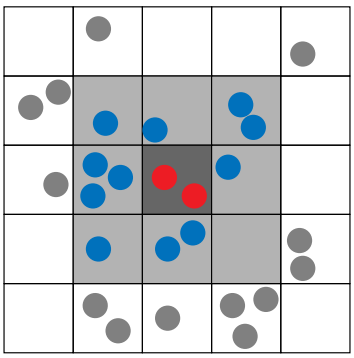

Figure 2: Linked cells method (2D): The red particles in the dark gray center cell must only be checked for intersections with the blue particles in the eight neighboring light gray cells and with every other red particle

3.1 Slice-Grid Method

The domain is divided into a grid with one entry per GPU. Balancing takes then place in multiple steps, once for each dimension d_i where only the k_i grid edges along this dimension are moved. First of all, each of the k_1 edges in the first dimension d_1 is moved in such a way, that the number of particles in each of the $k_1 + 1$ emerging subdomains is balanced. Now do the same for the second dimension d_2 for each of the $k_1 + 1$ subdomains individually to create $(k_1 + 1)(k_2 + 1)$ balanced subdomains. Continue this for the remaining dimensions. See Figure 3 for an example.

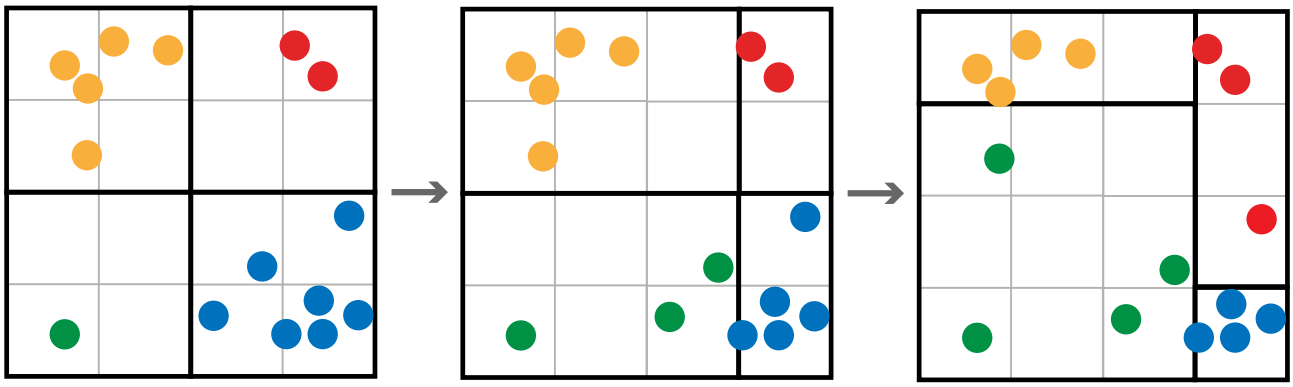

Figure 3: Slice-grid method for 2D cells with four subdomains: In the first step the grid is balanced adapting the edges in x dimension and in the second individually in the y dimension

Advantages of the slice-grid method are that each subdomain is cuboid and thus relatively simple to handle in the implementation and that the balancing can be performed completely on the GPU. Thus, there is no need to copy the particles back to the host memory, which reduces the time for the balancing immensely. [4]

3.2 Space-Filling Curves

A space-filling curve (SFC) is a continuous mapping of a multi-dimensional space to a one-dimensional space (often to the unit interval $[0, 1]$). One popular curve in computer science is the Hilbert curve as it is quite simple to implement and provides a good spatial locality (i.e. points that are close to each other in the original space are also close in the linearization) and a small communication distance. [2]

Using an SFC allows mapping all our cells to a linear list that retains this locality of close elements. Then we can split the list into parts that contain a balanced number of particles and thus get an assignment of our cells to the different subdomains. See Figure 4 for an example.

The main advantage of SFCs is the good spatial locality. This comes at a cost though. As the decomposition is dependent on the total state of the simulation, global communication is necessary. This can prevent this method from scaling above a certain amount of processes / nodes as shown by [2]. Their

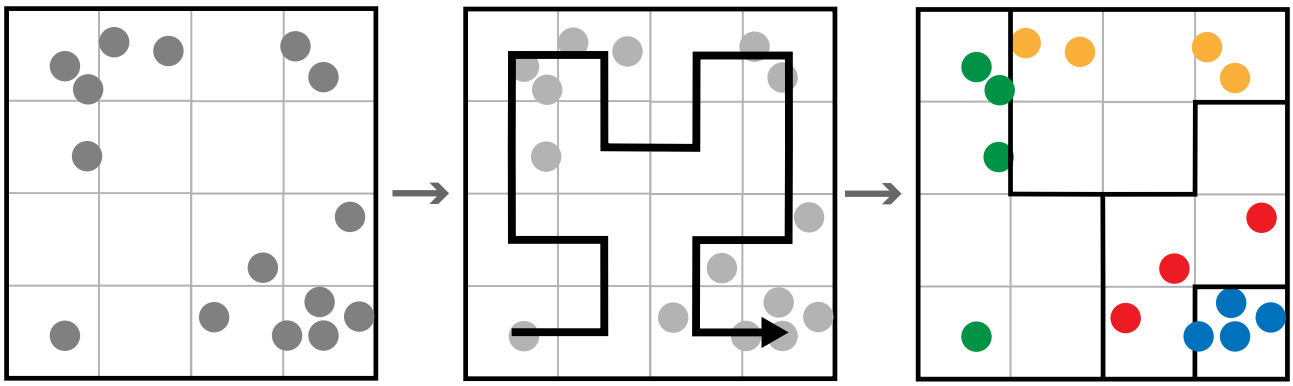

Figure 4: Space filling curves for 2D cells: The domain is split into four subdomains using a Hilbert curve

practical limit of 8192 nodes is way above the number of nodes even the TSUBAME3.0 supercomputer currently has (540, see Table 1) and therefore should not affect GPU computing in the near future.

4 Results

We implemented a *Granular Particle Simulation* in C++ using Cuda to run large parts of it on GPU. This includes creating Cuda kernels for e.g. the time integration and particle interactions, as well as adapting the vector class, force calculation, etc. to execute on the device. When using multiple GPUs, a Hilbert curve is used to decompose the domain for the dynamic load balancing.

The code is executed on a single node of the TSUBAME3.0 supercomputer (see Table 1) utilizing up to four Nvidia P100 GPUs. Using more GPUs would require the use of MPI for communication between multiple nodes. First results show already a massive performance increase from just using a single GPU compared to an OpenMP parallelized CPU-only version of the code.

CPUs	RAM	GPUs	Interconnect
Xeon E5-2680 V4 (14 × 2.4 GHz) × 2	256 GB	Tesla P100 (16GB HBM2) × 4	Intel Omni-Path × 4

Table 1: Specifications of one of the 540 nodes of the TSUBAME3.0, which is as of June 2019 the number 25 on the TOP500 list of supercomputers

5 Conclusion

Using GPUs for granular particle simulations offers huge performance advantages compared to CPU-only computations. Careful memory management is a necessity though, as the number of particles that fit onto the same GPU is limited. Dynamic load balancing is required to utilize multiple GPUs and further improve the simulation performance. There exist many different methods for dynamic load balancing on GPUs that have to be carefully evaluated and optimized for implementation on GPUs.

References

[1] N. Bell, Y. Yu, and P. J. Mucha. "Particle-based Simulation of Granular Materials." In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '05. Los Angeles, California: ACM, 2005, pp. 77–86. ISBN: 1-59593-198-8. DOI: 10.1145/1073368.1073379.

[2] S. Eibl and U. Rüde. "A Systematic Comparison of Dynamic Load Balancing Algorithms for Massively Parallel Rigid Particle Dynamics." In: *CoRR* abs/1808.00829 (2018). arXiv: 1808.00829.

[3] M. Griebel, S. Knapek, and G. Zumbusch. *Numerical simulation in molecular dynamics*. Springer Berlin, 2007.

[4] S. Tsuzuki, S. Watanabe, and T. Aoki. "Large-scale DEM Simulations for Granular Dynamics." In: *TSUBAME e-Science Journal* 13 (Mar. 2015), pp. 13–18. ISSN: 2185-6028.