UDC: 681.5.004 DOI 10.36910/775.24153966.2024.77.10

В. В. Сергєєв

Луцький національний технічний університет

МОДЕЛЮВАННЯ ТВЕРДИХ ЧАСТИНОК МЕТОДАМИ HOOMD-BLUE PYTHON ТА МОНТЕ-КАРЛО

Методи Монте-Карло виявилися цінними при моделюванні фізичних об'єктів і процесів. Мова програмування Python є основним інструментом, який широко використовується серед наукових дослідників. Метод, використаний у цій статті, задає алгоритмічну основу для моделювання засипки твердих частинок. Використаний варіант метода Монте-Карло (HPMC). представляє частинки як об'єкти різноманітної форми, що не можуть накладатися одна на другу. У системі немає сил. притягання або відштовхування Виключно жорсткі зв'язки призводять до впорядкованих структур. Запропонований фреймворк включає всі етапи моделювання: метамову для опису форми частинок, інтегратор опуклих многогранників, ініціалізацію стану системи, пробні ходи (варіанти проміжних станів), зберігання конфігурації у файловій системі.

Ключові слова: *моделювання, частинка, пітон монте-карло, структура, форма, мета, система.*

V. Serhieiev

SIMULATION OF THE HARD PARTICLES WITH HOOMD-BLUE PYTHON AND MONTE CARLO METHODS

Monte Carlo methods proved to be valuable in simulating physical objects and processes. The Python programming language is the mainstream tool used widely among scientific researchers. The method used in this article sets an algorithmic basis to the problem of hard particle Monte Carlo (HPMC) simulation. The HPMC represents particles as extended objects which are not allowed to overlap. There are no attractive or repulsive forces in the system. Purely hard interactions induce effective attractions between simulated particles which can lead to ordered structures. The proposed framework includes all steps for the simulation: meta language to describe particle shape, convex polyhedron integrator, system state initialization, trial moves, storing the configuration to the file system.

Keywords: *simulation, particle, structure, python monte carlo, framework, shape, meta, system.*

Formulation of the problem.

Hard particle simulations are arguably one of the most fundamental model systems in soft matter physics, and hence a common topic of simulation studies. The primary obstacle in these simulations is the computational requirements for the system with a waste amount of simulated particles. The complexity calculation for N-body simulations is stated in [1] as follows: Given initial positions and velocities of n particles that have pairwise force interactions, simulate the movement of these particles so as to determine the positions of the particles at a future time.

During recent years various automated frameworks have been developed for designing hardware accelerators on reconfigurable platforms [2]. Applications written using general-purpose programming languages (Python, R, Julia) or specialized computational environments like MATLAB utilizing such algorithms on large-size data sets require high-volume computation platforms [3]. The focus is on N-body interaction problems which have a wide range of applications spanning from astrophysics to molecular dynamics.

Another prospective approach is to utilize the capabilities of the quantum processing units or quantum computers. Quantum computing and quantum Monte Carlo (QMC) are respectively the most powerful quantum and classical computing methods for understanding many-body systems [4, 5]. Unfortunately current state-of-art near term quantum devices (NISQ era) are limited in computing power and error correction protocols; the quantum hardware capable of simulating practical-useful quantities of particles is not yet available for the wide audience.

HPMC and Newtonian event-chain Monte Carlo method.

This article uses the HOOMD-blue particle simulation engine to set an algorithmic basis and framework to the problem of hard particle Monte Carlo (HPMC) simulation. The HOOMD-blue package [6] is actively developed by the University of Michigan and available open source. HOOMD-blue is a Python package with a high performance C++/CUDA [7] backend that is built from the ground up for GPU [8] acceleration. The Python interface allows users to combine HOOMD-blue with other packages to create simulation and analysis workflows. The framework can execute simulations both on CPUs or GPUs. The library default installation supports NVIDIA GPUs and CPU support is always enabled.

Typical Monte Carlo simulations run more efficiently on GPUs for simulation sizes larger than a few thousand particles (1KB in memory for particle with modern hardware gives a ~100 000 000 soft limit for simulation), although this strongly depends on the details of the simulation:

- The complexity of the definition of particle shape.
- Spherical or cylindrical particles are easier to simulate from the computational cost while complex, unregular and more realistic particles require more computations to perform.
- Simulation step parameters.

HOOMD-blue engine implements Newtonian event-chain Monte Carlo (NEC) simulation method. In NEC a trial move is applied to a sequence and these collective moves form a chain which updates the configuration. Each update chain has a random start particle and is followed by the series of deterministic events. This is different in nature from the classical Monte Carlo method with local update. In classical trial moves a single randomly selected particle is either moved or rotated and new configuration is then accepted with Boltzmann factor. The Boltzmann factor collapses to an overlap check when simulating a system of hard particles. Newtonian event-chain NEC has best performance with a system of anisotropic particles and obeys the balance condition [9].

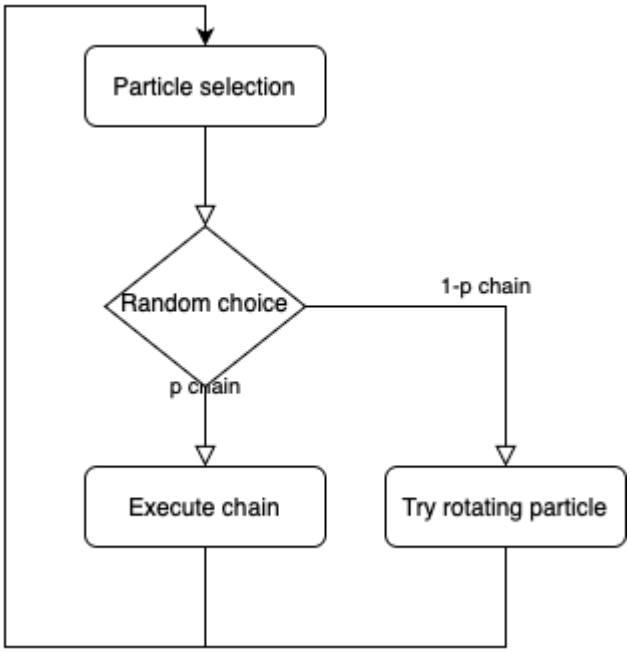

FIG. 1. **Flowchart of Monte Carlo simulation for anisotropic particles. The algorithm combines event chains for translation moves and random trial rotation moves as in local Monte Carlo**

The core object of the engine - is ConvexPolyhedron integrator which implements HPMC simulations:

```
mc = hoomd.hpmc.integrate.ConvexPolyhedron()
```

The particle in simulation is described as a set of vertices, normalized in range 0..1. The shape of the particle alone controls how it interacts with other particles. Formally, the potential energy of the system is zero when there are no overlaps and infinite when there are. Purely hard interactions induce effective attractions between particles which can lead to ordered structures. The example below illustrates the definition of Octahedron particle:

```
mc.shape['octahedron'] = dict(
vertices = [
(-0.5, 0, 0),
(0.5, 0, 0),
(0, -0.5, 0),
(0, 0.5, 0),
(0, 0, -0.5),
(0, 0, 0.5),
```

])

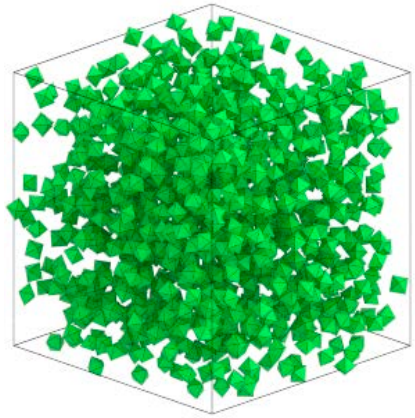

FIG. 2. **The initial random state of the simulation of octahedron particles**

The particle class in the example above, 'octahedron' acts as a descriptor for defined shape. The simulation can have [1..N] particle shapes and sizes. The next example uses the “Dodecahedron” particle shape.

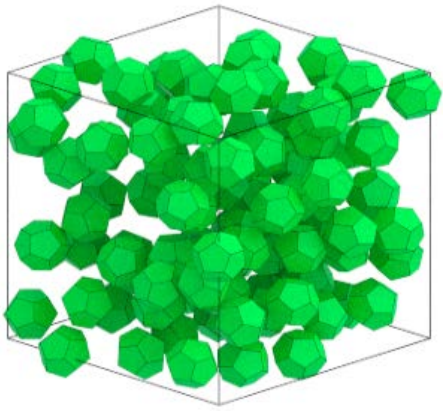

FIG. 3. **The initial random state of the simulation of dodecahedron particles**

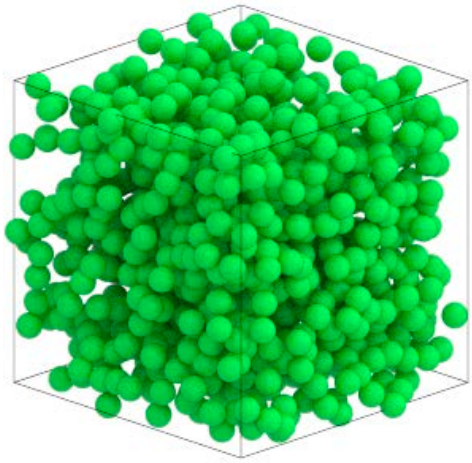

FIG. 4. **The initial random state of the simulation of spherical particles**

This shape structure is the standard array of the points (X, Y, Z) and could reuse existing public datasets for the complex particle shapes (construction materials, powder metallurgy, crystals).

We have analyzed the speed of simulation depending on the sphericity of the particles. The trend is that the higher sphericity results in the higher speed-up. This makes sense because higher sphericity makes translations less critical for equilibrating polyhedra and rotation moves number decreases (since it has no effect on spheres).

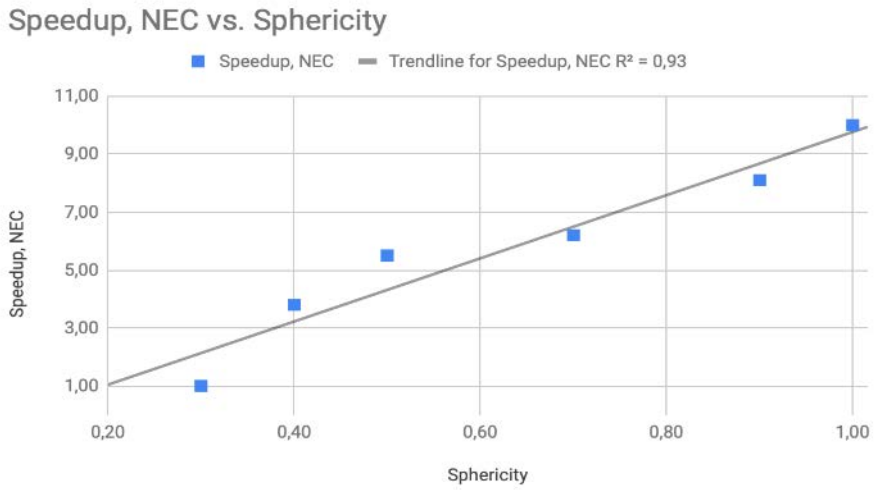

FIG. 5. **The speedup of the simulation with dependence on the sphericity of the particles**

During each time step in Monte Carlo the algorithm attempts to make a number of `nselect` trial moves on each particle in the simulation. The NEC method has two types of moves:

- Translation move - affects the position of the start particle, and then impacts all chains of particles.
- Rotation move - rotate the particle by a random angle about a random axis.

These parameters are properties of the integrator:

```
mc.nselect = 2
mc.d['octahedron'] = 0.15
mc.a['octahedron'] = 0.2
```

For the test simulation with ideal spherical particles the result with 10,000,000 steps gives the expected packing lattice:

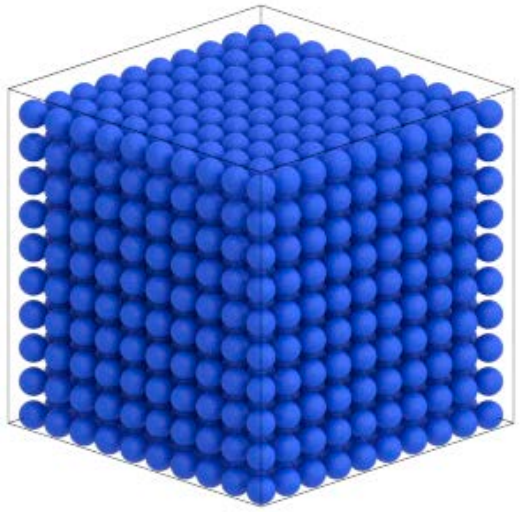

FIG. 6. **The convergence test for spherical simulation**

The HOOMD-blue integrator should be set to use either CPU or GPU. It is a type of operation and integrates the simulation state at every time step. Assigning the HPMC integrator is required to the simulation object:

```
cpu = hoomd.device.CPU()
simulation = hoomd.Simulation(device=cpu, seed=1)
simulation.operations.integrator = mc
```

The seed value (passed to the simulation constructor above) selects the sequence of values in the pseudorandom number stream. Given the same initial condition and seed, HPMC simulations will produce exactly the same results. All operations that generate pseudorandom numbers use the seed set in the

simulation. Whenever you add operations that utilize random numbers, you should set the seed to a non-default value.

HOOMD-blue does not adopt any particular real system of units. Instead, HOOMD-blue uses an internally self-consistent system of units and is compatible with many systems of units. For example: if you select the units of meter, Joule, and kilogram for length, energy and mass then the units of force will be Newtons and velocity will be meters/second. A popular system of units for nano-scale systems is nanometers, kilojoules/mol, and atomic mass units.

Run the simulation to randomize the particle positions and orientations. The run method takes the number of steps to run as an argument. 10,000 steps is enough to randomize a low density system:

simulation.run(10e3)

During each step the algorithm combines event chains for translation moves and random trial rotation moves as in local Monte Carlo. In the example the acceptance ratio (the fraction of attempted moves which are accepted) is very high since this is a low density simulation.

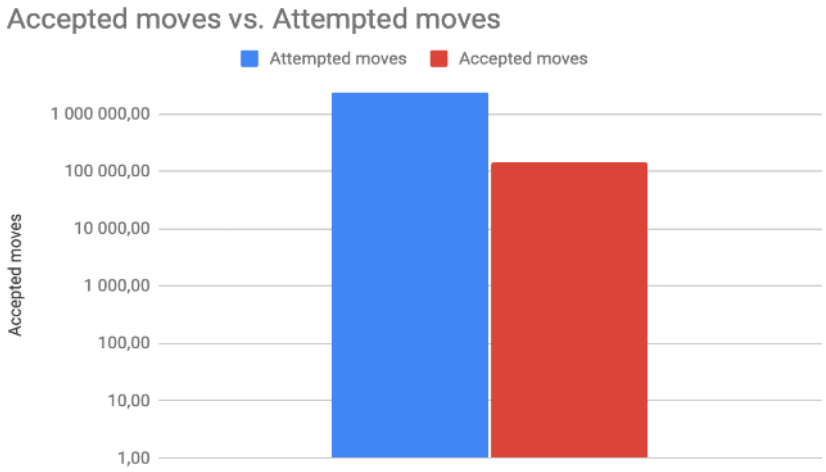

FIG. 7. **The accepted moves versus attempted moves ratio**

The resulting packed cluster of particles shown on the figure below.

FIG. 8. **The converged results visualization for octahedron simulation**

Conclusion.

The simulation algorithm uses the XenoSweep extension of XenoCollide [10] library for detecting collisions among a limitless variety of convex shapes. It is based on the technique “Minkowski Portal Refinement”. The method of simulation of the hard particles using HOOMD-blue is an important improvement both in terms of speed and in terms of simplicity of the algorithm. Such computation efficiency and simplicity in setup simulation is of interest to researchers working on a broad range of problems in computer graphics, robotics, and granular dynamics.

The great feature of the HPMC is that it automatically tunes simulation parameters to improve performance. During the first 1,000 - 20,000 timesteps of the simulation run, runtime will optimize kernel parameters each time it calls a kernel.

Future extensions should advance this method in two directions: improved handling of rotations and generalization to anisotropic particles with extended interaction, i.e. particles that are not purely hard. Next steps in the method improvement should focus on developing collective rotations moves that resemble conceptually the idea of event chains for translations. Such an improved algorithm has the potential to reach a speed-up of up to one order of magnitude across all convex polyhedra.

References:

1. Reif, John & Tate, Stephen. (1995). The Complexity of N-Body Simulation. 10.1007/3-540-56939-1_70.

2. Kim, Jungsub & Deng, Lanping & Mangalagiri, Prasanth & Irick, Kevin & Sobti, Kanwaldeep & Kandemir, Mahmut & Narayanan, Vijaykrishnan & Chakrabarti, Chaitali & Pitsianis, Nikos & Sun, Xiaobai. (2009). An Automated Framework for Accelerating Numerical Algorithms on Reconfigurable Platforms Using Algorithmic/Architectural Optimization. IEEE Trans. Computers. 58. 1654-1667. 10.1109/TC.2009.78.

3. Hamada, T. & Nakasato, Naohito. (2005). PGR: A software package for reconfigurable super-computing. Proc International Conference on Field Programmable Logic and Applications (FPL'05). 2005. 366- 373. 10.1109/FPL.2005.1515749.

4. Zhang, Yukun & Huang, Yifei & Sun, Jinzhao & Lv, Dingshun & Yuan, Xiao. (2022). Quantum Computing Quantum Monte Carlo. 10.48550/arXiv.2206.10431.

5. Mazzola, Guglielmo. (2023). Quantum computing for chemistry and physics applications from a Monte Carlo perspective.

6. Anderson, Joshua & Glaser, Jens & Glotzer, Sharon. (2019). HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. Computational Materials Science. 173. 109363. 10.1016/j.commatsci.2019.109363.

7. NVIDIA, Vingelmann, P. & Fitzek, F.H.P., 2020. CUDA, release: 10.2.89, Available at: https://developer.nvidia.com/cuda-toolkit.

8. Anderson, Joshua & Keys, Aaron & Nguyen, Trung & Glotzer, Sharon. (2010). HOOMD-blue, general-purpose many-body dynamics on the GPU.

9. M. Klement and M. Engel, " Efficient equilibration of hard spheres with Newtonian event chains," The Journal of Chemical Physics 150, 174108 (2019).

10.G. Snethen, "XenoCollide: Complex Collision Made Simple," in Game Programming Gems 7, edited by S. Jacobs (Charles River Media, 2008) pp. 165–178.