ITC 1/51
Information Technology and Control
Vol. 51 / No. 1 / 2022
pp. 48-58
DOI 10.5755/j01.itc.51.1.29455

Hash Functions and GPU Algorithm of Infinite Grid Method for Contact Search

Received 2021/07/13 | Accepted after revision 2022/01/25

http://dx.doi.org/10.5755/j01.itc.51.1.29455

HOW TO CITE: Pacevič, R., Kačeniauskas, A. (2022). Hash Functions and GPU Algorithm of Infinite Grid Method for Contact Search. *Information Technology and Control*, 51(1), 48-58. https://doi.org/10.5755/j01.itc.51.1.29455

Hash Functions and GPU Algorithm of Infinite Grid Method for Contact Search

R. Pacevič

Department of Graphical Systems; Vilnius Gediminas Technical University; Saulėtekio al. 11 , LT-10223, Vilnius, Lithuania; phone: +370 5 274 4913; e-mail: ruslan.pacevic@vilniustech.lt

A. Kačeniauskas

Department of Graphical Systems; Vilnius Gediminas Technical University; Saulėtekio al. 11 , LT-10223, Vilnius, Lithuania; Laboratory of Parallel Computing, Vilnius Gediminas Technical University; Saulėtekio al. 11 , LT-10223, Vilnius; phone: +370 5 274 4913; e-mail: arnas.kaceniauskas@vilniustech.lt

Corresponding author: arnas.kaceniauskas@vilniustech.lt

The paper presents the memory-saving GPU algorithm of the infinite grid method with various hash functions for contact search in discrete element computations. The implemented hash table of fixed size has the added benefit of allowing the grid to be potentially infinite in size, which is particularly suitable for a large number of discrete particles, moving in large computational domains with empty regions. Research on the application of various hash functions and hash table sizes was performed to preserve the computational performance of the developed memory-saving GPU algorithm and its OpenCL implementation. The performance of the developed software was evaluated by solving the hopper fill and discharge as well as the artificial avalanche problems on the NVIDIA® Tesla™ P100 GPU. The performance achieved by using the memory-saving implementation of contact search was compared with that attained by using the standard implementation of the uniform grid method. The performed analysis revealed that the developed GPU algorithm and its OpenCL implementation reduced the GPU memory consumed by the uniform grid method up to 69.7 times, which resulted in the contact search memory equal to 1.86% of the total memory required by DEM computations. Moreover, the application of the Morton hash function and the proper hash table size allowed for preserving high computational performance of the infinite grid method and the developed GPU software.

KEYWORDS: parallel computing, GPGPU computing, OpenCL, contact search, hash functions, infinite grid method, discrete element method.

1. Introduction

The discrete element method (DEM) suggested in the pioneering work of Cundall and Strack [3] presents numerical methodology providing a quantitative description of discrete particulate media. The key concepts of the DEM imply that the domain of interest is treated as an assemblage of rigid or deformable particles and that the contacts among them need to be identified and updated during the entire motion process. DEM has been applied to solve numerous problems related to granular flows [32], cohesive powders [31] and diluted aerosols [16]. However, the simulation of systems at the particle level of detail has the disadvantage of making DEM computationally very expensive. Naturally, to solve the industrial-scale problems, parallel computing has become an obvious option for significantly increasing computational capabilities. However, the selection of the efficient parallel solution algorithm is highly dependable on the specific features of the considered problem and the numerical method [14, 15, 27]. Compared to the CPU-based parallelization, GPU has a higher parallel structure, which makes them more efficient for particle-based algorithms, where large blocks of data can be processed in parallel. Software environments, such as CUDA or OpenCL [4], are targeted at general-purpose GPU (GPGPU) programming.

Usually, the computing time of industrial applications can be significantly reduced, using the parallelization benefits of GPU, which is becoming increasingly more important as an alternative computational platform for DEM simulations [9]. Shigeto and Sakai [26] have proposed a new algorithm for multi-thread parallel computation of DEM, pointing out that their calculation speed ratio of the GPU to CPU was up to 3.4 when single-precision floating-point numbers are used. Durand et al. [5] have performed GPU simulation of an impact on a reinforced concrete slab with 14274 particles, which is insufficient in the case of civil engineering structures. Yue et al. [35] have made a GPU version of Trubal code and demonstrated its application to die filling. In 3D simulations, containing 20000 particles, an average speedup of 19.66 was achieved on the NVIDIA Tesla K40c card. Govender et al. [9] have designed the modular high performance Blaze-DEMGPU framework for the GPU architecture and investigated the influence of shape non-uniformity and polydispersity of polyhedral particles on hopper discharge. Kelly et al. [18] have adopted an adimensionalization process combined with mixed-precision data to simulate 3D scenarios with up to 710 million spherical frictionless particles.

To achieve a higher speedup ratio for a larger number of particles, a few efforts were made to use the combined GPU and MPI technology. Xu et al. [33] have achieved the quasi-real-time simulation of an industrial rotating drum, when about $9.6 \cdot 10^6$ particles were treated with 270 GPUs. The one-dimensional domain decomposition with multiple GPUs has been applied to the simulation of 128 million particles by Tian et al. [29]. GPU-based DEM combined with MPI has been applied by Gan et al. [7] to study flow in different granular handling and processing systems related to the ironmaking industry.

Contact search is a very important and time-consuming part of DEM computations of granular materials [34]. Spatial partitioning techniques subdivide domain into particular regions, such as cells. Since two particles can only be in contact if they are in the neighboring regions, the number of pairwise contact checks is significantly decreased [22]. The uniform grid method (UGM) is the efficient space subdivision scheme often used to speed up the contact search queries on GPU in the case of nearly monosized particles. A typical grid representation stores a list of particles contained by each cell. However, in the cases of large grids, the mere storing of the lists can exceed limits of available GPU memory [2]. Thus, large uniform grids only partially filled with particles inefficiently allocate memory, and contact search can violate the limits of GPU resources, which restricts the application of DEM in industry.

Most of the authors [9, 29, 35] have performed standard implementations of the UGM for contact search or its minor modifications [26, 33]. Kalojanov and Slusallek [17] have developed the first GPU-based parallel algorithm of uniform grid construction for ray tracing. NVIDIA SDK provides a sample code [10] for simulating simple particle interactions by using a sorting algorithm and a uniform grid structure. Xu et al. [33] have used atomic functions to register particles to cells and the thread block per cell to perform the neighbour search. Durand et al. [5] have used a dynamic bounding box to reduce the number of cells in the uniform grid. Cai et al. [1] have modified cell-

linked list method using dynamic mesh for better utilization of CPU memory. Zheng et al. [36] have applied a complex and costly contact search algorithm, allowing a particle to overlap an arbitrary number of cells, to the self-compacting concrete flow simulations. The developed algorithm performed well up to 140063 heterogeneous particles in a simple case of the gravity packing problem. However, only 29580 particles were employed to simulate a complex self-compacting concrete flow. Gan et al. [8] have applied the hierarchical grids method for large-scale polydispersed particulate systems and noticed the decrease in efficiency caused by the load balance problem and the time-consuming MPI communication between GPUs on different nodes.

The main drawback of the UGM is that too much memory as buckets are allocated with predefined capacity. In a typical DEM simulation, many of these buckets are not used. Hence, the occupied GPU memory cannot be used for any other purpose. It seems that mapping of cells into a hash table of a fixed size requires less memory than storing the grid in a dense array [23]. Moreover, positions of particles are mapped by the hash function into a finite number of cells. The size of used storage is associated with the number of particles. Memory necessary for the hash table does not depend on the number of grid cells, therefore, the grid can be potentially infinite in size. However, memory usage and performance of the GPU code might depend on the considered solution domain and dynamics of particles, as well as the applied hash function and the hash table size. Moreover, the performance of contact search algorithms on large or complex computational domains with unfilled regions, covered by large number of grid cells, has been rarely investigated in the frame of the DEM simulations on GPU.

The paper presents performance analysis of the developed memory-saving GPU algorithm of the infinite grid method (IGM) with various hash functions for contact search of DEM simulations in computational domains that can be covered only by a large number of uniform grid cells. The other parts of the paper are organized as follows: Section 2 describes the implemented DEM model and used software, Section 3 presents the developed GPU algorithm for contact search, Section 4 describes the solved applications, Section 5 provides the performance analysis, while Section 6 gives the concluding remarks.

2. The DEM Model and Software

In the present work, the employed DEM software models the non-cohesive frictional visco-elastic particle systems. The dynamic behaviour of a discrete system is described by considering the motion and deformation of the interacting individual particles in the frame of Newtonian mechanics. An arbitrary particle is characterized by three translational and three rotational degrees of freedom. The forces acting on the particle may be classified into the forces induced by the external fields and the contact forces between the contacting particles. In the present work, the electromagnetic force [30], the aerodynamic force [16] and other external forces [20], except for the gravity force, are not considered. The normal contact force can be expressed as the sum of the elastic and viscous components. In the present work, the normal elastic force is computed according to the Hertz's contact model. The viscous counterpart of the contact force linearly depends on the relative velocity of the particles at the contact point. The tangential contact force is divided into the parts of static friction and dynamic friction. The dynamic friction force is directly proportional to the normal component of the contact force. The static friction force is calculated by summing up the elastic counterpart and the viscous damping counterpart. It is worth noting that the elastic counterpart includes the length of the tangential displacement, which depends on the time history. Thus, the considered friction model is incremental or time history-dependent, which requires storing the values of the tangential displacement during the contact between the neighbouring particles in the memory. The details of the applied DEM model can be found in [6, 11].

The DEM code is developed by using OpenCL [4] to run the same software on GPU and CPU of different vendors. In the present research, the main attention is focused on contact search because it can take a large part of the computing time and the consumed memory of granular material simulations on GPU. Therefore, at the beginning the whole DEM algorithm is only briefly outlined for completeness. The developed DEM algorithm for shared-memory architectures can be described as follows. At the start of the simulation, preprocessing is performed on CPU and the initial data are copied into the GPU memory. No further memory transactions between the CPU and

GPU memory are required, except for the result storage. All GPU kernels run thread per particle, which takes advantage of the massive parallel computation capabilities of modern GPUs and can be considered to be the most suitable parallelism in the case of DEM computations. The initial kernel performs the predictor step according to the Gear predictor-corrector method, starting the time integration. The several following kernels run contact search, therefore, they are circumstantially outlined below. The other kernels handle the time history of contacts, compute the contact forces and evaluate the external forces. The last kernel completes the time integration by performing the Gear corrector step. The values of positions, velocities and accelerations of the particles are corrected. At the end of the time step, the particle data can be copied from the GPU memory to CPU and stored on the hard disk drive in HDF5 format. It is recommended to transfer the data to the CPU memory as seldom as possible because it is a time-consuming process.

3. The GPU Algorithm of Contact Search

In fact, a very effective space subdivision scheme is a regular grid, which is often applied to perform fast contact search. However, a serious drawback of the UGM is that too much memory as buckets are allocated for a large number of uniform grid cells. To avoid storing the grid by using a dense array and to save the GPU memory each cell of the grid or the particles located there can be mapped into a hash table of a fixed set of M buckets. The main information is stored in two arrays (Figure 1), containing the list of pairs PARTICLE_ID(N) and PARTICLE_HASH(N), where N is the number of particles. The information for accessing the particles associated with a particular bucket is stored in two additional arrays BUCKET_START(M) and BUCKET_END(M), where M is the considered size of the hash table. The output of the contact search algorithm is stored in the arrays CONTACT_N(N) and CONTACT_IDS(K), where K is the number of particles N multiplied by the maximal number of contacts. In the case of packed monosized particles, the maximal number of contacts equals 12. These arrays are necessary for computing the contact forces in the relevant kernels.

Figure 1
A scheme of the developed GPU algorithm of the infinite grid method

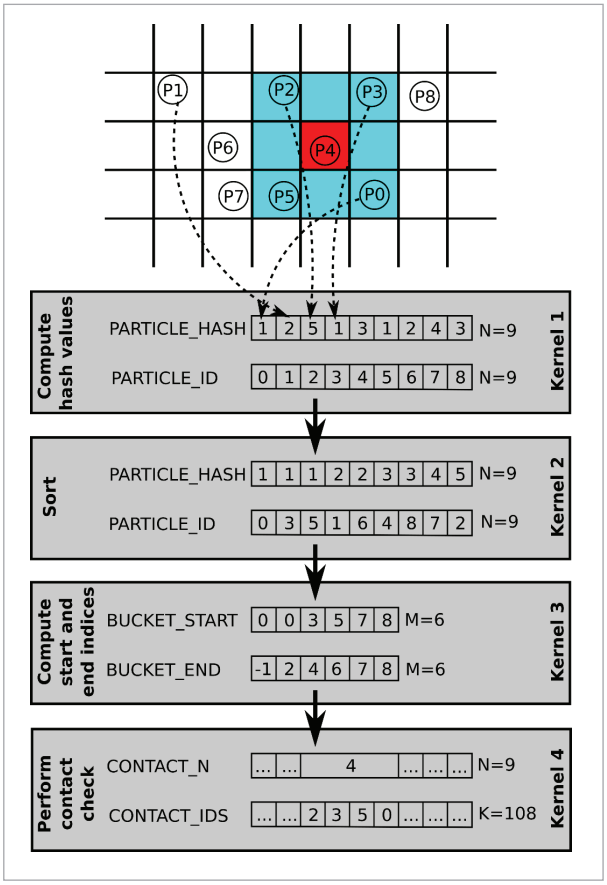


Figure 1 shows the scheme of the developed memory-saving GPU algorithm for the IGM. Kernel 1 runs a thread per particle and calculates hash values by using the considered hash function. In Figure 1, the values of the array PARTICLE_HASH are not calculated by any real hash function and only of an illustrative character. Kernel 1 stores the results to arrays PARTICLE_ID and PARTICLE_HASH in the global GPU memory. Kernel 2 sorts the particles according to their hash values. The sorting is performed by using the fast radix sort algorithm [25] provided by the Boost Compute library [21]. Kernel 3 fills the arrays BUCKET_START and BUCKET_END. The kernel compares the hash value of the current particle with the hash value of the previous particle in the sorted array PARTICLE_HASH. If the hash value is different, this indicates the end of the old bucket and the

start of a new bucket. The end index and the start index are written to the arrays BUCKET_END and BUCKET_START, respectively. In the array BUCKET_END, minus one indicates that this bucket has no particles and the relevant loop through the particles will not be performed. This kernel also performs computations on a thread per particle basis.

Kernel 4 performs the final contact check, which sometimes is referred to as the narrow phase of contact search. The contact check of the particular particle is performed against the particles located only in the neighbouring grid cells. Thus, for each particle, hash values of 27 neighbouring cells are calculated by using the considered hash function. It is worth noting that, contrary to the UGM, some hash values of the neighbouring cells might be the same and should be removed to avoid the duplicated contact. Then, the outer loop is performed over the determined buckets, while the inner loop is run over the particles of the current bucket. These neighbouring particles are accessed by using the indices to the array PARTICLE_ID that are stored in arrays BUCKET_START and BUCKET_END. When the distance between the checked particles is smaller than the sum of the radii of the particles, the contact is identified. The number of the contacting particles is increased in the array CONTACT_N, while the particle Id is added to the contact list CONTACT_IDS. The arrays CONTACT_N and CONTACT_IDS serve as the output of Kernel 4 and the whole contact search algorithm.

The occupancy of the hash table and the number of potential contacts is highly dependent on the employed hash function, which can have a great influence on the required memory and computational performance of the contact search algorithm. The uniform function [28], the Morton function [2] and the universal function based on the primary numbers [19] are considered for wrapping the world coordinates into a finite number of buckets. Thus, three different hash functions are considered to investigate the influence of the function used to the computational performance of contact search in the case of problem-specific DEM simulations.

4. The Considered Applications

Three different problems are considered to evaluate the computational performance of the developed GPU implementation of contact search for problem-specific DEM simulations. The hopper fill problem has a relatively small and simple solution domain, therefore, it can be efficiently solved by using the UGM for contact search. The solution domains of hopper discharge and artificial avalanche problems, on the contrary, have a relatively complex shape, involving several regions that require a large number of uniform grid cells. Moreover, in each time instance, the moving particles occupy only a small part of the computational domain, leaving empty regions and limiting efficiency of standard contact search implementations.

In the case of the hopper fill and the hopper discharge problems, the pyramid-shaped hopper of the same geometry is considered. The dimensions for the hopper geometry are as follows: height 1.0m, width 1.0m, thickness 1.0m and the height of the pyramidal part 0.4m. The width of the orifice is considered to be 0.5m to make a faster discharge. At the initial time instant of the hopper fill problem, cubically packed particles are placed in the cubic part of the hopper. In the case of the hopper discharge problem, a large rectangular box, imitating the infinite plane at the bottom, has the following inward dimensions: height 0.5m, width 8.0m and thickness 8.0m.

In the initial state of the hopper discharge problem, the particulate material is obtained from the numerical solution of the hopper fill problem at t=0.5s. Granular material is presented by the assemblies of 117649, 970299, 1906624, 2863288 and 3723875 particles with the radii equal to 0.0100m, 0.0050m, 0.0040m, 0.0035m and 0.0032m, respectively. The material has the bulk particle density of 1290kg/m^3. The other physical data for particles are as follows: the Poisson's ratio is equal to 0.2, the particle elasticity modulus is equal to $2.36 \cdot 10^8$Pa, the friction coefficient between the particles is equal to 0.4, while the coefficient of restitution is equal to 0.5. The coefficients of the walls are considered to be as follows: the Poisson's ratio is 0.35, the elasticity modulus is 2.6310^{9}Pa and the friction coefficient between walls and particles is 0.4.

The avalanche flow is considered for DEM analysis of damping effects. This analysis resulted in large numbers of particles, moving in the partially filled computational domain covered by a large number of grid cells, which is a very suitable test case for the developed memory-saving GPU algorithm. The geometry of the considered avalanche problem is given in

Figure 2
The avalanche flow at t=17s: (a) 108640 particles with R=0.037m; (b) 1649520 particles with R=0.015m

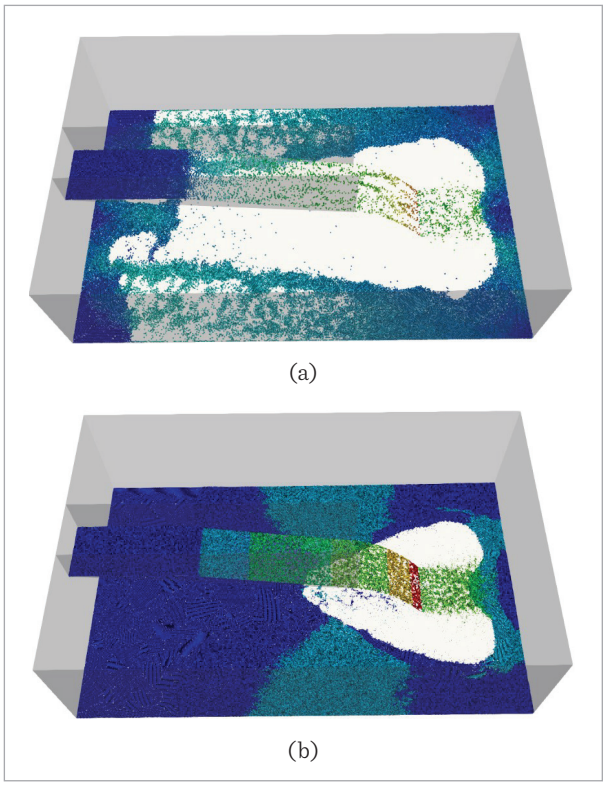


Figure 2. The main rectangular box has the following inward dimensions: height 9.5m, width 20.0m and length 32.0m. The cuboid of the cubically packed particles falls to the channel with an inclination of 10 degrees. The dimensions of the channel are as follows: width 4.0m, length 12.0m and the height of walls is equal to 5.5m. The left end and the right end of the channel are raised above the bottom of the rectangular box by 6.0m and 4.0m, respectively.

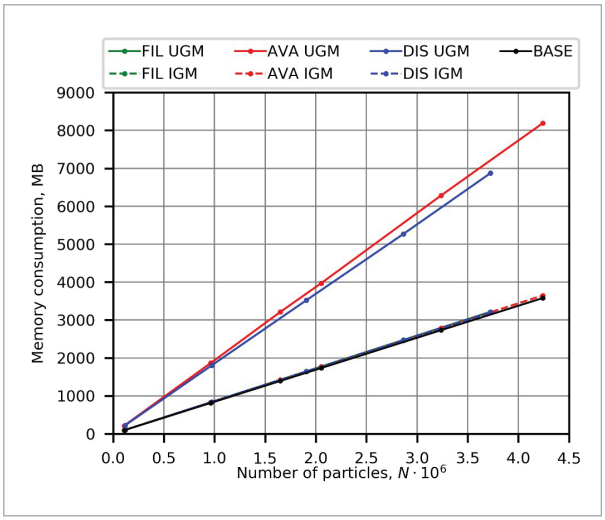
In the case of the avalanche problem, the granular material is presented by the assemblies of 108640, 963600, 1649520, 2053430, 3237300 and 4241517 particles with the radii equal to 0.037m, 0.018m, 0.015m, 0.014m, 0.012m and 0.011m, respectively. The material has the bulk particle density of 920kg/m^3. The other physical data for the particles are as follows: the Poisson's ratio equals 0.3, particle elasticity modulus is $9.33 \cdot 10^6$Pa, the friction coefficient between the particles equals 0.05 and the coefficient of restitution equals 0.5. Figure 2 shows the avalanche flow visualized by ParaView software [12]. The particles are coloured according to the magnitude of the particle's velocity. It can be observed that the results of the avalanche flow simulations are highly dependable on the number of particles.

4. The Performance Analysis

The considered benchmark problems were solved to evaluate memory consumption and computational performance of the developed memory-saving GPU implementation of contact search. In the frame of the IGM, the application of different hash functions was investigated, and the results were compared to those obtained by using the standard UGM. It is worth noting that the UGM is efficient in the case of monosized and nearly monosized particles. A large heterogeneity ratio can reduce the performance of the contact search based on the UGM. Thus, only monosized particles were considered to perform a correct performance comparison between the UGM and the developed GPU implementation of the IGM in the case of all the considered problems. All double-precision computations were performed on the NVIDIA® Tesla™ P100 GPU Computing Accelerator (56 Streaming Multiprocessors, 1792 FP64 CUDA Cores, 12GB HBM2, 549GB/s memory bandwidth), which was installed on the workstation with the hardware characteristics as follows: Intel®Xeon™ E5-2630 2.20GHz 2xCPU, 32GB DDR4 2133MHz RAM. The simulation and visualization were performed on the computational infrastructure [13, 24] hosted by Vilnius Gediminas Technical University.

Figure 3 shows GPU memory consumption of the developed DEM code, solving the hopper fill problem (the abbreviation FIL), the hopper discharge problem (the abbreviation DIS) and the artificial avalanche problem (the abbreviation AVA). The curves FIL UGM, DIS UGM and AVA UGM represent memory consumption of the code, using the UGM for contact search, while the curves FIL IGM, DIS IGM and AVA IGM represent the memory consumed by the developed implementation of the IGM with the table size equal to the number of particles. It is obvious that the developed implementation of the IGM consumed a very small amount of memory compared to other

Figure 3
GPU memory consumption for DEM computations



DEM procedures represented by the BASE curve for the artificial avalanche problem. The IGM consumed only 59.6MB, 59.6MB and 67.9MB in the cases of the hopper fill problem with 3723875 particles, the hopper discharge problem with 3723875 particles and the artificial avalanche problem with 4241517 particles, respectively, which was up to 1.9% of the total memory required by DEM computations. Therefore, it was difficult to observe the difference between the BASE curve and the curves FIL IGM, DIS IGM and AVA IGM, representing the IGM. Moreover, it was hardly possible to distinguish these curves from each other.

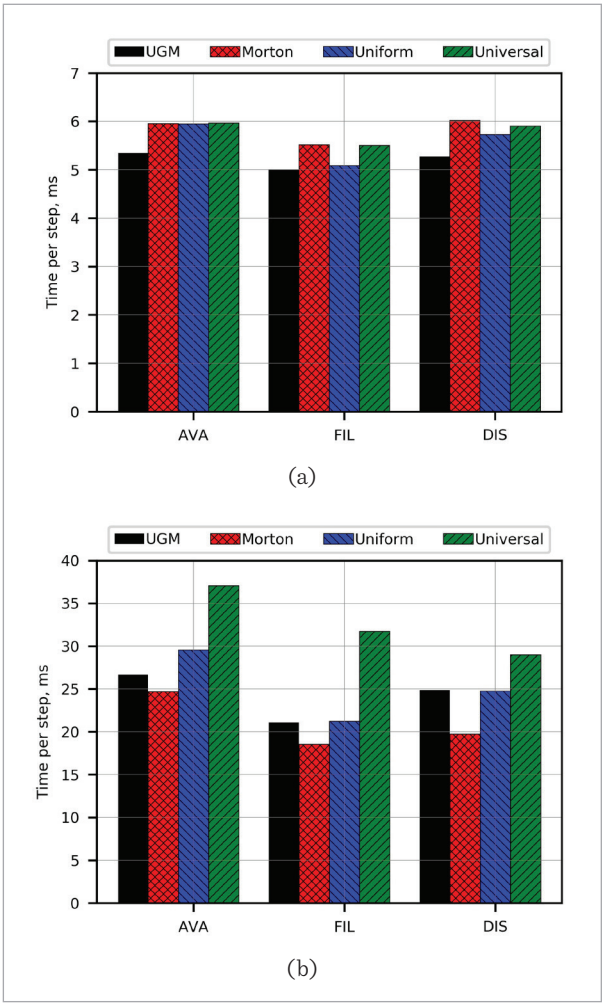
In the case of the hopper fill problem, which has a simple and small solution domain, the UGM consumed only 1.2 times more memory for contact search than the IGM, which results in 2.2 and 2.3 times more memory for the whole DEM computations, respectively. Therefore, the curves FIL IGM and FIL UGM are also located very close to each other. However, the UGM applied to solve the hopper discharge problem (the curve DIS UGM) and the artificial avalanche problem (the curve AVA UGM) consumed up to 65.1 and 69.7 times more memory for contact search then the IGM, which results in 2.2 and 2.3 times more memory for the whole DEM computations, respectively. It is worth noting that the developed GPU implementation reduced the percentage of memory consumed by contact search from 55.2% to 1.86% and from 56.9% to 1.86% of the total benchmark memory in the case of the hopper discharge problem and the artificial avalanche problem, respectively.

The developed implementation significantly reduced the GPU memory required by the standard UGM, which could lead to a decrease in computational performance. The applications of the uniform function [28], the Morton function [2] and the universal function based on the primary numbers [25] were investigated to evaluate and improve computational performance of the developed GPU implementation of the IGM.

Figure 4 presents the time-averaged values of the computing time consumed by contact search per time step in the case of three problems solved with different numbers of the particles. Figure 4a shows the contact search time of the hopper fill problem (FIL), the hopper discharge problem (DIS) and the artificial avalanche problem (AVA) solved with 108640, 108640 and 117649 particles, respectively. Figure 4b shows the computing time consumed by contact search of the considered problems solved with 963600, 963600 and 970299 particles. The UGM was the fastest for smaller numbers of particles (Figure 4a), but it was not the case for larger numbers of the particles (Figure 4b). The main observation was that the application of the Morton function resulted in the lowest computational performance in the case of smaller numbers of particles (Figure 4a). However, the lowest values of the computing time were measured applying the Morton function in the case of the larger numbers of particles (Figure 4b).

Figure 5 shows the time-averaged values of the computing time consumed by contact search per time step in the case of the considered hash functions mapping particles to hash tables of various sizes. The presented values of the contact search time were measured, solving the hopper fill problem (FIL), the hopper discharge problem (DIS) and the artificial avalanche problem (AVA) with 963600, 963600 and 970299 particles, respectively. In the legend of Figure 5, the abbreviations UGM, Morton, Uniform and Universal represent the solutions obtained by using the uniform grid method, the Morton hash function, the uniform hash function and the universal hash function, respectively. Initially, the table size was considered to be equal to the number of particles (the abbreviation S1). The memory required by the IGM was negligibly small compared to the memory consumed by the UGM, therefore, there was no sense to consider small-

Figure 4
The time-averaged values of the contact search time of three considered problems solved with different numbers of particles: (a) 108640 (FIL), 108640 (DIS) and 117649 (AVA) particles; (b) 963600 (FIL), 963600 (DIS) and 970299 (AVA) particles



er hash tables. Thus, the use of two (the abbreviation S2) and five (the abbreviation S5) times larger hash tables was investigated to increase the computational performance. The presented results show that the increased size of the hash table allowed for reducing the contact search time in the most cases. However, the observed performance increase often became very small in the case of the uniform and Morton functions with table size S5. Therefore, further increase in the table size did not appear very promising for the considered size of the solved problems.

Figure 5
The performance of contact search for various sizes of the hash tables

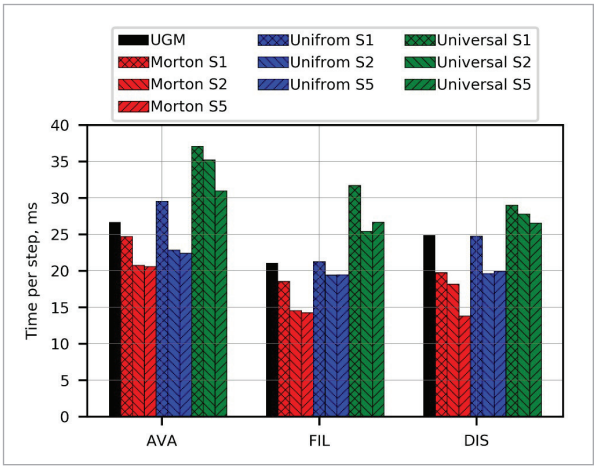
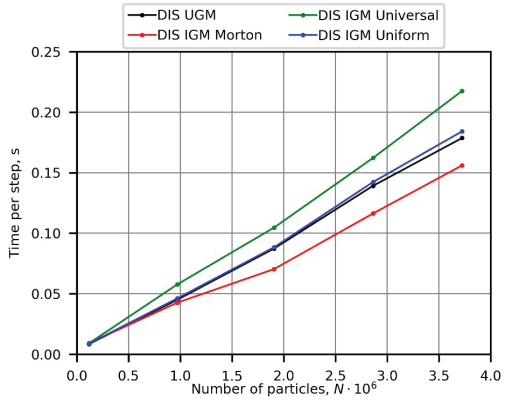


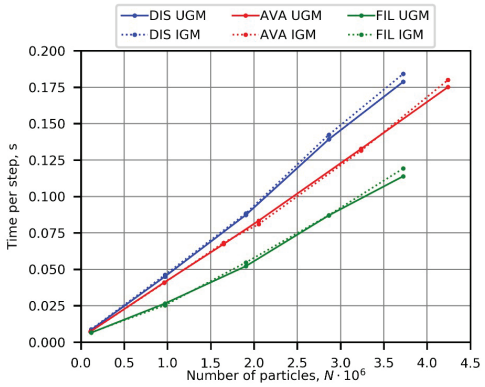
Figure 6 presents the performance scaling of the developed GPU software, which solved the considered problems, applying various hash functions for contact search. The average execution time of the OpenCL code per time step was measured computing 40000 time steps. In the legend of Figure 6, the abbreviations have the same meanings as in the previous figures. Figure 6a shows the performance scaling measured by solving the hopper discharge problem and applying various hash functions. As expected, the lowest performance could be observed by using the universal hash function, while the highest performance was achieved by applying the Morton hash function for contact search. Moreover, larger performance differences can be observed for larger numbers of particles, starting from 1906624. The difference in contact search time between the UGM and the IGM with the Morton hash function varied from 10.0% to 36.6% of the contact search time of the UGM. The resulting difference in the total computing time was up to 19.4% of the benchmark computing time obtained by using the UGM.

Figure 6b compares the performance of the developed GPU implementation, using the uniform hash function, and that of the standard UGM. In general, both methods demonstrate nearly the same performance for various numbers of particles. In most cases, the computing time of DEM simulations with the UGM is slightly shorter. However, the observed difference in

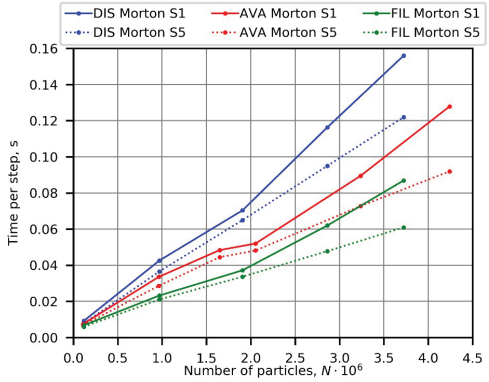
Figure 6
The performance scaling with various numbers of particles: (a) The solution of hopper discharge problem by using various hash functions; (b) The application of the uniform hash function and the UGM; (c) The application of the Morton hash function, increasing the size of the hash table



(a)



(b)



(c)

the computing time did not exceed 4.9% of the computing time with the UGM. It was due to the difference in the contact search time equal to 6.8% of the measured execution time of the UGM. The largest performance differences were obtained simulating the hopper fill because of the simple and comparatively small solution domain, which was very suitable for contact search performed by the UGM. The shortest computing times were measured solving the hopper fill problem because it had the smallest number of the detected contacts and the smallest number of the potential contacts in the considered time interval.

Figure 6c shows the influence of the table size on the measured performance in the case of applying the Morton function. The fivefold increase in the table size resulted in significant reduction in the computing time, which varied from 7.4% to 29.9% of the computing time measured by using the table size equal to the number of particles. Moreover, the largest differences in the computing time could be observed in the cases of the largest numbers of particles.

The computing time was reduced by 29.9%, 21.8% and 28.1% of the computing time measured by using the smaller table size, solving the hopper fill problem, the hopper discharge problem and the avalanche problem, respectively. It also allowed for achieving large Cundall numbers (FPS×the number of particles) equal to $6.11 \cdot 10^7$, $3.05 \cdot 10^7$ and $4.61 \cdot 10^7$ for the considered problems. Thus, the reduction in the table size can significantly increase computational performance, especially, in the case of large numbers of particles.

5. Conclusions

The paper presents the developed memory-saving GPU algorithm of contact search for DEM simulations in computational domains covered by a large number of grid cells and filled by particles, changing the occupied regions. The developed GPU algorithm and its OpenCL implementation significantly reduce memory consumption of the standard UGM. Moreover, the percentage of memory consumed by contact search becomes almost negligible comparing to the total benchmark memory, which is very important in the case of computational domains covered by a large number of grid cells. Computational performance of contact search highly depends on the applied hash

function and the considered number of particles. The standard UGM demonstrates the highest computational performance for problems with the number of particles, which slightly exceeds 100000. The IGM with Morton hash function is recommended, when the number of particles approaches one million. The presented performance analysis reveals a high potential of the developed GPU algorithm and OpenCL software, which significantly reduce memory consumption of contact search and preserve high computational performance, applying the Morton function and the proper hash table size.

Acknowledgement

The present research is part of the project No. 09.3.3-LMT-K-712-02-0131, funded under the European Social Fund measure "Strengthening the Skills and Capacities of Public Sector Researchers for Engaging in High Level R&D Activities", administered by the Research Council of Lithuania.

References

1. Cai, R., Xu, L., Zheng, J., Zhao, Y. Modified Cell-Linked List Method Using Dynamic Mesh for Discrete Element Method. Powder Technology, 2018, 340, 321-330. https://doi.org/10.1016/j.powtec.2018.09.034
2. Christer, E. Real-Time Collision Detection. CRC Press, 2004.
3. Cundall, PA., Strack, ODL. A Discrete Numerical Model for Granular Assemblies. Géotechnique, 1979, 29, 47-65. https://doi.org/10.1680/geot.1979.29.1.47
4. Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J. From CUDA to OpenCL: Towards a Performance-Portable Solution for Multi-platform GPU Programming. Parallel Computing, 2012, 38, 391-407. https://doi.org/10.1016/j.parco.2011.10.002
5. Durand, M., Marin, P., Faure, F., Raffin, B. DEM-Based Simulation of Concrete Structures on GPU. European Journal of Environmental and Civil Engineering, 2012, 16, 1102-1114. https://doi.org/10.1080/19648189.2012.716590
6. Džiugys, A., Peters, B. An Approach to Simulate the Motion of Spherical and Non-spherical Fuel Particles in Combustion Chambers. Granular Matter, 2001, 3, 231-266. https://doi.org/10.1007/PL00010918
7. Gan, J., Evans, T., Yu, A. Application of GPU-DEM Simulation on Large-Scale Granular Handling and Processing in Ironmaking Related Industries. Powder Technology, 2020, 361, 258-273. https://doi.org/10.1016/j.powtec.2019.08.043
8. Gan, J. Q., Zhou, Z. Y., Yu, A. B. A GPU-based DEM Approach for Modelling of Particulate Systems. Powder Technology, 2016, 301, 1172-1182. https://doi.org/10.1016/j.powtec.2016.07.072
9. Govender, N., Wilke, DN., Pizette, P., Abriak, N-E. A Study of Shape Non-uniformity and Poly-dispersity in Hopper Discharge of Spherical and polyhedral Particle Systems Using the Blaze-DEM GPU Code. Applied Mathematics and Computation, 2018, 319, 318-336. https://doi.org/10.1016/j.amc.2017.03.037
10. Green, S. Particle Simulation Using CUDA. NVIDIA whitepaper, 2010, 6, 121-128.
11. Kačeniauskas, A., Kačianauskas, R., Maknickas, A., Markauskas, D. Computation and Visualization of Discrete Particle Systems on gLite-based Grid. Advances in Engineering Software, 2011, 42, 237-246. https://doi.org/10.1016/j.advengsoft.2011.02.007
12. Kačeniauskas, A., Pacevič, R., Bugajev, A., Katkevičius, T. Efficient Visualization by Using ParaView Software on BalticGrid. Information Technology and Control, 2010, 39, 108-15.
13. Kačeniauskas, A., Pacevič, R., Starikovičius, V., Maknickas, A., Staškūnienė, M., Davidavičius, G. Development of Cloud Services for Patient-specific Simulations of Blood Flows Through Aortic Valves. Advances in Engineering Software, 2017, 103, 57-64. https://doi.org/10.1016/j.advengsoft.2016.01.013
14. Kačeniauskas, A., Pacevič, R., Staškūnienė, M., Šešok, D., Rusakevičius, D., Aidietis, A., Davidavičius, G. Private Cloud Infrastructure for Applications of Mechanical and Medical Engineering. Information Technology and Control, 2015, 44(3), 254-261. https://doi.org/10.5755/j01.itc.44.3.7379
15. Kačeniauskas, A., Rutschmann, P. Parallel FEM software for CFD problems. Informatica, 2004, 15(3), 363-378. https://doi.org/10.15388/Informatica.2004.066
16. Kačianauskas, R., Rimša, V., Kačeniauskas, A., Maknickas, A., Vainorius, D., Pacevič, R. Comparative DEM-CFD Study of Binary Interaction and Acoustic Agglom-

eration of Aerosol Microparticles at Low Frequencies. Chemical Engineering Research and Design, 2018, 136, 548-563. https://doi.org/10.1016/j.cherd.2018.06.006

17. Kalojanov, J., Slusallek, P. A parallel algorithm for construction of uniform grids. Proceedings of the Conference on High Performance Graphics, 2009, 1-23. https://doi.org/10.1145/1572769.1572773
18. Kelly, C., Olsen, N., Negrut, D. Billion Degree of Freedom Granular Dynamics Simulation on Commodity Hardware Via Heterogeneous Data-Type Representation. Multibody System Dynamics, 2020, 50, 355-379. https://doi.org/10.1007/s11044-020-09749-7
19. Lefebvre, S., Hoppe, H. Perfect Spatial Hashing. ACM Transactions on Graphics, 2006, 25, 579-589. https://doi.org/10.1145/1141911.1141926
20. Liu, G., Marshall, J. S., Li, S. Q., Yao, Q. Discrete-Element Method for Particle Capture by a Body in an Electrostatic Field. International Journal for Numerical Methods in Engineering, 2010, 84, 1589-1612. https://doi.org/10.1002/nme.2953
21. Lutz, K. Boost Compute. https://www.boost.org/doc/libs/1_76_0/libs/compute/doc/html/index.html. Accessed on July 9, 2021.
22. Mazhar, H., Heyn, T., Negrut, D. A Scalable Parallel Method for Large Collision Detection Problems. Multibody System Dynamics, 2011, 26(1), 37-55. https://doi.org/10.1007/s11044-011-9246-y
23. Miao, Q., Huang, M., Xue, J., Ben, Y. Spatial Hashing Based Contact Detection for Numerical Manifold Method. Geomechanics and Geoengineering, 2014, 9(2), 153-159. https://doi.org/10.1080/17486025.2013.871072
24. Pacevič, R., Kačeniauskas, A. The Development of VisLT Visualization Service in Openstack Cloud Infrastructure. Advances in Engineering Software, 2017, 103, 46-56. https://doi.org/10.1016/j.advengsoft.2016.06.012
25. Satish, N., Harris, M., Garland, M. Designing Efficient Sorting Algorithms for Manycore GPUs. IEEE International Symposium on Parallel & Distributed Processing, 2009, 1-10. https://doi.org/10.1109/IPDPS.2009.5161005
26. Shigeto, Y., Sakai, M. Parallel Computing of Discrete Element Method on Multi-Core Processors. Particuology, 2011, 9, 398-405. https://doi.org/10.1016/j.partic.2011.04.002
27. Stupak, E., Kačianauskas, R., Kačeniauskas, A., Starikovičius, V., Maknickas, A., Pacevič, R., Staškūnienė, M., Davidavičius, G., Aidietis, A. The Geometric Model-Based Patient-Specific Simulations of Turbulent Aortic Valve Flows. Archives of Mechanics, 2017, 69(4-5), 317-345.
28. Tang, M., Liu, Z., Tong, R., Manocha, D. PSCC: Parallel Self-Collision Culling with Spatial Hashing on GPUs. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2018, 1, 1-18. https://doi.org/10.1145/3203188
29. Tian, Y., Zhang, S., Lin, P., Yang, Q., Yang, G., Yang, L. Implementing Discrete Element Method for Large-Scale Simulation of Particles on Multiple GPUs. Computers & Chemical Engineering, 2017, 104, 231-240. https://doi.org/10.1016/j.compchemeng.2017.04.019
30. Tumonis, L., Kačianauskas, R., Kačeniauskas, A., Schneider, M. The Transient Behavior of Rails Used in Electromagnetic Railguns: Numerical Investigations at Constant Loading Velocities. Journal of Vibroengineering, 2007, 9(3), 15-19.
31. Tykhoniuk, R., Tomas, J., Luding, S., Kappl, M., Heim, L., Butt, H-J. Ultrafine Cohesive Powders: from Interparticle Contacts to Continuum Behaviour. Chemical Engineering Science, 2007, 62, 2843-2864. https://doi.org/10.1016/j.ces.2007.02.027
32. Walther, J. H., Sbalzarini, IF. Large-Scale Parallel Discrete Element Simulations of Granular Flow. Engineering Computations, 2009, 26, 688-697. https://doi.org/10.1108/02644400910975478
33. Xu, J., Qi, H., Fang, X., Lu, L., Ge, W., Wang, X., Xu, M., Chen, F., He, X., Li, J. Quasi-Real-Time Simulation of Rotating Drum Using Discrete Element Method with Parallel GPU Computing. Particuology, 2011, 9, 446-450. https://doi.org/10.1016/j.partic.2011.01.003
34. Yan, B., Regueiro, R. Comparison Between O(n2) and O(n) Neighbor Search Algorithm and Its Influence on Superlinear Speedup in Parallel Discrete Element Method (DEM) for Complex-Shaped Particles. Engineering Computations, 2018, 35(6), 2327-2348. https://doi.org/10.1108/EC-01-2018-0023
35. Yue, X., Zhang, H., Ke, C., Luo, C., Shu, S., Tan, Y., Feng, C. A GPU-Based Discrete Element Modeling Code and Its Application in Die Filling. Computers & Fluids, 2015, 110, 235-244. https://doi.org/10.1016/j.compfluid.2014.11.020
36. Zheng, J., An, X., Huang, M. GPU-Based Parallel Algorithm for Particle Contact Detection and Its Application in Self-Compacting Concrete Flow Simulations. Computers & Structures, 2012, 112-113, 193-204. https://doi.org/10.1016/j.compstruc.2012.08.003