Multibody Syst Dyn
https://doi.org/10.1007/s11044-020-09749-7

Billion degree of freedom granular dynamics simulation on commodity hardware via heterogeneous data-type representation

Conlain Kelly[1] · **Nicholas Olsen**[1] · **Dan Negrut**[1]

Received: 1 February 2020 / Accepted: 27 May 2020

Abstract We discuss modeling, algorithmic, and software aspects that allow a simulation tool called Chrono::Granular to run billion-degree-of-freedom dynamics problems on commodity hardware, i.e., a workstation with one GPU. The ability to scale the solution to large problem sizes is traced back to an adimensionalization process combined with the use of mixed-precision data types that reduce memory pressure and improve arithmetic intensity, judicious use of the memory ecosystem on GPU cards as exposed by CUDA on Nvidia architectures, and a software implementation that prioritizes execution speed over modeling generality. The simulation approach is demonstrated for 3D scenarios with up to 710 million bodies for the frictionless case (of relevance in emulsions), and up to 210 million bodies for scenarios with friction (of relevance in terradynamics, additive manufacturing, soft-matter physics). The frictional contact model used draws on the Discrete Element Method (DEM). A performance benchmark shows linear scaling with problem size up to GPU memory capacity. The implementation has an application programming interface that enables it to interact in a cosimulation framework with third-party dynamics engines. This interaction is anchored by a force–displacement data exchange protocol that brings in external bodies as geometries defined by triangle meshes. We demonstrate the cosimulation mechanism by interfacing to an open source, multiphysics simulation engine called Chrono. Therein, triangular meshes define moving boundary conditions for Chrono::Granular, which in turn provides forces and torques acting on the triangular meshes. Several tests are considered for validation and scaling analysis purposes. The limiting aspects of the current implementation are its exclusive support of monodisperse granular systems, and its lack of handling geometries beyond spheres. These limitations are addressed by ongoing work.

✉ D. Negrut
negrut@wisc.edu

C. Kelly
ckelly5@wisc.edu

N. Olsen
nicholas.olsen@wisc.edu

1 Dept. of Mechanical Engineering, University of Wisconsin-Madison, Madison, WI 53706, USA

Published online: 03 June 2020

Keywords Many-body dynamics · GPU parallel computing · Friction and contact · Discrete element method · Granular material

1 Introduction

More than 50% of the materials processed/handled in industry come in granular form [1]. Understanding their dynamics is relevant in a range of practical applications and scientific pursuits, e.g., additive manufacturing, terramechanics, composite materials, suspensions, pyroclastic flows, formation of asteroids and planets, meteorite cratering, etc. Handling frictional contact dynamics in granular materials can be done via a so-called contact dynamics (CD) approach [2] or a Discrete Element Method (DEM) formalism [3]. For a comparison of the two methodologies, see [4]; herein, the approach used draws exclusively on DEM. Regardless of the formulation though, CD or DEM, fully resolved simulations for the granular dynamics problem remain time consuming. Attempts to coarse-grain the granular material (see, for instance, [5–7]) have met with good success when certain conditions are met: the material is dry, the particles have spherical shape, and the spheres have identical radii. Yet there are numerous applications that don't meet these assumptions and which typically fall back on DEM, the "go-to" solution, extensively validated and proven to provide reliable results in granular dynamics [8–11]. However, the use of DEM comes with caveats: (i) selecting the right DEM model parameters is problem-specific and requires nontrivial effort and insights; (ii) fully resolved granular systems associated with practical problems can lead to billions of elements, which poses storage and processing challenges; (iii) the shape of the discrete elements in practical applications is complex, which makes the contact detection process time consuming to the point where in most cases one compromises on the geometry for the sake of maintaining reasonable execution speed; and (iv) the physical properties of the grains (material and size attributes), might place stringent limits on the size of the simulation time step.

This contribution addresses (ii) above by relying on GPU computing, with potential future work outlined for addressing (iii). Leveraging bandwidths close to 1000 GB/s (compared to 100 GB/s on high end CPU systems) and compute rates close to 15 Tflops/s (compared to 1 Tflops/s on high end CPU systems), we demonstrate granular dynamics simulations with more than 700 million elements for frictionless systems (relevant in emulsions) and 220 million for systems that employ a Mindlin-type friction model with memory. To the best of our knowledge, the largest granular dynamics simulation of practical relevance to date contained 2.4 billion elements. It was run on 16,384 CPUs (131,072 cores) of Japan's K-supercomputer [12–14], the 2012 fastest supercomputer in the world and now the 18th in the ranking of the world's supercomputers [15]. Although we report frictional contact dynamic problem of sizes roughly one-tenth of the size reported in [12–14], our approach uses one GPU instead of more than 100,000 cores. Moreover, the source code of the software implementation is available as open source for unrestricted use. The key aspects of the approach adopted are: an adimensionalization process combined with mixed-precision data, full leverage of the GPU memory ecosystem exposed through the Nvidia CUDA programming environment, and exploiting a select set of assumptions that trade modeling generality for performance and scalability.

The manuscript is organized as follows. Section §2 contains a description of the DEM model and equations of motion used, and the adimensionalization process that affords the use of integers for storing position information in the DEM simulation. Section §3 highlights implementation aspects tied to heterogeneous data use (integers for positions and single

Table 1 Model and simulation parameters

N_b	Particle count
m^i	Particle i mass
R^i	Particle i radius
ρ^i	Particle i density
$\mathbf{g}$	Gravitational acceleration
k_n	Normal elastic coefficient
γ_n	Normal damping coefficient
k_t	Tangential elastic coefficient
γ_t	Tangential damping coefficient
μ_s	Coefficient of *sliding* friction
μ_r	Coefficient of *rolling* friction

precision floating point values for velocity); data structures used in the code; extending the simulation domain via local coordinates; computational flow, and use of cosimulation for interfacing to third-party dynamics engines. Section §4 reports results of a scaling analysis tied to a commonly used benchmark problem, namely filling of a box with granular material. Section §5 contains results associated with several numerical experiments: hopper flow, dam brake test, and communicating vessels experiments. Section §6 places the present work in a broader context by providing a literature overview in which the focus is on problem sizes, as they have been defining the state-of-the-art in various application areas. The last section summarizes a set of conclusions and directions for future work.

2 The DEM model

2.1 The equations of motion

We summarize below a general formulation of the equations of motion that draws on the classical DEM method [3]. Superscripts denote the relevant particle (or pair of particles), whereas subscripts denote components of a vector. Since no exponentiation occurs in this section, there should be no conflict with this use of superscripts. Table 1 summarizes the parameters associated with the DEM model adopted herein. Two spheres in mutual contact are shown in Fig. 1, along with the normal and tangential (friction) forces and several kinematic quantities that come into play in computing the frictional contact force.

For spherical element i, let $\mathcal{C}(i;t)$ be the collection of bodies that i is in mutual contact with at time t. The time evolution of sphere i is governed by the Newton–Euler equations of motion:

$$m^i \frac{d\mathbf{v}^i}{dt} = m^i \mathbf{g} + \sum_{j \in \mathcal{C}(i;t)} \left(\mathbf{F}_n^{ij} + \mathbf{F}_t^{ij} \right), \tag{1a}$$

$$I^i \frac{d\boldsymbol{\omega}^i}{dt} = \sum_{j \in \mathcal{C}(i;t)} \mathbf{M}^{ij} = \sum_{j \in \mathcal{C}(i;t)} \Delta\mathbf{r}^{ij} \times \mathbf{F}_t^{ij} + \mathbf{M}_{rr}^{ij}, \tag{1b}$$

where, for element i, m^i is its mass; I^i is its mass moment of inertia (a scalar in the spherical case); the velocity $\mathbf{v}^i = \dot{\mathbf{r}}^i$ is the time derivative of the position; $\Delta\mathbf{r}^{ij}$ is the vector from element's i center of mass to the point where the friction force $\mathbf{F}_t^{ij}$ is applied; and $\mathbf{M}_{rr}^{ij}$ is

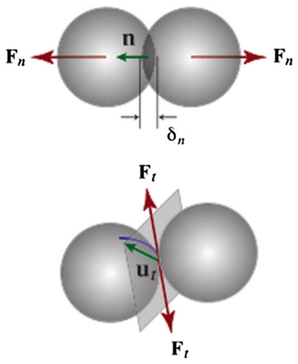


Fig. 1 Schematic of two spheres in contact

the rolling resistance moment. Given that all elements are spheres, there is no need to store their orientation, and the angular velocity $\boldsymbol{\omega}^i$ will provide the information needed to track the friction history at the point of contact. The normal and tangential forces are determined using the Hertzian and Mindlin contact formulations, respectively:

$$\mathbf{F}_n^{ij} = \sqrt{\frac{\delta^{ij}}{2\bar{R}^{ij}}}\left(k_n\delta^{ij}\mathbf{n}^{ij} - \gamma_n\bar{m}^{ij}\mathbf{v}_n^{ij}\right) + \mathbf{F}_c, \tag{1c}$$

$$\mathbf{F}_t^{ij} = \sqrt{\frac{\delta^{ij}}{2\bar{R}^{ij}}}\left(-k_t\mathbf{u}_t^{ij} - \gamma_t\bar{m}^{ij}\mathbf{v}_t^{ij}\right) . \tag{1d}$$

Above, $\mathbf{F}_c$ is a cohesion force, $\bar{m}^{ij}$ and $\bar{R}^{ij}$ represent the effective mass and effective radius for the contact, and $\mathbf{v}_n^{ij}$ and $\mathbf{v}_t^{ij}$ are the normal and tangential components of the relative velocity:

$$\bar{m}^{ij} = \frac{m^i m^j}{m^i + m^j}, \tag{1e}$$

$$\bar{R}^{ij} = \frac{R^i R^j}{R^i + R^j}, \tag{1f}$$

$$\mathbf{v}^{ij} = \mathbf{v}^j + \boldsymbol{\omega}^j \times \Delta\mathbf{r}^{ji} - \mathbf{v}^i - \boldsymbol{\omega}^i \times \Delta\boldsymbol{r}^{ij}, \tag{1g}$$

$$\mathbf{v}_n^{ij} = (\mathbf{v}^{ij} \cdot \mathbf{n}^{ij})\mathbf{n}^{ij}, \tag{1h}$$

$$\mathbf{v}_t^{ij} = \mathbf{v}^{ij} - \mathbf{v}_n^{ij} . \tag{1i}$$

The friction force is capped relative to the normal contact force via the Coulomb criterion $|\mathbf{F}_t^{ij}| \leq \mu\,|\mathbf{F}_n^{ij}|$.

In order to compute the forces in Eqs. (1c) and (1d), one must evaluate the contact normal $\mathbf{n}^{ij}$, normal penetration δ^{ij}, tangential displacement at the contact point $\mathbf{u}_t^{ij}$, and normal and tangential velocities $\mathbf{v}_n^{ij}$ and $\mathbf{v}_t^{ij}$, respectively. Except for $\mathbf{u}_t^{ij}$ these quantities can be readily computed from the sphere positions, linear velocities, and angular velocities.

The computation of $\mathbf{u}_t^{ij}$ is somewhat more involved. Dropping the ij superscripts for notational brevity, $\mathbf{u}_t$ is updated in a multistep fashion, i.e., from integration step to integration step, tracking its history between steps. We employ the notation $\mathbf{u}_{t,k}$ to denote the tangential displacement history at time step k. In Chrono::Granular, this quantity is always enforced to be perpendicular to the current contact normal, i.e., to lie in the contact plane:

$$\begin{aligned} \mathbf{u}^*_{t,k} &= \mathbf{u}_{t,k-1} + \mathbf{v}_{t,k}\Delta t, \\ \mathbf{u}_{t,k} &= \mathbf{u}^*_{t,k} - (\mathbf{n}_k \cdot \mathbf{u}^*_{t,k})\mathbf{n}_k \,. \end{aligned} \tag{2}$$

This pairwise interaction history must be maintained for each partner in $\mathcal{C}(i;t)$ for each particle i. The Coulomb capping of the friction force is then enforced via

$$\mathbf{u}_t = min\left(\mathbf{u}_{t,k}, \mathbf{u}_{t,k}\frac{\mu_s k_n \delta_n}{k_t|\mathbf{u}_{t,k}|}\right) , \tag{3}$$

where μ_s is taken to be the same between static and dynamic motions. Note that since the nonlinear $\sqrt{\frac{\delta^{ij}}{2\bar{R}^{ij}}}$ term appears in both $\mathbf{F}_t$ and $\mathbf{F}_n$, it cancels out of the Coulomb criterion and does not effect clamping. Moreover, only the frictional force computed in Eq. (1d) is clamped – the rolling resistance torque described below is not affected by the Coulomb criterion.

In Chrono::Granular, $\mathbf{v}_t$ is interpreted as the relative velocity at the contact point including both sliding and rolling contributions. This enforces a situation of "rolling without slipping", where only the relative displacement of the contact point is relevant and not the means of locomotion. An additional rolling resistance torque $\mathbf{M}^{ij}_{rr}$ of Eq. (1b) is introduced based on ideas outlined in [16–19] to bring in an interparticle rolling resistance meant to emulate effects induced by roughness and interlocking. Two rolling resistance models are implemented based on work reported in [18]. The simpler scales a constant torque by the magnitude of the normal component of the contact force, where $\boldsymbol{\omega}^{ij} = \boldsymbol{\omega}^i - \boldsymbol{\omega}^j$:

$$\mathbf{M}^{ij}_{rr} = \frac{\boldsymbol{\omega}^{ij}}{|\boldsymbol{\omega}^{ij}|}\mu_r \bar{R}^{ij}|\mathbf{F}^{ij}_n|. \tag{4}$$

The second model applies a resistive moment based on the relative slip at the contact point due to relative roll, i.e.,

$$\mathbf{M}^{ij}_{rr} = -\mu_r \bar{R}^{ij}|\mathbf{F}^{ij}_n|(\boldsymbol{\omega}^i \times \Delta\mathbf{r}^{ij} - \boldsymbol{\omega}^j \times \Delta\mathbf{r}^{ij}) \,. \tag{5}$$

2.2 User units vs. simulation units

Regardless of the system of units employed by the user, e.g., CGS, MKS, etc. (called herein "user units"), the code internally adimensionalizes and rescales quantities. This change of units is motivated by the interplay between two aspects (further discussed in Sect. 3.1): (i) for granular systems with elements that are very small (low mass) and stiff, computing normal or tangential deformations in user units may become numerically compromised due to finite precision representation and arithmetic constraints; and (ii) computationally it is more efficient to use integers than floats, and even better to use single-precision floats than doubles. As a consequence, the position of the elements is stored internally using 32-bit integers as explained in Sect. 3.2. Most other quantities are stored as 32-bit floating point numbers (`float`), and in instances where critical values are evaluated, the computation carries out in 64-bit double precision, and then quantities appropriately recast to `int` or single precision `float` type. As such, a means of converting quantities into a consistent range of values is needed so that we can reliably use mixed-precision data types and avoid related overflow/underflow issues.

The unit conversion consists of two steps—establishing base units and determining scaling factors. To convert all quantities, we first select characteristic mass, length, and time units. Each unit is subsequently divided by a scaling factor (ψ_m, ψ_ℓ, or ψ_t) for the unit to become more tractable with mixed-precision types. Based on a Hertzian force model with spring constant k_n, sphere mass m_s, sphere radius R_s, and system gravity g, these are chosen as

$$m_{\text{unit}} = \frac{1}{\psi_m} m_s, \tag{6a}$$

$$\ell_{\text{unit}} = \frac{1}{\psi_\ell} \left(\frac{m_s g}{k_n}\right)^{\frac{2}{3}} R_s^{\frac{1}{3}}, \tag{6b}$$

$$t_{\text{unit}} = \frac{1}{\psi_t} \sqrt{\frac{m_s}{k_n}}. \tag{6c}$$

All derived units are then expressed as multiples of these units, i.e., the simulation velocity unit is $\frac{\ell_{\text{unit}}}{t_{\text{unit}}}$. Internally, all user-provided quantities are converted to this adimensional set of so called "simulation units". In fact, the user's unit system does not come into play insofar as the simulation's internals are concerned.

This decision of ℓ_{unit} was chosen so that the deformation required to sustain a single sphere resting on top of another sphere will be ψ_ℓ in the new set of units. Thus, a typical deformation will be order ψ_ℓ or greater, so that most deformations are above order unity and therefore positions can be well represented by integral multiples of ℓ_{unit}. We have worked with $\psi_\ell = 16$ in most experiments, but this selection is user-controlled. Indeed, this choice translates into determining how many units of length go along with the overlap witnessed when one sphere sits at rest on the top of a different one. Note that this adimensionalization applies to particulate systems beyond monodisperse spheres. One could replace m_s and R_s with a characteristic particle mass and size, or perhaps those of the smallest in the system to ensure stability for all particles. One possible limitation with this approach would occur in low-gravity situations, so that g (and correspondingly ℓ_{unit}) are very small. However, these situations also lead to an ill-conditioning of the system similar to very still or small particles, regardless of the adimensionalization.

Of the three user-selected scaling factors, the most consequential selection is that of ψ_ℓ as it impacts the accuracy of data representation for position. Although Chrono::Granular uses a nonlinear Hertzian contact force model, it still relies on a pseudolinear spring constant k_n (having units force per distance) and adds in the nonlinearity through a dimensionless Hertz stiffening factor $\sqrt{\frac{\delta^{ij}}{2\bar{R}^{ij}}}$. Quantitatively, this means that the spring constant supplied to provide a certain normal force for a given deformation needs to be larger for the Hertzian case than a pure Hookean case, since the stiffening factor actually has magnitude less than unity. However, the nonlinear spring stiffens faster, so that the force evolution over a period of contact is different. By using this stiffening, we can retain the meaning of a linear spring constant but obtain the desirable nonlinear behavior a Hertz model provides.

3 Software design aspects. Implementation details

3.1 Heterogeneous data-type representation

Several scales come into play in the typical DEM simulation, i.e., that of the target application, of the particle, and of the particle deformation. The cornerstone of the approach embraced is its use of different data types for storing and computing information pertaining these scales. Consider, for instance, two spheres of density $\rho = 2600$ kg/m^3, radius $R = 0.5 \times 10^{-3}$ m, and stiffness $k = 10^8$ N/m – one resting on top of the other. With a simple Hookean contact model, the force balance leads to $\frac{4}{3}\rho\pi R^3 g = k\delta \Longrightarrow \delta = \frac{4}{3}\frac{\rho\pi R^3 g}{k}$. Then, δ is on the order of 10^{-13}. This penetration must be computed from two elements' position values. Note that the machine precision for an IEEE float is $\epsilon_{\text{mach}} \approx 10^{-7}$ for a 32-bit float and $\epsilon_{\text{mach}} \approx 10^{-16}$ for a 64-bit double precision floating point number. Imagine next that two spheres in contact are part of a deformable terrain on which a rover operates; it is reasonable to assume that the application scale, and thus the two position values mentioned above, are at least order 1 m. As such, given the IEEE standard machine precision ϵ_{mach} values above, calculating δ will experience catastrophic loss of precision with a 32-bit float and have little margin with a 64-bit float. The loss of precision situation becomes more dramatic if the rover operates on the moon, Mars, a moon of Mars, or an asteroid, when the gravitational acceleration, and thus δ, can be markedly smaller. Note that albeit to a lesser degree, these loss of precision issues also pertain to a Hertzian force model, and indeed to any penalty-based DEM model that spans multiple scales.

By using the unit scaling in Eq. (6b), a particle's position herein is stored in a 32-bit `int` (integer) type. From a high vantage point, the idea promoted is as follows: both the single precision `float` and `int` data types have a 32 bit budget. However, for the former data type, this bit budget is used to generate machine numbers, i.e., proxies, for *all* values between $-\infty$ and $+\infty$. In a rover mobility application or some other typical granular flow simulation, one needs to capture position values in the range, for instance, $(-10, 10)$ meters. As such, bits provisioned to represent, for instance, values in the hundreds of millions, go wasted since the simulation doesn't hit such values. Instead, it is more advantageous to use the 32 bits organized as an `int` thus capturing approximately four billion of them, and represent any location as an integer multiple of a smallest value, which was denoted in Eq. (6b) by ℓ_{unit}. Then, the range of positions that can be captured, extends over a distance of $2^{32} \times \ell_{\text{unit}}$ in each of the three directions of the 3D space (a discussion of how to go beyond this range is saved for Sect. 3.3). Hence, no bits are wasted on position values in the hundreds of millions, which do not actually come into play in the typical DEM application. It should be noted that the size of the error introduced by this approach is of the order $\frac{\ell_{\text{unit}}}{2}$ (in user units). Indeed, the center of mass (CM) of the body, can only be placed at multiples of ℓ_{unit}. As such, any scenario in which the solver would try to place the body somewhere in between two grid points will lead to "round off." For simplicity, assume that the problem is 1D. In this case, the CM can be placed at location $\ldots, -1, 0, 1, 2, \ldots$, etc. For instance, the body cannot be placed at a location 4.32, in this case it will be placed at location 4; likewise, it cannot be placed at position 7.68, in this case it would be rounded to the next integer, that is, 8. Thus, for any location "off-grid", there will be round-off that, in user units, is less than $\frac{\ell_{\text{unit}}}{2}$. It should be pointed out that the straight use of C++/C's `double` data type instead of `int` to store positions would be superior in terms of precision. However, this would require 64 bits, which translates into doubling the amount of memory necessary to store position information. Also, it would lead to an increase in simulation time given that the effective bandwidth would drop by a factor of two. Finally, one additional advantage of the integer-position approach is that the numerical error is uniform over the entire range, whereas `float` would have nonuniform error over its range of values.

3.2 Data structures

Throughout its implementation, Chrono::Granular uses CUDA's managed memory mechanism [20, 21], wherein the CUDA runtime provides a unified virtual address space in which data pointers can be invoked transparently from either host or device. The benefits are twofold: effortless host–device data transfers, and cleaner implementation. The dynamic quantities for an element, i.e., positions, velocities, and accelerations, are stored in a structure-of-arrays format to maximize coalesced memory accesses and leverage effectively the GPU's high bandwidth. For host-side data management, C++ STL vectors are used with a custom memory allocator that relies on CUDA unified memory, a decision that improved the performance, brevity, and maintainability of the code base. A list of managed memory pointers that come into play in device execution is stored itself as a structure that is passed by pointer between CUDA function calls.

The rules followed in the implementation are summarized as follows: (I) use `double` data type (8 bytes) variables only when absolutely necessary and if at all possible only in conjunction with variables stored in fast or very fast memories (see Fig. 2); (II) use four-byte (or less) data types for variables that are stored in slow memory; (III) in any circumstance, use the data type that meets the accuracy and/or range necessary at the smallest byte budget. The specific data types used in the implementation are elaborated below (carrying C++/CUDA semantics).

The position of the center of mass of a particle is stored using three `signed ints`. The translational and angular velocities are each stored as a collection of three `float`-type variables. Note that only when friction is present do angular velocities come into play, and memory is conditionally allocated accordingly. The common particle mass, the constant cohesion force, and the acceleration of gravity are all stored as `float`. No element state information is stored in global memory in `double` precision in order to improve memory bandwidth and reduce memory pressure; however, `double`-precision types are used internally for intermediate quantities when computing penetrations and forces.

During collision detection, spheres can be binned using just the integer positions. However, when the normal penetration δ is computed, all quantities are first upcast to `doubles` and then used to compute the normal penetration in double precision. The normal force is computed from the penetration, but then downcast to float. There is no major loss of significant digits to this downcast since these forces were only computed from 32-bit quantities to start with. This ensures that minimal precision is lost in the many arithmetic operations required, including multiple subtractions, divisions, and even a reciprocal square root. Although doing this in `double` is slightly slower, the overhead is minimal compared to the overhead of loading all the required data from global memory. When friction is present, the friction force is computed as `float` using local tangential kinematic information (see Eq. (2)) stored as `floats`. Other nonposition temporary and helper variables are generally stored as `floats`.

These choices of data types are largely motivated by the hardware design of the modern graphics card. Referring to Fig. 2, which uses hardware notation conventions associated with Nvidia GPUs, a triplet of integers are stored in the "Device Global Memory" and used to locate in the 3D space a particle i. When brought over to a GPU stream multiprocessor (SM) for use, this data is placed in cache and/or shared memory and/or registers, i.e., in fast memory. Then, for instance, when a distance from two points is evaluated to compute the penetration δ of two elements, the integers are first converted to doubles and the distance is computed like $\sqrt{(x^i - x^j)^2 + (y^i - y^j)^2 + (z^i - z^j)^2}$, where x^i is obtained upon double precision conversion from the integer value p^i_x, y^i is obtained from p^i_y, and so on. Note that this conversion and the ensuing math operations (carried out by the scalar processors (SP) or by the special function units, the latter not shown in Fig. 2) involve the fast memory of the SM, i.e., registers/L1 cache/shared memory. Storing integers in "Device Global Memory" and then bringing them over to the SM through the purple conduit (i) reduces the

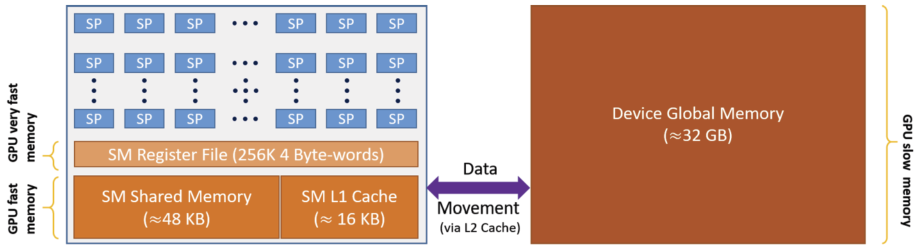

SP
GPU very fast memory
GPU fast memory
SM Register File (256K 4 Byte-words)
SM Shared Memory (≈48 KB)
SM L1 Cache (≈ 16 KB)
Data Movement (via L2 Cache)
Device Global Memory (≈32 GB)
GPU slow memory

Fig. 2 Schematic showing the hardware layout and the memory hierarchy in a typical Nvidia GPU. Each stream multiprocessor (SM), shown as the left rectangle, possesses a relatively limited amount of fast memory, i.e., registers, L1 cache, shared memory. The SM has access though to a relatively large amount of slower global memory (represented as the right rectangle of solid color). The slower memory still has a very significant bandwidth, approximately one order of magnitude higher than what is typically available on a CPU. A better GPU card has by comparison more SM units

Table 2 Slices of the different history vectors for a given body. Notation used: RCI – Relative contact index

RCI	*Active* `(bool)`	*Partner Body ID* `(unsigned int)`	*History vector* `(float3)`
0	x	x	(x, x, x)
1	x	x	(x, x, x)
2	x	x	(x, x, x)
⋮	⋮	⋮	⋮
11	x	x	(x, x, x)

memory footprint since the “Device Global Memory” stores integers; and (ii) improves the effective bandwidth since twice the amount of useful information is brought over to the SM in the unit of time. As order of magnitude, while bringing the integers from the “Device Global Memory” happens rarely but incurs an overhead of hundreds of cycles, the values in registers/L1 cache/shared memory are accessed very often but at very small overheads, of order from 1 to 10 s of cycles. This is critical as most often the simulation bottleneck is tied to the memory transactions and not to the arithmetic operations. Indeed, in a pipelined architecture present on today’s GPUs one fused multiply–add operation can be completed at the end of each clock cycle, which is to be compared to hundreds of cycles required by a “Device Global Memory” access.

Regarding restrictions present in the current implementation, the data structures are mostly general, except in two cases. First, the data structures for frictional force computation build off the observation that one element can have up to 12 neighbors (specific data structure, with information pertaining these 12 neighbors, shown in Table 2). For polydisperse or nonspherical particles, either this 12 would have to be replaced by a different cap on the maximum number of simultaneous contacts allowed in the simulation, or a more general data structure (such as a GPU hash map) would be required. Note that for the table used in the current implementation, this memory is only allocated once (to avoid reallocations and copy overhead), so that one cannot adjust this number during runtime. However, such an adaptive reallocating approach could potentially be used, and indeed was employed for the cosimulation API (discussed in Sect. 3.6). Second, given that particles are monodisperse, little storage was required for problem parameters such as stiffnesses, radii, or particle shape. This translated into a smaller memory footprint as well as less data movement. For a polydisperse system, the memory overhead would be commensurately larger to accommodate variation in these parameters.

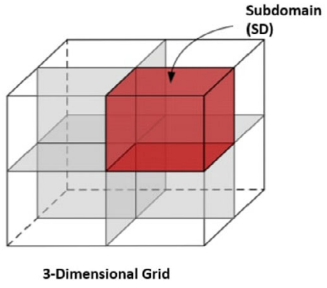
Finally, one salient aspect of the current implementation is that the DEM problem size is not strictly limited to the size of the memory available on the GPU card. Indeed, given that the implementation relies on CUDA-managed memory, a user-transparent paging mechanism kicks in to page data between CPU and GPU memory when the latter is exhausted. This point is further elaborated upon in Sect. 4 in conjunction with a scaling analysis experiment.

3.3 Extending the spanned physical domain via local coordinates

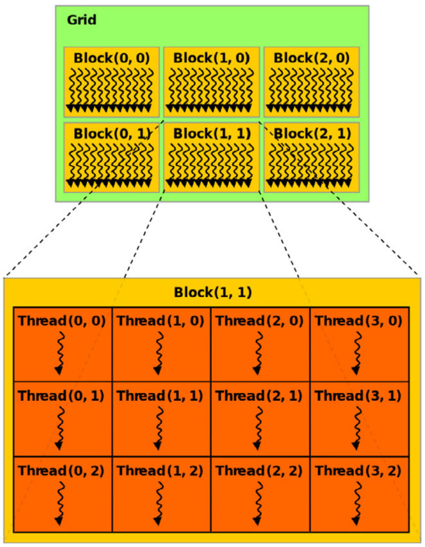
A DEM problem with any combination of the following: large contact stiffnesses, small element radii, or low gravity, will lead to tiny deformations (akin to a very small ψ_ℓ) that must be tracked in simulation. Consequently, the small simulation unit for length in combination with the use of `int` data to store positions could limit too much the size of the domain that can be spanned in simulation (called in what follows the "big domain"). The current implementation evades this limitation by employing local coordinates, a solution that can also be regarded as using one level of indirection in producing particle location. In the indirection-based approach, a second set of integers is used to identify a collection of subdomains (SDs) that together span the big domain. Then, each particle's position field embeds one additional piece of information – the ID of the SD whose reference frame is used to define its location.

The following process is employed to position a sphere: The big domain is split into SDs or bins (used also for broad-phase contact detection, see Sect. 3.5), each spanning approximately 4 sphere-diameters in each dimension. These axis-aligned SDs form a cubic grid that spans the entire big domain, visualized in Fig. 3a. Each sphere's position is defined as the combination of two values: the index of the SD that owns a sphere, and the sphere's relative position within that SD. The SDs are indexed via one `unsigned int`; within the SD, the sphere's relative position is represented as an `int3` (x, y, and z offsets). This requires a total of $4 \times 4 = 16$ bytes to represent a sphere's position, compared to the $3 \times 8 = 24$ bytes if three `long ints` or `doubles` are used. Because the SDs are ordered in a grid, most of the relevant digits of a sphere's absolute positions can be stored in that single `unsigned int` (assuming there are less that 2^{32} SDs, which holds most problems of physically reasonable scale). Moreover, since the contact and friction forces are inherently local quantities, these local integer positions are the only information needed for almost the entire simulation process. In output/postprocessing situations where absolute global positions are required, global positions are computed in simulation units as triplets of long integers and then dumped back into user units.

This approach provides several advantages. First, by assigning one physical SD to one CUDA thread block, the implementation exploits data locality by having the execution draw on the SM's fast memory, see Fig. 2. In fact, the DEM solution maps really well onto the GPU architecture: the CUDA grid is associated with the simulation big domain; the grid is broken into blocks, each of which is associated to a simulation SD; finally, a block is composed of threads, each of which is associated with one granular element. Figure 3b provides a visualization of this decomposition. When computing forces between spheres in the same SD, only the SD-relative position triplet is required, so that only 12 bytes of information are needed per sphere, and the representation of an SD has minimal impact on performance. Moreover, each sphere is described with uniform accuracy dependent on ψ_ℓ, which can be tuned as needed, but will always be able to represent the characteristic deformation. Since an `unsigned int` represents an SD index and each SD can contain



(a) Visualization of SD geometry



(b) CUDA thread block layout [20].

Fig. 3 The simulation big domain is split into axis aligned SDs, each containing no more than a predefined sphere count C. The 3D physical grid of SDs is mapped onto a CUDA grid, which has blocks, each of which has threads. One element in the DEM approach is handled by one CUDA thread

dozens of spheres, this representation scheme could scale to billions of particles without overflow/underflow. For a system with many extremely small particles and a long aspect ratio, any potential overflow concerns in SD index could be alleviated by using a triplet to represent an SD, similar to particle positions.

3.4 Storing contact history information for friction force computation

Storing contact history, which requires caching tangential displacements between time steps, is a prerequisite for obtaining meaningful friction forces [22]. Since in the current implementation each sphere has the same radius, one sphere can touch at most 12 other spheres at any given time [23]. Because of this constraint, each sphere needs space allocated for only 12 contacts. An array of size $12 \times N_b$ structures stores contact histories; each "history" structure is made of three pieces of information (see Table 2): a variable *active* of type `bool`, which indicates if the contact is active or not; a *partner body ID* of type `unsigned int` that stores the id of body in mutual contact with the body of interest (if such a partner body exists); and a *history vector* of type `float3` that stores the local tangential displacement.

At the beginning of a timestep, all contacts have *active* set to false. If the *partner body ID* is set to `(unsigned int) -1`, the entry is not in use. Otherwise, that entry is the index of the body that occupied that contact slot in the last time step. When a contact is detected between bodies i and j, each body undergoes the following process. Body i checks each of its entries to see if body j appears there; if so, that slot is set to *active* = `True` and the current value of the history vector is used for the process described in Eq. (2). If body j does not appear in the contact table for body i, the first empty entry is claimed for this contact pair and *active* is set to `True`. At the end of a timestep, any entries with *active* set to false have their partner body ID's set to `-1` and their history vectors zeroed out. This provides a means to hold the entire contact histories for all possible contact pairs in only $O(N_b)$ space, with each contact lookup happening in constant time. Although this data structure is designed to have a small memory footprint, it still demands up to 70% of the total memory required to carry out the simulation. This system could be extended to a system of general (polydisperse, nonspherical) particles through the use of an efficient hash map or by imposing limitations on the number of contacts any particle can experience. Such a hash map approach is feasible due to continuing advances in GPU-friendly data structures [24].

3.5 Computational flow

Simulating one timestep consists of four stages: (i) broad-phase contact detection; (ii) narrow-phase contact detection; (iii) force computation (see Sect. 2.1); and (iv) time integration. The contact detection is a rehashing of the so-called linked-cell method [25] implemented to leverage the GPU architecture [26]. Specifically, for sphere–sphere broad-phase, Chrono::Granular performs a geometric decomposition by uniformly dividing the user-specified domain into cubic SDs. These SDs are the same used in computing local coordinates (Sect. 3.3). Chrono::Granular bins spheres into SDs via the following procedure: first, each sphere determines which SDs it touches. As an SD is much larger than a sphere, a sphere can touch at most 8 SDs, which only occurs if it lies near where SDs juxtapose. This provides a mapping of the SDs that each sphere touches, which is then inverted into a mapping of what spheres each SD touches. As each sphere undergoes the same procedure and requires only knowledge of sphere and SD locations, this process involves minimal memory transactions and computational overhead. If N_{SD} denotes the number of SDs, the outcome of the broad-phase is a list collection of N_{SD} lists. List L_i contains the IDs of all the spheres that touch SD of index i. To avoid any dynamic memory allocation, it is assumed that an SD contains no more than C DEM elements. In practice we selected the parameter $C = 256$, although the limit imposed by geometric packing is lower. For the narrow phase of the contact detection, in each SD, a linear sweep is carried over each sphere to check it against every other sphere in the SD, for a total of $O(C^2)$ comparisons for each SD (see Fig. 3). Each comparison is a straightforward distance check since the particles are spherical, although different particle geometries could be modeled simply by changing their specific contact detection algorithm. Once a contact is detected, for frictionless systems, the associated contact force is computed on the spot. However, in the presence of friction, trying to peg the evaluation of both the normal and frictional forces to the narrow phase contact detection becomes unwieldy owing to high register pressure. Consequently, the frictional contact force computation process is split into two separate steps with the contact map described in conjunction with Eq. (2) used to keep track of contact information from time step to time step.

The current implementation supports four time integrators, see Table 3. Explicit Euler is the simplest; it has modest stability properties and exists largely for legacy reasons. Extended Taylor brings in the next Taylor series term ($\frac{1}{2}\boldsymbol{a}\,(\Delta t)^2$) into the position-level update [27]. Centered Difference was proposed by the original DEM method [28] and displays improved stability for minimal overhead. Chung's method [29] features strong numerical damping and good stability, but requires higher memory overhead to cache data from the previous time step.

3.6 Co-simulation

Chrono::Granular is designed for dual use: it can work as a standalone granular dynamics simulation engine; or it can be used in cosimulation mode with any other simulation engine

Table 3 Summary of time integrators in Chrono::Granular. The order-of-accuracy (OOA) is listed for position and velocity variables. Notation used: P-OOA – Position OOA; V-OOA – Velocity OOA

Name	P-OOA	V-OOA	Notes
Explicit Euler	1	1	Simplest, modest stability
Extended Taylor	2	1	Similar to Euler, higher order
Centered Difference	2	2	Velocities are half-timestep off of positions
Chung	2	2	Requires caching old accelerations

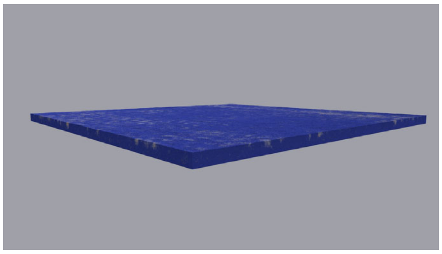
with a C++ interface. The cosimulation coupling draws on a force–displacement pairing paradigm [30] in which an object, e.g., a wheel, is presented to Chrono::Granular as a triangle mesh with specified position, orientation, translational and angular velocities. This kinematic information is used to establish boundary conditions for Chrono::Granular. At each step of the cosimulation cycle, Chrono::Granular reports back kinetic information, i.e., rigid-body forces and torques on the triangle mesh. The third-party simulation engine is then responsible for advancing its system's state and subsequently update the kinematic state of the mesh shared with Chrono::Granular.

The cosimulation mode draws on a second contact detection algorithm, also implemented on the GPU, used to resolve the contact between external meshes and the Chrono::Granular elements. This second contact detection algorithm scaffolds on the existing sphere–sphere collision algorithm. In broad-phase, mesh triangles are paired against the same grid of SDs used for spheres. Given its own GPU thread, each triangle is checked for a few convenient cases, like being confined to a single SD or a single column of SDs. In these nice cases, the implementation quickly and iteratively registers all SDs touched by this triangle; else, the triangle undergoes a more tedious check for contacts with all SDs in its axis-aligned bounding box. Given one GPU block per SD, narrow-phase then identifies contact pairs and parameters by checking all registered sphere–triangle pairs within each SD. This contact check consists of projecting the sphere center to the plane of the triangle, snapping it to the nearest point on the triangle, and then checking the distance from that point to the center of the sphere against the sphere radius R as in [31].

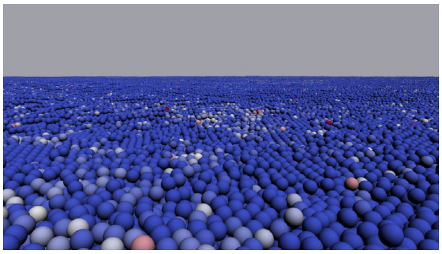
4 Scaling analysis

A scaling analysis of the code has been performed for three generations of Nvidia hardware: Pascal (GTX 1080 and Tesla P100), Volta (Tesla V100), and Turing (RTX 2080Ti and Titan RTX). The benchmark problem was that used in [32–34]: increasingly more elements were dropped into a box and allowed to settle. The registered time is associated with this settling process and the results are reported for the frictionless case in Fig. 5. The spheres had radius 5 mm, density 2.5 g/cm^2, and spring constant 1×10^8 g/s^2 or 1×10^5 N/m; the simulated time was 1 s; the simulation timestep was 2×10^{-5} s; the integrator used was Explicit Euler. The problem domain was a square patch of dimension $L \times L \times 1$ m, where, in order to accommodate an increasing number of elements, L ranged from 0.2 m to 2.7 m. A snapshot of the elements in the box is shown in Fig. 4a, with a zoom-in in Fig. 4b.

The purpose of this scaling analysis was threefold: estimating how the code scales; understanding how varying generations of Nvidia graphics cards impact the speed of simulation; and understanding how large of a problem each GPU card can solve and what happens upon



(a) 715 million frictionless bodies settling on a Tesla V100



(b) Magnified version of Fig. 4a, color is particle velocity.

Fig. 4 Snapshots, settling simulation for scaling analysis via Chrono::Granular (Color figure online)

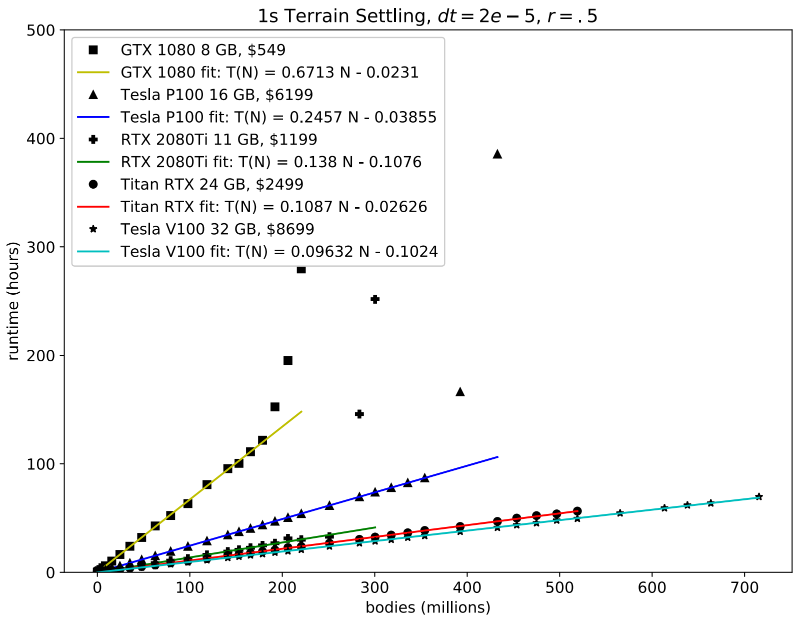


Fig. 5 Scaling analysis of a one second terrain settling simulation showing linear scaling of the implementation in Chrono::Granular, demonstrated for a collection of GPU cards. As expected, the more SMs (see Fig. 3b), the faster the simulation; and, the larger the memory, the larger the problem size for which the GPU–CPU memory paging kicks in (Color figure online)

the GPU memory being exhausted. On each card, the code exhibits linear scaling with problem size, up until the device memory is exhausted. For example, on a GTX 1080 with 8 GB device memory, the code scales linearly up to about 180 million bodies and appears to scale quadratically after that point. The superlinear scaling kicks in when the code exhausts the device's onboard memory and the CUDA runtime begins paging in and out. Unsuccessful attempts were made to optimize the code to handle these paging scenarios; at the time of development this was a new feature with very little documentation available. As the paging technology improves, looking further into this aspect is certainly warranted, especially in context of multi-GPU solutions. One reason to look into either optimizing the paging via the use of CUDA streams or into using multiple GPUs is that keeping the simulation on one system leads to manifestly higher bandwidths and lower latencies than one enjoys in a distributed-memory, MPI solution. For instance, looking at Japan's K-supercomputer used in [14], based on information reported in [35], the bandwidth at which information is exchanged between MPI ranks is of the order 4 GB/s and latencies of the order 4×10^{-3} s. This is to be compared to 900 GB/s and 4×10^{-7} s when moving data on the GPU; or 10–16 GB/s and 1×10^{-5} s when moving data between the host and device over a PCI bus or NVLINK interconnect.

Looking beyond the GTX 1080 card, the same scaling and paging occur on the other graphics cards at values commensurate to their memory capacity. For newer cards these results can likely be improved by memory "hints"; furthermore, cards with more device memory can simply run bigger problem sizes. In the largest case, an Nvidia Tesla V100 with 32 GB of device memory can scale up to 715 million frictionless spheres. As each sphere has 3 translational degrees of freedom, this provides a system with over 2 billion degrees of freedom on a single GPU. For this scenario, it takes 75 hours of simulation time to capture 1 s of system dynamics. Although the large body count is impressive, the salient point is that the implementation scales linearly and a problem with one million bodies concludes 700 times faster.

5 Numerical experiments

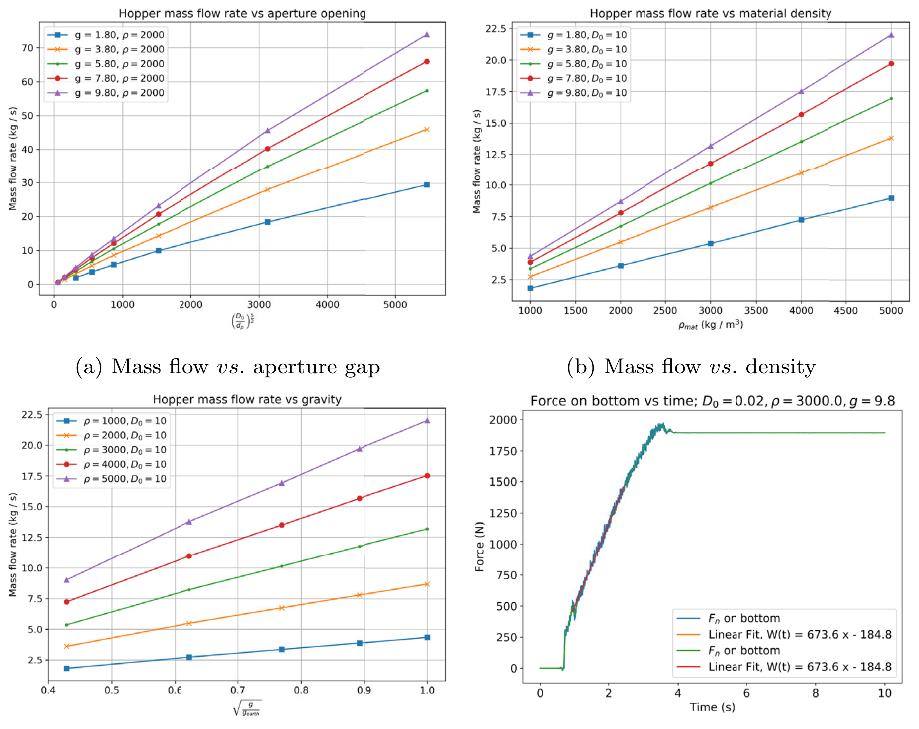
5.1 Hopper flow

The phenomenon of granular discharge, i.e., material flowing out of a hopper, is common in many industrial applications. Oftentimes, the flow is restricted through an orifice of diameter significantly larger than the particle size. Experimentally, these flows have been studied extensively, with the most famous result being the "Beverloo" equation [36] relating mass flow rate $\dot{M}$ to gravity g, particulate density ρ_b, aperture diameter D_0, and particle diameter d_p via two empirical constants, C and k, that are material specific:

$$\dot{M} = C\rho_b\sqrt{g}\left(D_0 - kd_p\right)^{\frac{5}{2}} \quad . \tag{7}$$

The Beverloo empirical relation has been shown to describe the flow rate relatively well within a range of the parameters. Little is known though about the dependence of C and k on material parameters, but common intuition is that $k \approx 1$ (since $D_0 - d_p$ is the "effective" aperture gap). This relation also assumes that $D_0 \gg d_p$, so that the boundary effects at the outlet can be neglected. In the regime where the Beverloo law is assumed to hold, one would expect the following macroscale relations to also hold, where "$\sim$" represents a linear correspondence: $\dot{M} \sim \rho_b$, $\dot{M} \sim \sqrt{g}$, and $\dot{M} \sim (D_0/d_p)^{\frac{5}{2}}$. Note that the last linear dependency is somewhat contrived and is expected to hold only when $k \approx 1$ and $D_0 \gg d_p$.

A series of numerical experiments were conducted using Chrono::Granular to gauge its predictive attributes. The goal was to understand whether by tracking the motion of each element in the problem the simulation can capture the macroscale emergent behavior of the granular material. We explored whether the Beverloo macroscale relations also hold in simulation by sweeping over the parameters with which the flow rate should have known dependency: density, gravity, and aperture gap. The results of these simulations are summarized in Fig. 6. It appears that the scalings for gravity and density are linear, with the scaling for aperture gap showing dependency close, but not quite linear and thus coming in line with



(a) Mass flow *vs.* aperture gap (b) Mass flow *vs.* density

(c) Mass flow *vs.* gravity (d) Example of flow rate used to compute $\dot{M}$

Fig. 6 Hopper mass flow rate experiments: numerical study of the $\dot{M} \sim \rho_b$, $\dot{M} \sim \sqrt{g}$, and $\dot{M} \sim (D_0/d_p)^{\frac{5}{2}}$ dependencies. Simulation setup: 240,306 spheres of *diameter* $d_p = 8$ mm; integration time step $\Delta t = \times 10^{-5}$ s; amount of simulated time 10 s; simulation run time 130 s; GPU used was Nvidia GTX 1080 Ti (Color figure online)

theoretical predictions. Finally, the results in Fig. 6d come in line with the expectation that an hourglass has a constant rate of discharge for most of its operation, i.e., between 1.5 and 4 seconds. Note that these simulations were all frictionless. All hopper numerical experiments used an integration step size of 10^{-5} s and simulated 10 s of the system's dynamics.

In order to further explore the relationship between flow rate and aperture gap, a "helper" variable C' is introduced as

$$C' = C\left(D_0 - kd_p\right)^{\frac{5}{2}} = \frac{\dot{M}}{\rho_b\sqrt{g}}. \tag{8}$$

Given that C is a constant, C' should vary as $\left(D_0 - kd_p\right)^{\frac{5}{2}}$, or equivalently $C'^{\frac{2}{5}}$ should vary linearly with D_0 for a fixed value of d_p. By creating linear fits between D_0 and $C'^{\frac{2}{5}}$, one can compute estimates for empirical parameters C and k. Using the simulation results previously described, C' was estimated for each instance, raised to power $\frac{2}{5}$, and plotted as a function of D_0 in Fig. 7 (d_p was held constant). This chart is an alternative representation of Fig. 6a, with the added advantage that estimates for C and k can be evaluated based on the slope and intercept of the linear fit.

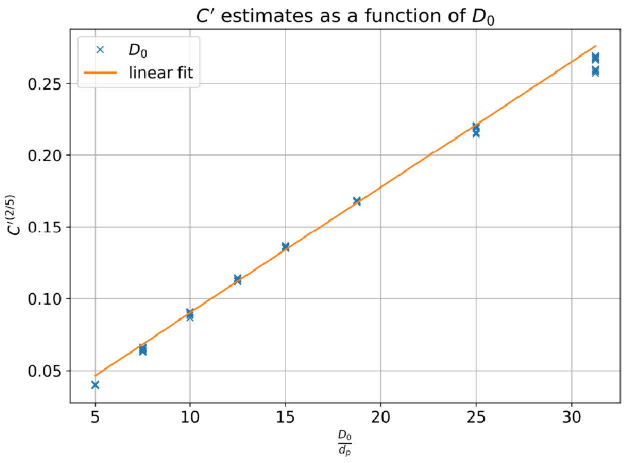


Fig. 7 Plot of $C'^{\frac{2}{5}}$ *vs.* $\frac{D_0}{d_p}$ to explore Beverloo reliability and coefficients

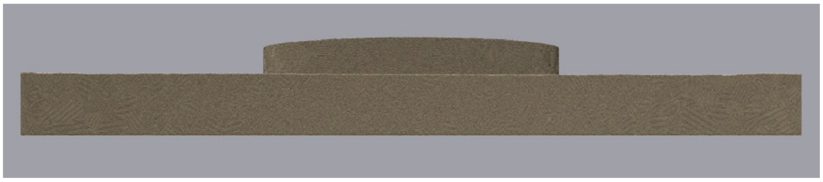


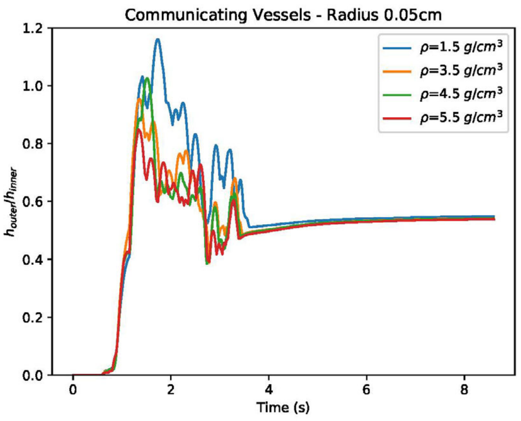
Fig. 8 End state of simulation experiment for the communicating vessels test with 114,106,578 elements of radius 0.5 mm. The data indicates that $\Delta h \neq 0$. Amount of time simulated was 9 s. Simulation run time took approximately 16 days

5.2 The "communicating vessels" test

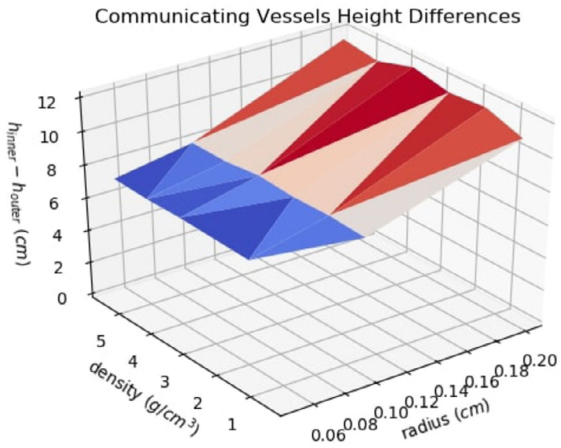
This experiment seeks to answer the following question: does frictionless granular material obey the law of communicating vessels? For fluids, this law states that two communicating vessels display at equilibrium the same fluid surface height. This is a consequence of taking the velocity to be constant in the Navier–Stokes momentum balance equation. In the proposed experiment, a hollow cylinder of interior radius 25 cm and overall radius 27.5 cm is placed inside a containing box with a square base of side length 100 cm. First, while resting on the bottom of the box, the cylinder is filled up with granular material to a height $h_0 = 100$ cm. After the granular material settles, the cylinder is lifted to create a 4 cm gap with the floor. The granular material flows out of the bottomless cylinder to reach equilibrium after approximately 5 seconds (the length of the experiment was 9 seconds). The question is whether, just like for fluids, the heights of the material inside and outside of the cylinder will match at equilibrium. Such a steady state solution is shown in Fig. 8 with the cylinder removed to reveal the material column. The metrics of interest are the ratio between material height h_{inner} inside the cylinder and the height of material h_{outer} outside of the cylinder, as well as the difference $\Delta h \equiv h_{\text{inner}} - h_{\text{outer}}$.

Other aspects of interest pertain to how the radius and density of the granular material affect the steady state solution. Figure 9a shows how the ratio $h_{\text{outer}}/h_{\text{inner}}$ evolves over time in one of the experiments. At first, there is no material in the outer chamber until the moment

Fig. 9 Communicating vessels results (Color figure online)



(a) Time evolution of the ratios of the heights of material inside and outside the cylinder with a radius of 0.05 cm.



(b) Resulting height differences for the inner and outer chambers of the communicating vessels test sweep. The surface is always above the zero (horizontal) plane, which indicates that in this study the height in the two vessels was never the same.

when the gap between the cylinder and the floor is large enough for the first layer of particles to fit under the lip of the cylinder and spill out. The ratio eventually settles to the same value, not the 1 : 1 of fluid hydrostatics, regardless of the density of the material the spheres are made of (all experiments carried out with a monodisperse material).

The results of a sweep over particle radius and density are presented in Fig. 9b. They reveal a coherent, bilinear trend throughout the data. The particle density has no significant impact on the height differential. What does significantly impact Δh is the particle radius. Indeed, the height differential between vessels decreases significantly with particle size. However, even at a radius of 500 microns, or alternatively 1.25% of the height of the gap made with the floor, there is a large differential Δh of roughly 6 cm. Thus, even for the optimal case of frictionless spherical particles, granular material does not obey the law of communicating vessels owing to its ability to jam and stack. This phenomenon is reduced, but not eliminated, as the particle radius decreases.

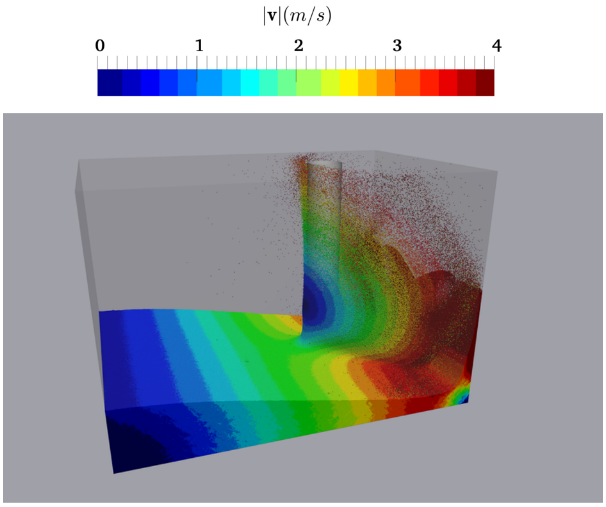


Fig. 10 Granular dam break with material flow impacting cylinder. Granular material colored according to velocity magnitude (Color figure online)

Table 4 Radii and corresponding particle counts for the dam break simulations

Particle radius (cm)	N_b
2.0	292,625
1.0	2,358,036
0.5	18,939,589

5.3 The "dam break with obstacle" test

We borrow a common test from the computational fluid dynamics literature – a dam-break simulation – and change it slightly by including a cylindrical obstacle placed in the way of the wave. A discussion of this experiment for the fluid side is provided in [37]. Herein, we focus exclusively on the granular material physics.

Granular material is initially settled for 1 second in a 2 m × 4 m × 4 m volume at the front of a 6 m × 4 m × 4 m containing box. The cylinder had a radius of 0.3 m and it was placed 5 m from the left wall of the container. After the granular material settles on the left side of the container, the plane holding back the material is removed and the material is left to collapse and "flow" for 9 seconds. As the granular material front hits the cylinder (see Fig. 10), the force on the obstacle is recorded as a function of time. The force exerted on the cylinder should depend on the material's bulk properties, e.g., density, and, *asymptotically*, it should not depend on simulation or particulate parameters, e.g., the simulation time step and particle radius, respectively. Moreover, we hypothesize that since the material flows quickly, its motion is largely "inertial" and governed by particle density more than geometric locking or friction. As such, we posit that the force exerted on the obstacle should have a linear dependence on element density – the higher the density, the larger the force exerted on the obstacle. A set of simulations were conducted for the dam break with and without friction ($\mu = 0.5$) for spheres of three different sizes. Sizes along with element counts are provided in Table 4.

Figure 11 shows the force on the cylinder in the direction of flow scaled by element density. First, we note that all frictionless numerical experiments (left column) yield the same characteristics: peak in force as the material first washes over the cylinder, a sharp

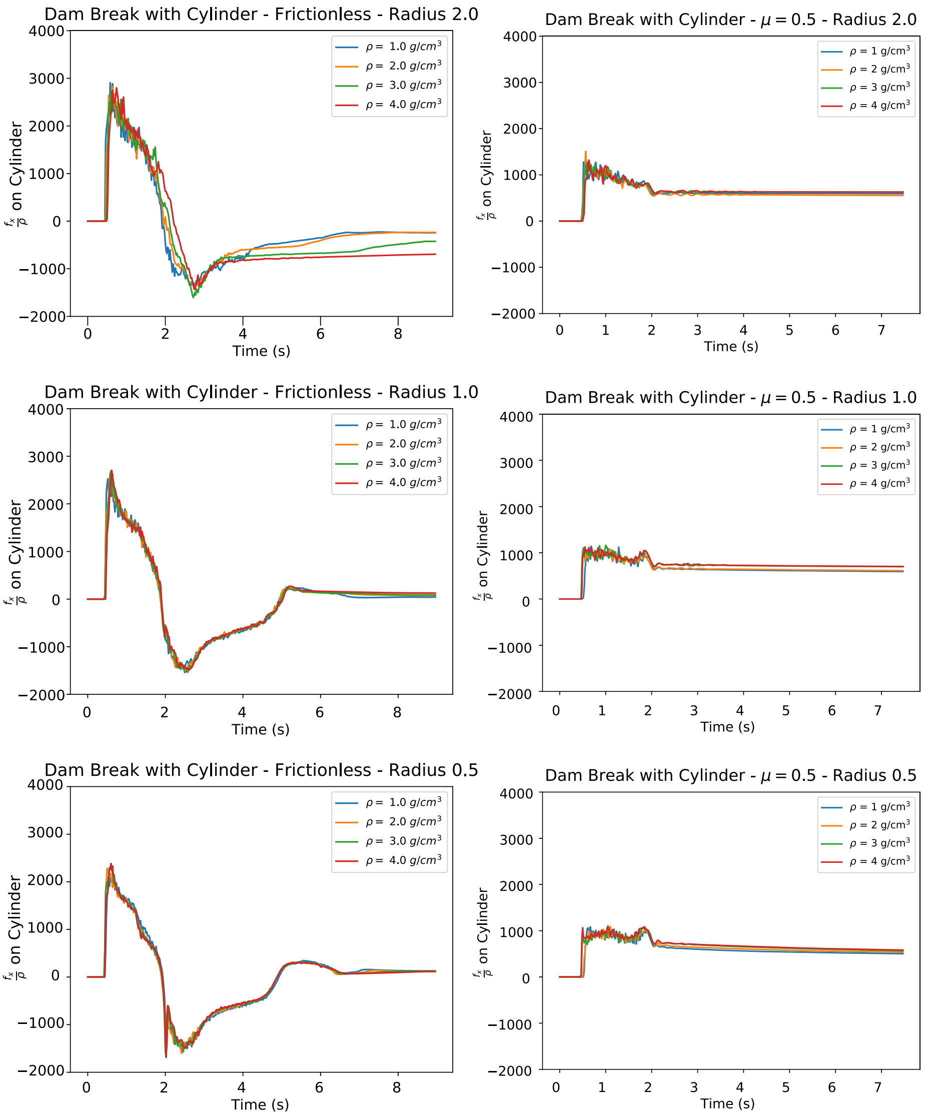


Fig. 11 Comparison of cylinder forces in the dam break test for varying densities and particle sizes without friction (left column), and with friction (right column). These simulations were all run with a step size of $\Delta t = \times 10^{-5}$ s. Regardless of element density, the simulation for the frictionless case required 37,804 s to capture the dam break dynamics for 9 s using 18,939,589 elements (Color figure online)

drop in force which reverses direction as the material splashes back, and finally a leveling-off as the material settles. More than just matching the same evolution pattern, these results also match quantitatively, with nearly the same extrema. It is noted that for large particle size (top picture in the left column), the net force on the cylinder does not zero out, a consequence of the geometric locking. Indeed, fine particles lead to an overall behavior similar to that of a fluid (bottom picture in the left column), while larger, bulkier particles lock up, particularly so when the density of the granular material is large. The right column in Fig. 11 reports results when friction was present in the simulation. The high transients are eliminated, the motion is more subdued. The relatively large value for the friction coefficient ($\mu = 0.5$) arrests the relaxation of the granular material in a configuration that yields a nonnegligible force on the cylinder.

6 Discussion

The main goal of this contribution is to outline a solution approach that opens the door to large granular dynamics simulations on affordable hardware. This effort stopped short of surpassing the state-of-the-art, i.e., a 2.4 billion element geomechanics simulation out of Japan [12–14]. However, the simulations discussed herein come relatively close in size. Indeed, the settling simulations discussed in Sect. 4 reached up to 715 million bodies for the *frictionless* case; for the frictional case we settled more than 200 million bodies. We used more than 115 million bodies in the communicating vessels experiment, and 18 million bodies in the "dam brake with obstacle" test. The notable aspect though is that while the geomechanics simulations in [12–14] ran on an architecture that afforded more than 130,000 cores on what used to be the fastest supercomputer in the world, the results reported here ran on one GPU.

The present work is placed in a broader context through a literature overview in which the focus is on problem sizes, as they have been defining the state of the art in various application areas. In [38], benchmark simulations were run for 2D systems with 200,000 elements using Swiss-T1, a 64-node supercomputer. In [39], a mixing via tumbling mill of 1,000,000 spheres required one week to simulate a 1.5 s rotation of the mixer on a cluster with 32 processors. In [40], an MPI-based implementation scaled up to 1,000,000 particles using up to 16 processors. In [41], the problem size for a hopper simulated via DEM went up to 400,000 bodies. The simulation drew on MPI [42] and a cluster with 36 processors. In [43], 150,000 glass beads were considered in the "high fill" scenario for a spheronizer simulation. Ten seconds of simulation required 375 hours of run time on a Dell SC1425 cluster with 16 Intel Xeon (3.6 GHz) processors. In [44], a 81,000-body simulation took 35 days on a 32-core architecture managed under MPI to simulate 120 s of system dynamics. In [34], the authors switched to a multi-GPU solution for systems with more than one million particles owing to GPU memory exhaustion; a settling simulation with 10 million bodies was run with up to 32 GPUs that were coordinated via MPI. The one million body simulation in [34] was carried out on one GPU using a monodisperse systems. In [45], the number of bodies was 130,000. Results pertaining powder compaction simulation are reported in [46]. Therein, the authors use mixtures of spheres of up to three different radii. Strong scaling analysis results were presented for the dynamics of one million bodies using from four up to 20 GPUs. A compaction analysis was carried out using up to 563,000 elements. In [47] the authors ran powder simulations. The number of particles in the simulation was 4,030,000. For a 2,800,000 element system run on Nvidia GTX 1080Ti GPU, the simulation took roughly 0.03 s per time step and certain measures were taken to shorten the computation, which would otherwise have required 32 days to complete. In [48], a DEM solver running on the GPU was stated to handle systems with 60 million elements. Several simplifications were made to reach this particle count, e.g., the arithmetic was carried out in single precision and, as the authors indicate, this "limits the range of values in a single calculation to $< 1 \times 10^{-6}$." This is, however, hardly sufficient, given that the deformations in granular material are of the order 10^{-13} m and lower. Therein, the normal force model could only be Hookean; moreover, the friction force model did not keep track of history, which is known to lead to inaccurate results, see, for instance, [22]. Another issue with the friction force model was that the friction force "opposes the motion of the two particles." This decision will prevent the apparition of a friction force without relative velocity since with no relative velocity, the direction of the force is undefined. As such, granular material could not pile, for instance. The issue with simplifying the friction force model by dropping the history and not handling the stick mode is also encountered in [49], where the implementation is reported to handle approximately one million spheres that are clumped in sets of four to form 256,000 million aggregate bodies of a more complex shape. Shifting the discussion to terradynamics applications, the number of elements in the simulation is typically smaller. In [50], a set of four deformable wheels interacts with a granular terrain made up of 900,000 elements. The 7.64 s long simulation of the vehicle operating on granular terrain required 5.5 days of compute time. A similar problem was analyzed in [51], where the terrain was made up of 90,304 spheres. The settling of the particles was reported as taking 46 hours of simulation time, while the rolling of one tire for approximately 5 s took approximately 52 hours. In [52], the DEM particle was obtained as the union of three spheres of identical radii. A wheel–terrain interaction problem was simulated using 392,049 such particles; no simulation times were reported.

7 Conclusions. Directions of future work

This contribution describes an approach that extends the problem size in DEM simulations of practical relevance run on commodity hardware. Using one GPU card, for frictionless DEM relevant in the simulation of particle suspensions, we report numerical experiments with more than 700 million elements. In the presence of friction, the problem size drops to 220 million elements, a consequence of the need to store more state information, e.g., body orientation and contact history. In both cases, the 3D system dynamics is carried out using more than one billion generalized coordinates to capture the time evolution of the distinct elements. Given that the software implementation of the methodology described scales linearly with problem size, problems that are smaller run correspondingly faster, to the point where they open the door to parameter sweeps like those carried out herein for three numerical experiments: hopper flow, communicating vessels, and dam break with obstacle.

The ability to handle large problem sizes and the linear scaling attribute are traced down to the following aspects: a process of adimensionalization; use of local coordinate representation; parsimonious use of memory via data types that require few bytes; and judicious use of memory bandwidth and other GPU resources such as shared memory and registers. In particular, we highlight the use of integer type variables for positioning the discrete elements in the simulation space. This allowed for a smaller memory footprint and shorter variable access times as the amount of data moved in the unit of time improved twofold relative to the double precision alternative. With slight alterations to the underlying assumptions, the key points of this work apply to a wider variety of discrete systems of pairwise interactions, e.g., molecular dynamics or Smoothed Particle Hydrodynamics.

Looking ahead at the computing framework, ongoing work seeks to generalize particle geometries to polydisperse spheres. A second thrust is aimed at support for 3D ellipsoids. A third one concentrates on improving the friction model insofar as the rolling and spinning friction are concerned. Finally, an ongoing effort seeks to use a homogenization approach [53] to treat problems when the DEM body count is too large for a fully resolved approach, and a continuum representation of the granular flow becomes more desirable.

The software that implements the methodology described and used to generate the results reported herein is called Chrono::Granular. It is open source and freely available online under a permissive BSD3 license for unfettered use, modification, and release. It is part of the Project Chrono simulation platform [54] that can be cloned/downloaded from a public GitHub repository [55].

Acknowledgements The modeling/numerical method development associated with this project was funded through Army Research Office grant W911NF1910431. The hardware assets used herein have been available through Army Research Office grant W911NF1810476. The software development effort associated with this project was funded through National Science Foundation grant CISE—1835674.

References

1. Richard, P., Nicodemi, M., Delannay, R., Ribiere, P., Bideau, D.: Slow relaxation and compaction of granular systems. Nat. Mater. **4**(2), 121–128 (2005)
2. Moreau, J.J.: Unilateral contact and dry friction in finite freedom dynamics. In: Moreau, J.J., Panagiotopoulos, P.D. (eds.) Nonsmooth Mechanics and Applications, pp. 1–82. Springer, Berlin (1988)
3. Cundall, P., Strack, O.: A discrete element model for granular assemblies. Geotechnique **29**, 47–65 (1979)
4. Pazouki, A., Kwarta, M., Williams, K., Likos, W., Serban, R., Jayakumar, P., Negrut, D.: Compliant versus rigid contact: a comparison in the context of granular dynamics. Phys. Rev. E **96**, 042905 (2017)
5. Goyon, J., Colin, A., Ovarlez, G., Ajdari, A., Bocquet, L.: Spatial cooperativity in soft glassy flows. Nature **454**(7200), 84–87 (2008)
6. Jop, P., Forterre, Y., Pouliquen, O.: A constitutive law for dense granular flows. Nature **441**(7094), 727–730 (2006)
7. Kamrin, K., Koval, G.: Nonlocal constitutive relation for steady granular flow. Phys. Rev. Lett. **108**(17), 178301 (2012)
8. Iwashita, K., Oda, M.: Rolling resistance at contacts in simulation of shear band development by DEM. J. Eng. Mech. **124**(3), 285–292 (1998)
9. Silbert, L., Ertaş, D., Grest, G., Halsey, T., Levine, D., Plimpton, S.: Granular flow down an inclined plane: bagnold scaling and rheology. Phys. Rev. E **64**(5), 051302 (2001)
10. da Cruz, F., Emam, S., Prochnow, M., Roux, J.N., Chevoir, F.: Rheophysics of dense granular materials: discrete simulation of plane shear flows. Phys. Rev. E **72**, 021309 (2005)
11. Parteli, E., Poschel, T.: Particle-based simulation of powder application in additive manufacturing. Powder Technol. **288**, 96–102 (2016)
12. Furuichi, M., Nishiura, D., Asai, M., Hori, T.: Poster: the first real-scale DEM simulation of a sandbox experiment using 2.4 billion particles. In: Supercomputing Conference (2017)
13. Furuichi, M., Nishiura, D., Kuwano, O., Bauville, A., Hori, T., Sakaguchi, H.: Arcuate stress state in accretionary prisms from real-scale numerical sandbox experiments. Sci. Rep. **8**, 12 (2018)
14. Nishiura, D., Sakaguchi, H., Yamamoto, S.: Multibillion particle DEM to simulate centrifuge model tests of geomaterials. In: Physical Modelling in Geotechnics, Volume 1: Proceedings of the 9th International Conference on Physical Modelling in Geotechnics (ICPMG 2018), London, United Kingdom, July 17–20, 2018, p. 227. CRC Press, Boca Raton (2018)
15. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: TOP500 Supercomputer Site. http://www.top500.org
16. Zhou, Y.C., Xu, B.H., Yu, A.B., Zulli, P.: An experimental and numerical study of the angle of repose of coarse spheres. Powder Technol. **125**(1), 45–54 (2002)
17. Oda, M., Iwashita, K.: Study on couple stress and shear band development in granular media based on numerical simulation analyses. Int. J. Eng. Sci. **38**(15), 1713–1740 (2000)
18. Ai, J., Chen, J.-F., Rotter, M., Ooi, J.: Assessment of rolling resistance models in discrete element simulations. Powder Technol. **206**(3), 269–282 (2011)
19. Geer, S., Bernhardt-Barry, M., Garboczi, E., Whiting, J., Donmez, A.: A more efficient method for calibrating discrete element method parameters for simulations of metallic powder used in additive manufacturing. Granul. Matter **20**(4), 77 (2018)

20. NVIDIA Corporation: Compute unified device architecture toolkit documentation (2019). https://docs.nvidia.com/cuda
21. Negrut, D., Serban, R., Li, A., Seidl, A.: Unified Memory in CUDA 6.0: a brief overview of related data access and transfer issues. Technical Report TR-2014-09, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, (2014). https://sbel.wisc.edu/wp-content/uploads/sites/569/2018/05/TR-2014-09.pdf
22. Fleischmann, J., Serban, R., Negrut, D., Jayakumar, P.: On the importance of displacement history in soft-body contact models. J. Comput. Nonlinear Dyn. **11**(4), 044502 (2016)
23. Musin, O.R.: The kissing problem in three dimensions. arXiv Mathematics e-prints (2004). math/0410324
24. Green, O.: Hashgraph – scalable hash tables using a sparse graph data structure (2019)
25. Hockney, R., Eastwood, J.: Computer Simulation Using Particles. CRC Press, Boca Raton (1988)
26. Mazhar, H., Heyn, T., Negrut, D.: A scalable parallel method for large collision detection problems. Multibody Syst. Dyn. **26**, 37–55 (2011). https://doi.org/10.1007/s11044-011-9246-y
27. Hairer, E., Norsett, S., Wanner, G.: Solving Ordinary Differential Equations I: Nonstiff Problems. Springer, Berlin (2009)
28. Cundall, P.: Formulation of a three-dimensional distinct element model–Part I. A scheme to detect and represent contacts in a system composed of many polyhedral blocks. Int. J. Rock Mech. Min. Sci. Geomech. Abstr. **25**(3), 107–116 (1988)
29. Chung, J., Lee, J.M.: A new family of explicit time integration methods for linear and non-linear structural dynamics. Int. J. Numer. Methods Eng. **37**(23), 3961–3976 (1994)
30. Schweizer, B., Li, P., Lu, D.: Explicit and implicit cosimulation methods: stability and convergence analysis for different solver coupling approaches. J. Comput. Nonlinear Dyn. **10**(5), 051007 (2015)
31. Ericson, C.: Real Time Collision Detection. Morgan Kaufmann, San Francisco (2005)
32. Zhou, Z., Pinson, D., Zou, R., Yu, A.: Discrete particle simulation of gas fluidization of ellipsoidal particles. Chem. Eng. Sci. **66**(23), 6128–6145 (2011)
33. Hou, Q., Zhou, Z., Yu, A.: Micromechanical modeling and analysis of different flow regimes in gas fluidization. Chem. Eng. Sci. **84**, 449–468 (2012)
34. Gan, J., Zhou, Z., Yu, A.: A GPU-based DEM approach for modeling of particulate systems. Powder Technol. **301**, 1172–1182 (2016)
35. University of Tennessee: High Performance Computing Challenge Benchmark (2019). http://icl.cs.utk.edu/hpcc/hpcc_results_lat_band.cgi
36. Mankoc, C., Janda, A., Arevalo, R., Pastor, J., Zuriguel, I., Garcimartín, A., Maza, D.: The flow rate of granular materials through an orifice. Granul. Matter **9**(6), 407–414 (2007)
37. Rakhsha, M., Kelly, C., Olsen, N., Serban, R., Negrut, D.: Multibody dynamics vs. fluid dynamics: two perspectives on the dynamics of granular flows. J. Comput. Nonlinear Dyn. (2020). https://doi.org/10.1115/1.4047237
38. Cleary, P., Sawley, M.: DEM modelling of industrial granular flows: 3D case studies and the effect of particle shape on hopper discharge. Appl. Math. Model. **26**(2), 89–111 (2002)
39. Bertrand, F., Leclaire, L., Levecque, G.: DEM-based models for the mixing of granular materials. Chem. Eng. Sci. **60**(8–9), 2517–2531 (2005)
40. Fleissner, F., Eberhard, P.: Load balanced parallel simulation of particle-fluid DEM-SPH systems with moving boundaries. In: Parallel Computing: Architectures, Algorithms and Applications, vol. 48, pp. 37–44 (2007)
41. Kloss, C., Goniva, C., Hager, A., Amberger, S., Pirker, S.: Models, algorithms and validation for open-source DEM and CFD–DEM. Prog. Comput. Fluid Dyn. **12**(2–3), 140–152 (2012)
42. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2nd edn. MIT Press, Cambridge (1999)
43. Bouffard, J., Bertrand, F., Chaouki, J., Dumont, H.: Discrete element investigation of flow patterns and segregation in a spheronizer. Comput. Chem. Eng. **49**, 170–182 (2013)
44. Alizadeh, E., Bertrand, F., Chaouki, J.: Comparison of DEM results and Lagrangian experimental data for the flow and mixing of granules in a rotating drum. AIChE J. **60**(1), 60–75 (2014)
45. Hou, Q., Dong, K., Yu, A.: DEM study of the flow of cohesive particles in a screw feeder. Powder Technol. **256**, 529–539 (2014)
46. He, Y., Evans, T., Yu, A., Yang, R.: A GPU-based DEM for modeling large scale powder compaction with wide size distributions. Powder Technol. **333**, 219–228 (2018)
47. Toson, P., Siegmann, E., Trogrlic, M., Kureck, H., Khinast, J., Jajcevic, D., Doshi, P., Blackwood, D., Bonnassieux, A., Daugherity, P.D., et al.: Detailed modeling and process design of an advanced continuous powder mixer. Int. J. Pharm. **552**(1–2), 288–300 (2018)
48. Govender, N., Wilke, D., Kok, S.: Blaze-DEMGPU: modular high performance DEM framework for the GPU architecture. SoftwareX **5**, 62–66 (2016)

49. Longmore, J.-P., Marais, P., Kuttel, M.M.: Towards realistic and interactive sand simulation: a GPU-based framework. Powder Technol. **235**, 983–1000 (2013)
50. Recuero, A.M., Serban, R., Peterson, B., Sugiyama, H., Jayakumar, P., Negrut, D.: A high-fidelity approach for vehicle mobility simulation: nonlinear finite element tires operating on granular material. J. Terramech. **72**, 39–54 (2017)
51. Zhao, C.-L., Zang, M.-Y.: Application of the FEM/DEM and alternately moving road method to the simulation of tire-sand interactions. J. Terramech. **72**, 27–38 (2017)
52. Johnson, J.B., Kulchitsky, A.V., Duvoy, P., Iagnemma, K., Senatore, C., Arvidson, R.E., Moore, J.: Discrete element method simulations of Mars exploration rover wheel performance. J. Terramech. **62**, 31–40 (2015)
53. Dunatunga, S., Kamrin, K.: Continuum modelling and simulation of granular flows through their many phases. J. Fluid Mech. **779**, 483 (2015)
54. Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., Taylor, M., Sugiyama, H., Negrut, D.: Chrono: an open source multi-physics dynamics engine. In: Kozubek, T. (ed.) High Performance Computing in Science and Engineering. Lecture Notes in Computer Science, pp. 19–49. Springer, Berlin (2016)
55. Project Chrono: Chrono: an Open Source Framework for the Physics-Based Simulation of Dynamic Systems (2020). http://projectchrono.org. Accessed: 2020-03-03