processes

MDPI

*Article*

# Load-Balancing Strategies in Discrete Element Method Simulations

**Shahab Golshan and Bruno Blais ***

Research Unit for Industrial Flows Processes (URPEI), Department of Chemical Engineering, École Polytechique de Montréal, P.O. Box 6079, Stn Centre-Ville, Montréal, QC H3C 3A7, Canada; shahab.golshan@polymtl.ca
* Correspondence: bruno.blais@polymtl.ca

**Abstract:** In this research, we investigate the influence of a load-balancing strategy and parametrization on the speed-up of discrete element method simulations using Lethe-DEM. Lethe-DEM is an open-source DEM code which uses a cell-based load-balancing strategy. We compare the computational performance of different cell-weighing strategies based on the number of particles per cell (linear and quadratic). We observe two minimums for particle to cell weights (at 3, 40 for quadratic, and 15, 50 for linear) in both linear and quadratic strategies. The first and second minimums are attributed to the suitable distribution of cell-based and particle-based functions, respectively. We use four benchmark simulations (packing, rotating drum, silo, and V blender) to investigate the computational performances of different load-balancing schemes (namely, `single-step`, `frequent` and `dynamic`). These benchmarks are chosen to demonstrate different scenarios that may occur in a DEM simulation. In a large-scale rotating drum simulation, which shows the systems in which particles occupy a constant region after reaching steady-state, `single-step` load-balancing shows the best performance. In a silo and V blender, where particles move in one direction or have a reciprocating motion, `frequent` and `dynamic` schemes are preferred. We propose an automatic load-balancing scheme (`dynamic`) that finds the best load-balancing steps according to the imbalance of computational load between the processes. Furthermore, we show the high computational performance of Lethe-DEM in the simulation of the packing of $10^8$ particles on 4800 processes. We show that simulations with optimum load-balancing need ≈40% less time compared to the simulations with no load-balancing.

**Keywords:** discrete element method; parallel computing; load-balancing; silo; rotating drum; V blender; packed bed



**Citation:** Golshan, S.; Blais, B. Load-Balancing Strategies in Discrete Element Method Simulations. *Processes* **2022**, *10*, 79. https://doi.org/10.3390/pr10010079

Academic Editor: Joanna Wiącek

Received: 17 November 2021
Accepted: 22 December 2021
Published: 31 December 2021



## 1. Introduction

Granular materials are prevalent in nature and global industry [1–3]. Due to their abundance and wide range of applications, the study of granular flows is an active research area where multiple challenges remain unanswered [4,5]. Generally, researchers use continuum (Eulerian) or discrete (Lagrangian) approaches to model granular systems. Although computationally efficient, continuum approaches can be inaccurate because the flow of granular systems may deviate significantly from that of continuous matter [6]. The Discrete Element Method (DEM) is a Lagrangian model which simulates the motion of all the particles and their collisions with other particles and boundaries in a system [4]. Since DEM tracks every particle individually, it is more accurate than continuum-based models. This accuracy, of course, comes with a high computational cost [7].

The computational cost for the best-designed DEM codes is of the order of $\mathcal{O}(n_p \log n_p)$, where $n_p$ is the number of simulated particles [7,8]. This limits the number of particles in a simulation based on the available computational resources. In the last two decades, parallel computing has helped researchers simulate granular systems containing millions of particles [4,8–13]. To this end, several parallel DEM softwares using multiple central processing units (CPU) to carry-out a single simulation have been developed [5,9,12,14–16].

These parallel DEM softwares generally employ a single program, multiple data (SPMD) approach to parallelize the simulations. SPMD is based on spatial domain decomposition in which the simulation domain is divided between the parallel processes and communication between the processes is ensured through the Message Passing Interface (MPI) [6]. As the particles move inside the simulation domain and migrate between the subdomains, the computational load on the processes changes [10]. Migration of particles between the subdomains may lead to load imbalance, in which the computational load on the processes is significantly different. This overburdens some cores while leaving others idle and thus slows down the simulation. Load-balancing can mitigate this problem.

Load-balancing re-equalizes the computational loads by re-distributing the simulation between the processes [10]. In the DEM, load-balance is mainly interpreted as having an equal number of particles on each process [6]. Compared to Computational Fluid Dynamics (CFD) [17], less attention has been paid to load-balancing in DEM. Fleissner et al. [13,18] used an orthogonal recursive bisection domain partitioning algorithm [19] in their simulations. This algorithm recursively subdivides the simulation domain using planes and moves these planes in order to reach a homogeneous computational load [10]. LIGGGHTS uses a recursive multi-sectioning algorithm on a Cartesian simulation grid for load-balancing [20]. Cintra et al. [21,22] used a recursive coordinate bisection in a simulation of hopper discharge and landslide using the DEMOOP software. The performances of different load-balancing algorithms were compared in other works [10,11,23]. Markauskas and Kačeniauskas [23] simulated the discharge of 5.1 M spherical particles on 128–2048 cores and reported a speed-up of 1785 on 2048 cores. They observed that complex adaptations of the k-way graph partitioning method do not improve the parallel performance. Our work in Lethe-DEM [8], using a forest of tree approach for load-balancing, aims at solving the limitations and challenges of load-balancing in DEM simulations.

## 2. Problem Definition

This work focuses on the load-balancing aspects of the DEM. The fundamentals of DEM are not reviewed here. Interested readers may find the fundamentals of DEM and the details of its implementation elsewhere [8]. We use an open-source DEM software, Lethe-DEM, which is parallelized using MPI [8]. This code is based on the deal.II finite element library [24,25] and uses its particles capabilities [26]. In this code, the Hertz–Mindlin, limiting the tangential overlap and limiting the tangential force, and Hertz and Hookean contact models are implemented. In the analyses in this research, we use the Hertz–Mindlin with limiting the tangential overlap model. Lethe-DEM uses a background grid in which the particles are mapped frequently. This grid, and consequently the particles, are distributed between the processes in a parallel simulation. Lethe-DEM handles the mesh partitioning using the p4est library via deal.II [27,28]. Initially, we create a coarse triangulation (which consists of not more than 100–10,000 cells in most cases) using deal.II or GMSH [29]. Lethe-DEM replicates this triangulation on all the processes, refines it locally to reach a desired element size in a forest-of-tree manner, and partitions the final triangulation through p4est. Each subdomain in the input triangulation is assigned to a process. A halo layer of the width of a single cell surrounds each subdomain. The cells in this halo are referred to as `ghost cells` and the particles that reside in these `ghost cells` as `ghost particles`. In Figure 1 the concepts of distributed triangulation and subdomains, `ghost cells`) and `ghost particles`) are explained.

According to these definitions, we categorize the particle-particle contacts into local-local and local-ghost contacts. Local-local collisions are fully handled by the process owning the particle, while in local-ghost contacts, each process only handles the calculations on the particle which is local to it. This means that calculations of local-ghost contacts are performed on the processes for which one particle is local. As the particles move in the simulation domain and change subdomain, we need three functions to: update the owner cells (and subdomain) of the particles, create and update the ghost particles, and update the properties and location of the ghost particles.

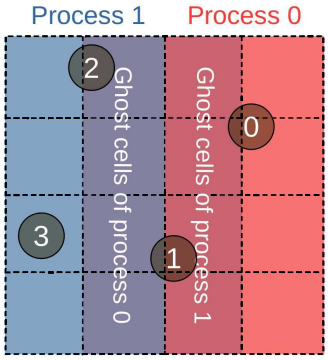


**Figure 1.** A triangulation distributed between two processes (Processes 0 and 1) and four particles. The `ghost cells` are highlighted for each process. Particles 0 and 3 are local particles for processes 0 and 1, respectively, while particle 1 is a `ghost particle` of process 1 and particle 2 is a `ghost particle` of process 0.

Load-balancing in Lethe-DEM is performed by assigning a weight to each cell according to its computational load and redistributing the cells between the processes such that all the processes have equal weight. The computational cost of some functions (for example, the function used to update the owner cells and subdomain) in the DEM software is proportional to $\mathcal{O}(n_c)$, while other functions (for example, particle-wall and particle-particle contact force) have computational costs proportional to $\mathcal{O}(n_p)$ and $\mathcal{O}(n_p \log n_p)$. Because these functions have different costs, assigning a weight to each cell based on the number of particles is a complicated task. We can use different strategies to assign the cell weight. Two intuitive strategies are linear and quadratic weighting:

$$W_c = \alpha n_p + \beta \tag{1}$$

$$W_c = \alpha {n_p}^2 + \beta, \tag{2}$$

where $\alpha$ and $\beta$ are weights of particles and cells. We investigate the performances of linear and quadratic weight models as well as the effects of $\alpha$ and $\beta$. Furthermore, load-balancing itself is an expensive operation. In other words, calling load-balancing too frequently adds extra computational cost to a simulation. Consequently, we have to call load-balancing depending on the dynamics of the granular flow. To this end, we introduce three different load-balancing schemes in Lethe-DEM:

- `single-step` (also referred to as `once` in this article): In this load-balancing strategy, we only call load-balancing once per simulation at a given iteration;
- `frequent`: In `frequent` load-balancing, the software calls load-balancing at a constant frequency (every $n_{LB}$ iterations);
- `dynamic`: In `dynamic` load-balancing, the software automatically detects if load-balancing is required by measuring the load imbalance. At a predefined frequency, the software checks the computational weights of all the processes. If the weight difference between the processes with the highest and lowest loads exceeds a threshold based on the average load (if $L_{max} - L_{min} > \epsilon L_{av}$, where $\epsilon$ is a defined threshold), load-balancing is performed. $L$ and $\epsilon$ denote the total process load (summation of the weights of the cells, $W_c$ in Equations (1) and (2), owned by each process) and a user-defined dynamic load-balancing threshold, respectively.

The goal of this study is to characterize and quantify the impact of both the weighting strategy as well as the load-balancing scheme. Using four benchmarks, namely:

- Packing of particles;
- A rotating drum;
- A silo;
- A V-blender.

We demonstrate that load-balancing can greatly reduce computational time even for simulations including a moving geometry. We quantify the load-balancing cost and identify the best load-balancing schemes for different types of granular flows frequently simulated with the DEM. Instead of focusing on the load-balance algorithms, which have been studied and compared in the literature [10,11,13,18,20–23,30], we study the weighting strategies and consistent load-balancing schemes in different scenarios that may occur in DEM simulations. This research is useful to the researchers interested in using the DEM simulations and of particle-based methods to optimize the computational costs of their parallel simulations. Furthermore, all the benchmarks and examples are accessible on the Lethe GitHub page (https://github.com/lethe-cfd/lethe/wiki, accessed on 21 December 2021).

## 3. Results and Discussion

### *3.1. Benchmark Cases*

We use four benchmark cases. The first case is the packing of particles in a rectangular box. We use this case to compare the computational performances of linear and quadratic weights as well as the load-balancing costs of large-scale simulations. The second case is the granular flow in a rotating drum. This is an example of the cases in which particles reside in a constant region of the simulation domain after the flow has reached pseudo steady-state. In the third case, we simulate the motion of particles during a silo discharge. This simulation is an example of cases in which the particles move in one direction inside the simulation domain. The last case is a V blender in which particles have a continuous reciprocating motion. We specify the physical properties in Table 1 by using the experimental research in the literature [31,32], and interested readers can find the parameter handler files of these simulations in the examples section of the Lethe Github repository: https://github.com/lethe-cfd/lethe, (accessed on 21 December 2021). Figure 2 illustrates the simulation geometries and configurations.

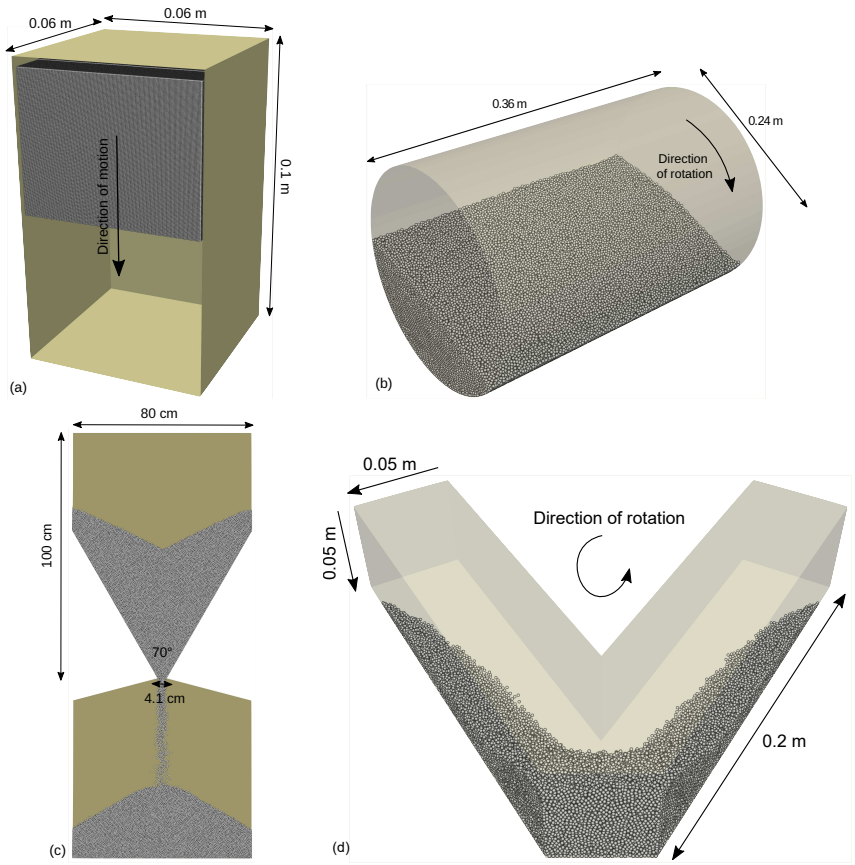


**Figure 2.** Configurations of the benchmark cases used in this work: (**a**) packing of particles in a rectangular box, (**b**) rotating drum [32], (**c**) silo filling and discharge [31], (**d**) V blender.

**Table 1.** Simulation and physical properties of the packing in rectangular box, rotating drum, silo and V-blender simulations.

| | **Packing** | **Drum** | **Silo** | **V-Blender** |
|---|---|---|---|---|
| $d_p$(mm) | 0.93, 0.43, 0.2, 0.093 | 3 | 5.83 | 1.5 |
| $n_p$ | $10^5, 10^6, 10^7, 10^8$ | $2.26 \times 10^5$ | $1.32 \times 10^5$ | $4 \times 10^4$ |
| $\rho_p$(kg/m$^3$) | 1000 | 2500 | 600 | 2000 |
| $Y$(MPa) | 100 | 100 | 5 | 10 |
| $\nu$ | 0.3 | 0.24 | 0.5 | 0.5 |
| $e$ | 0.9 | 0.97 | 0.7 | 0.7 |
| $\mu$ | 0.3 | 0.3 | 0.5 | 0.5 |
| $\mu_r$ | 0.1 | 0.01 | 0.01 | 0.01 |
| $t_f$(s) | 0.15 | 10 | 40 | 20 |
| $dt$(s) | $10^{-6}$ | $10^{-6}$ | $10^{-5}$ | $10^{-6}$ |

### *3.2. Influence of the Weighting Strategy (Linear and Quadratic)*

We compare the simulation times of the packing case with $10^5$ particles with a linear and quadratic load-balancing weighting strategy. We compare the simulation times at different ratios of $\alpha/\beta$. We select the range of 1–100 for $\alpha/\beta$ where the former value (1) corresponds to the situation in which the weight of owned cells is equal to the weight of owned particles (in favor of the number of cells), and the latter value (100) belongs to the situation where the weight of owned cells is almost negligible compared to that of the particles. We use `frequent` load-balancing with $f_{LB} = 100$ Hz, and repeat each simulation three times and report the average simulation time. Figure 3 shows the results of this comparison. Both linear and quadratic weighting strategies show the decreasing-increasing-decreasing-increasing trends. The linear weighting strategy shows two minimums at $\alpha/\beta = 15$ and 50, and the quadratic weighting strategy shows two minimums at $\alpha/\beta = 3$ and 40. The first minimums are attributed to the suitable distribution of cell-based functions, and the second minimums are attributed to the appropriate distribution of particle-based functions over the processes. The global minimum occurs at $\alpha/\beta = 50$ for linear load-balancing.

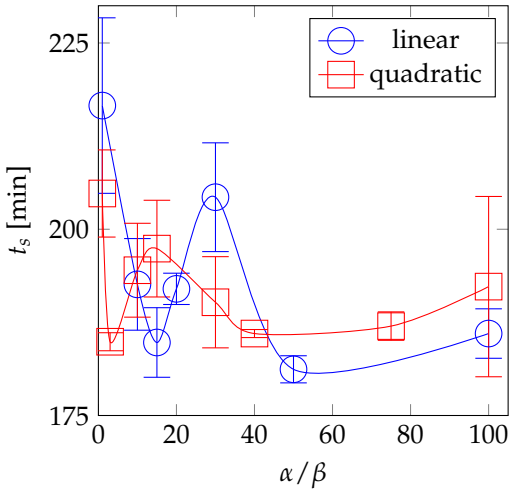


**Figure 3.** Influence of the load balancing strategy on the simulation time for the packing case. Error bars show standard deviation.

### *3.3. Packing in Box*

Figure 4a shows the comparison between the load-balancing times (for a single load-balancing operation) of four packing simulations in Table 1. We do not compare the simulation times of these simulations since we have performed strong and weak scaling analyses of Lethe-DEM in previous research [8]. We perform the simulations of the

packing of $10^5$, $10^6$, $10^7$, and $10^8$ particles on 8, 48, 480, and 4800 processes, respectively ($n_p/n_c \approx 20{,}000$). This figure shows that the cost of load-balancing increases as the number of particles increases in a simulation. Figure 4b shows the load-balancing times (for a single load-balancing operation) for the simulation of packing of 1 M particles on 32, 64, 96 and 128 processes. The cost of load-balancing decreases as the number of processes increases in the simulation. This decrease is super-linear. This indicates that the cost of load-balancing does not scale linearly with the number of particles per core. Consequently, load-balancing becomes more and more viable as the number of cores is increased.

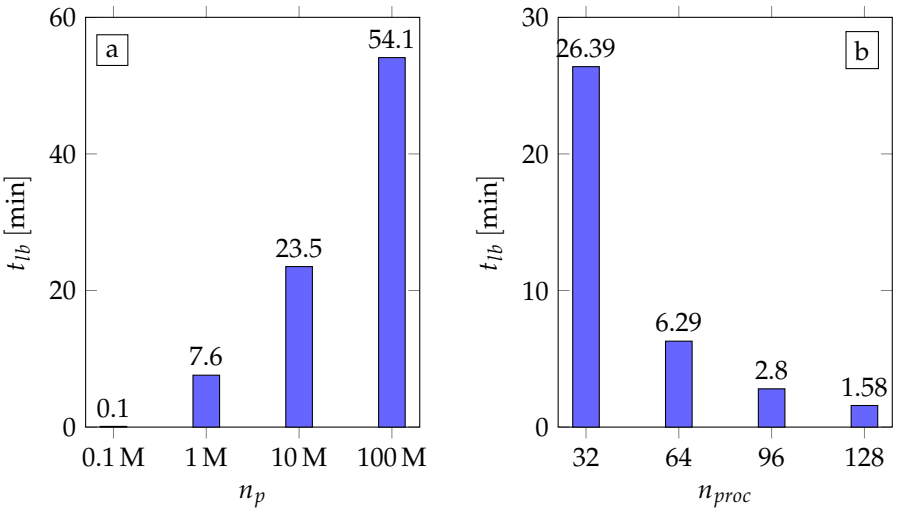


**Figure 4.** Required time to perform a single load-balancing for simulations of packing of (**a**) 0.1 M, 1 M, 10 M, and 100 M particles on 8, 48, 480, and 4800 processes, and (**b**) 1 M particles on 32, 64, 96, and 128 processes.

Figure 5 shows the simulation times of the packing of 1 M particles on 64 processes with `frequent` load-balancing ($f_{LB} = 10$, 20, 100, 200 and 1000 Hz). The total simulation time decreases from $f_{LB} = 10$ Hz to 20 Hz, and then increases as the load-balancing frequency increases. Not only the total cost of load-balancing increases with increasing the load-balancing frequency, it also shows that more frequent load-balancing cannot decrease the simulation time. This happens because some operations, such as the mapping of particles into subdomains and cells and the updating of the ghost particles has to be performed right after each load-balancing step. These operations add an extra computational cost to the total simulation time. In summary, we conclude that the time required to load-balance is non-negligible, especially for large-scale systems or a small number of processes and does not scale well with the number of particles and cells. As a result, we have to avoid any non-necessary load-balancing in the simulations. Indeed, for a larger number of particles, load balancing can become prohibitively expensive.

Figure 6 shows screenshots of packing simulations of 0.1 M, 1 M, 10 M, 100 M particles on 8, 48, 480, and 4800 processes at $t = 0.07$ s. Interested readers may find an animation of the simulation of 10 M particles in the supplementary materials Video S1 (`packing.mp4`). As the particles move towards the bottom wall of the rectangular box, load-balancing moves the subdomain of processes with the bulk of the particles to equalize the computational load on each process. Finally, one process handles the majority of the cells on top of the particle bed, while the rest of the processes are distributed evenly amongst the cells located in the bed of particles. Since 10,000 time-steps exist between two consecutive load-balance steps ($f_{LB} = 10$ Hz), load-balancing compensates its computational cost by increasing the speed of the DEM throughout the simulations. Consequently, a trade-off exists between the load-balancing time and the time it saves.

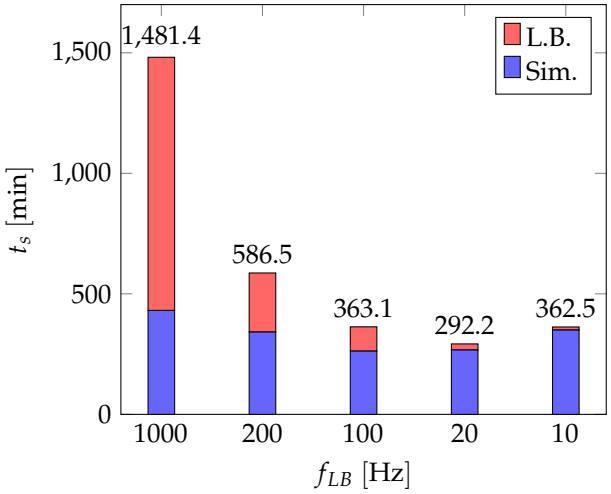


**Figure 5.** Simulation times of the packing of 1 M particles benchmark on 64 processes with `frequent` load-balancing at $f_{LB} = 10$, 20, 100, 200 and 1000 Hz. Load-balancing time is illustrated using the red color.

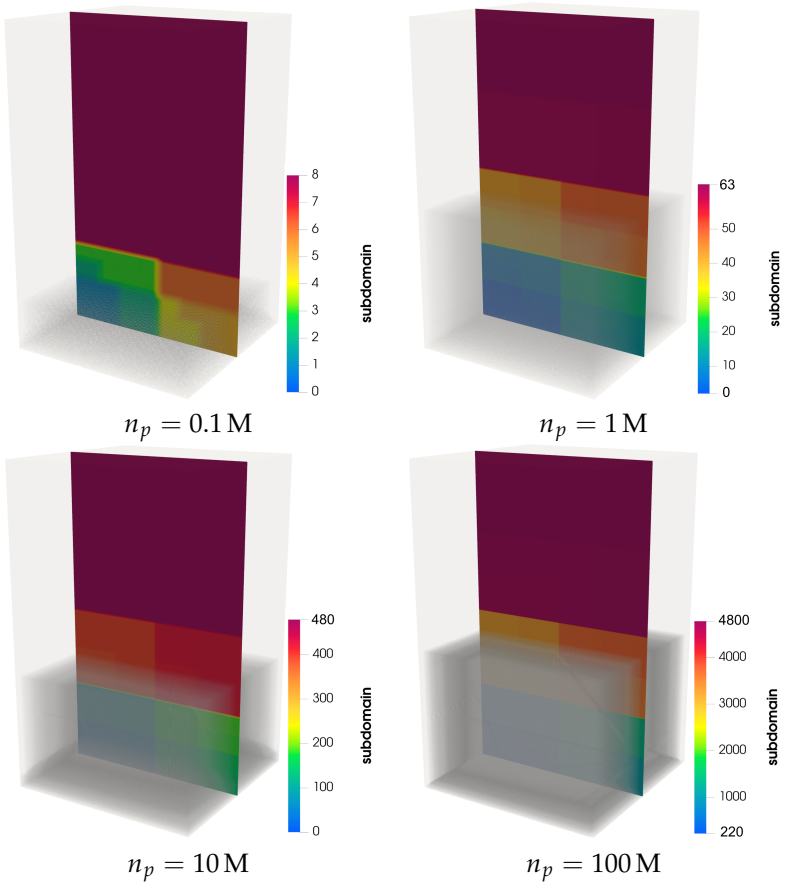


**Figure 6.** Simulation screenshots of packing of 0.1 M, 1 M, 10 M, 100 M particles on 8, 48, 480, and 4800 processes at $t = 0.07$ s.

### *3.4. Rotating Drum*

We performed the rotating drum simulation with three load-balancing strategies: `single-step` load-balancing at $t = 1.5$ s, `frequent` load-balancing with $f_{LB} = 10$ Hz, and `dynamic` load-balancing with $f_{LB} = 10$ Hz and $\epsilon = 0.8$. Since the system reaches the steady-state at approximately $t = 1$ s, we call load-balancing at $t = 1.5$ s when using the `single-step` scheme. The simulation with the `frequent` scheme calls the load-balancing

100 times during the simulation, while the simulation with the `dynamic` scheme calls load-balancing eight times. Figure 7 shows the distribution of subdomains before (at $t = 1$ s) and after (at $t = 8$ s) load-balancing. This distribution does not significantly change after $t = 1$ s, when the granular flow reaches pseudo steady-state.

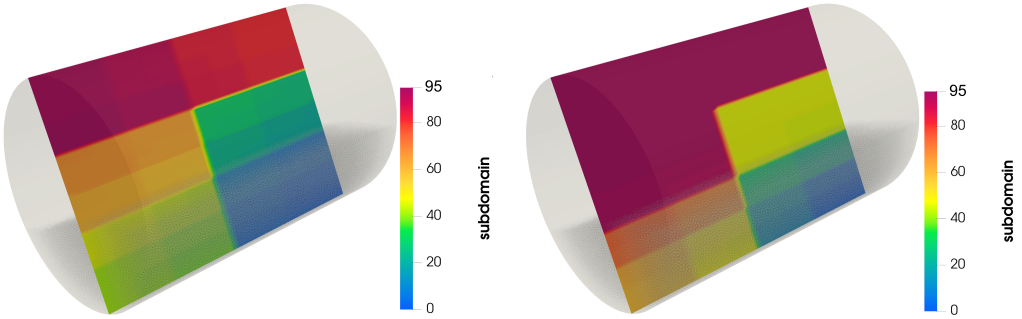


**Figure 7.** Distribution of particles and subdomains in the simulation of a rotating drum on 64 processes. (**left**): before load-balancing at $t = 1$ s, and (**right**): after load-balancing at $t = 8$ s.

Figure 8 shows the simulation times (including load-balancing time) with different load-balancing schemes. The simulation times without load-balancing, with `single-step`, `frequent`, and `dynamic` load-balancing schemes are 1337, 894.1, 762.4, and 868.3 min, respectively. These results show that `frequent` load balancing leads to a lower simulation time than the alternative strategy. However, these simulations include a small number of particles (0.226 M) for which the load balancing cost is small. According to the trends in Figures 4 and 5, for a similar simulation but with a higher particle count, minimizing the number of load-balancing operations using the `single-step` or `dynamic` strategy yields better results than `frequent` load-balancing. In general, at a small number of particles, cores and cells `dynamic` load-balancing is the most efficient scheme, while at a large number of particles ($n_p > 1$ M), cores and cells, `single-step` load-balancing is preferred for rotating drum simulations. Interested readers can find the animation of this simulation in the supplementary materials Video S2 (`drum.mp4`).

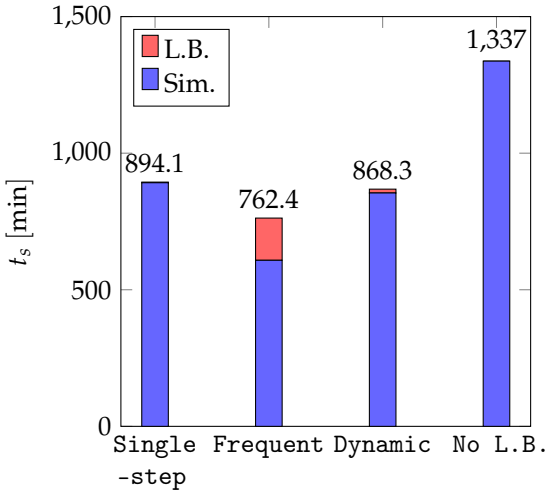


**Figure 8.** Simulation times of the rotating drum benchmark with different load-balancing schemes. Load-balancing time is illustrated using the red color.

### *3.5. Silo*

In the silo simulation, particles are packed on top of a stopper in the filling phase. Then in the discharge phase, the stopper is removed and particles leave the hopper and move into the bottom container. Figure 9 shows the distribution of subdomains and particles

in the silo simulation. Interested readers may find an animation of the silo simulation in the supplementary materials Video S3 (`silo.mp4`). At the beginning of the filling phase ($t = 4$ s), the particles and all the subdomains of processes except process 0 are in the hopper. As the discharge phase begins and particles leave the hopper, the simulation moves the subdomain of processes towards the bottom container. When all the particles completely leave the hopper ($t = 39$ s), all the particles and subdomains are located in the bottom of the geometry and one single process handles the hopper.

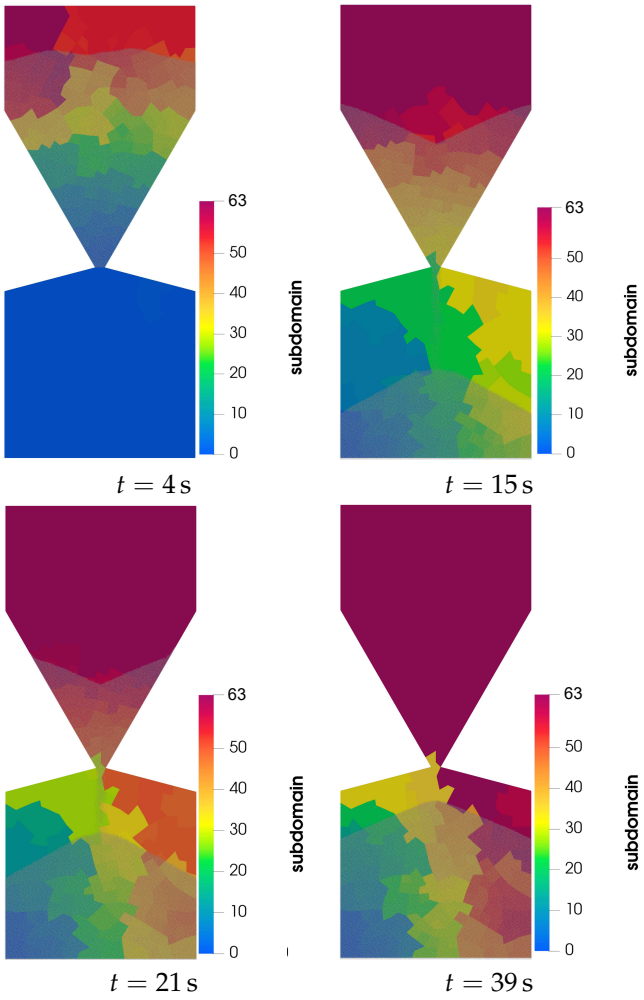


**Figure 9.** Distribution of subdomains and particles in the simulation of silo on 64 processes with $f_{LB} = 1$ Hz.

Using the `single-step` load-balancing for this simulation is meaningless. Figure 10 shows the simulation times (including load-balancing time) without and with `frequent` and `dynamic` load-balancing schemes. Figure 10 shows the simulation times of the silo simulation with no load-balancing, and with `frequent` and `dynamic` load-balancing schemes. `Frequent` (with $f_{LB} = 1$ Hz) and `dynamic` schemes (with $f_{LB} = 1$ Hz and a threshold of 0.8) call load-balancing 40 and 35 times, respectively, throughout this simulation. The simulation times are 154.1, 96.2 and 98.8 hours for no load-balancing, `frequent` and `dynamic` schemes, respectively. Load-balancing times are negligible compared to the total simulation time for this small-scale simulation ($n_p = 1.32 \times 10^5$). The `dynamic` scheme does not call load-balancing when the particles get packed on top of the stopper and in the bottom of the container (when the load-balancing is not necessary). Silo simulation results show that in systems where the particles move in one direction, `frequent` and `dynamic` load-balancing show the best computational performance.

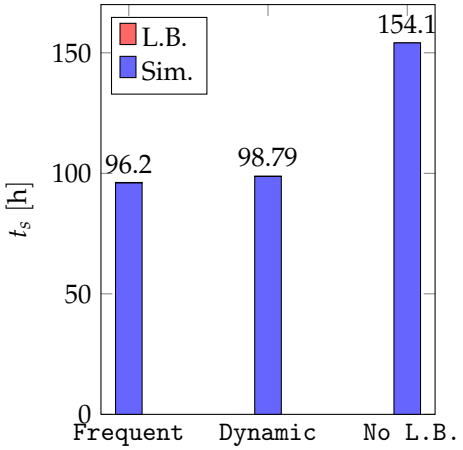


**Figure 10.** Simulation times of the silo benchmark without and with `dynamic` and `frequent` load-balancing schemes.

### *3.6. V Blender*

In a V blender, particles have a reciprocating motion. Similar to the silo, using a `single-step` load-balancing is meaningless here. We simulate the V blender with `frequent` (with $f_{LB} = 10\,\text{Hz}$) and `dynamic` (with $f_{LB} = 10\,\text{Hz}$ and $\epsilon = 0.8$) load-balancing schemes. `Frequent` and `dynamic` schemes call load-balancing 200 and 192 times, respectively. Figure 11 shows the distribution of the subdomains during the simulation on 64 processes with the `frequent` load-balancing scheme. Interested readers may find an animation of this simulation in the supplementary materials Video S4 (`VBlender.mp4`). Particles continuously move inside the V blender. As a result, `frequent` and `dynamic` load-balancing schemes are required to obtain the best performance in such systems. The simulations without load-balancing, and with `frequent` and `dynamic` schemes, take 3402.6, 2536.1, and 2325.1 min, respectively. In systems with reciprocating granular flow, `dynamic` and `frequent` schemes show the best performance.

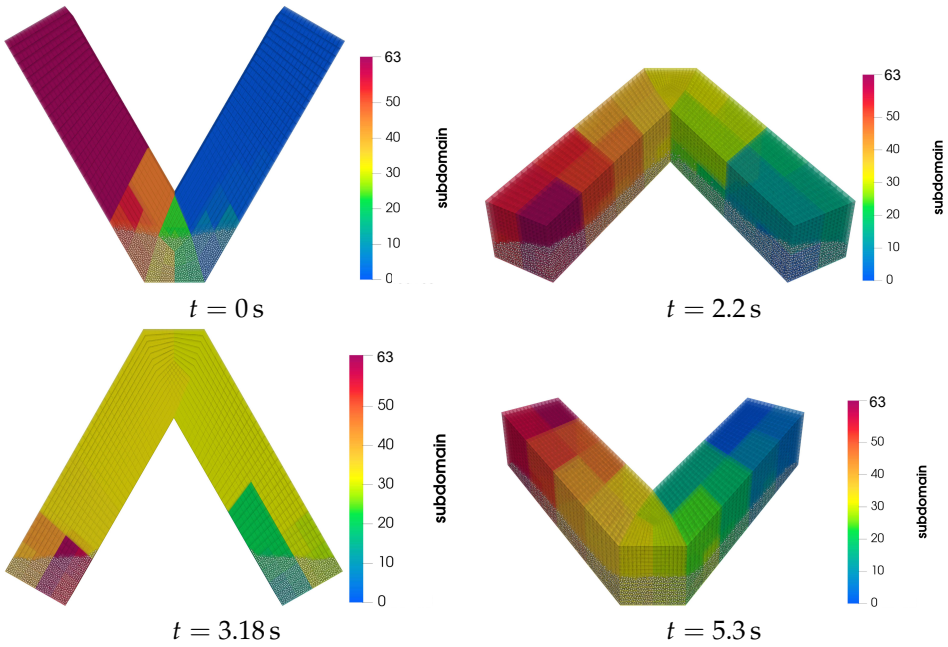


**Figure 11.** Distribution of subdomains and position of particles during the simulation of V blender on 64 processes with `frequent` scheme.

## 4. Conclusions

In this research, we compared the computational performances of linear and quadratic load-balancing weighting strategies as well as three different load-balancing schemes. Linear and quadratic strategies showed two minimums in the computational cost at particle weigh to cell weigh of 15, 50 and 3, 40, respectively. The first minimum is attributed to cell-based functions, while the second minimum is attributed to particle-based functions. We compared the load-balancing times in packing simulations. We observed that increasing the number of particles and cells and decreasing the number of processes in a simulation increases the load-balancing cost. As a result, we should avoid any non-necessary load-balancing in large-scale simulations. Afterwards, we evaluated the performances of different load-balancing schemes, namely `single-step`, `frequent` and `dynamic`, in systems with different behaviors. We observed that in large-scale systems in which particles reside in a constant region after reaching steady-state (for instance, the rotating drum), `single-step` load-balancing shows the best performance. In the simulations where particles move in one direction (for example, the silo) or where particles have a reciprocating motion (for example, the V blender) `frequent` and `dynamic` load-balancing schemes are favorable. `Dynamic` load-balancing is a scheme that calls load-balancing according to the imbalance between the computational load on the processes. Using `dynamic` load-balancing, we can avoid unnecessary load-balancing operations. In the packing simulation, we were able to simulate the packing of $10^8$ particles on 4800 processes, which shows the high computational performance of Lethe-DEM. On average, the computational cost of simulations with optimum load-balancing is 60% of simulations without load-balancing.

**Supplementary Materials:** The following are available online at https://www.mdpi.com/article/10.3390/pr10010079/s1, Video S1: packing.mp4, Video S2: drum.mp4, Video S3: silo.mp4, Video S4: VBlender.mp4.

**Author Contributions:** S.G.: Data curation, Formal analysis, Investigation, Software, Validation, Methodology, Visualization, Writing—original draft; B.B.: Funding acquisition, Software, Validation, Methodology, Project administration, Resources, Supervision, Writing—review & editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** This project was partially funded by the Natural Sciences and Engineering Research Council via NSERC Grant RGPIN-2020-04510.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Lethe-DEM is an open source DEM software available on GitHub https://github.com/lethe-cfd/lethe, (accessed on 21 December 2021).

**Acknowledgments:** The authors would like to acknowledge support received by the deal.II community. The authors would like to acknowledge the support received from Calcul Québec and Compute Canada. Computations shown in this work were made on the supercomputer Beluga, Cedar and Graham managed by Calcul Québec and Compute Canada. The operation of these supercomputers is funded by the Canada Foundation for Innovation (CFI), the ministère de l'Économie, de la science et de l'innovation du Québec (MESI) and the Fonds de recherche du Québec—Nature et technologies (FRQ-NT).

**Conflicts of Interest:** The authors declare no conflict of interest.

## Nomenclature

| | |
|---|---|
| $d_p$ | Particle diameter |
| $d_t$ | Time-step |
| $e$ | Coefficient of restitution |
| $f_{LB}$ | Load-balancing frequency |
| $L$ | Total computational load of a process |
| $n_c$ | Number of cells |
| $n_p$ | Number of particles |
| $n_{proc}$ | Number of processes |
| $t$ | Time |
| $t_f$ | Simulation time |
| $t_{LB}$ | Load-balancing time |
| $t_s$ | Simulation time |
| $W_c$ | Cell weight |
| $Y$ | Young's modulus |
| *Greek letters* | |
| $\alpha$ | Particle weight |
| $\beta$ | Cell weight |
| $\epsilon$ | `Dynamic` load-balancing threshold |
| $\mu$ | Coefficient of friction |
| $\mu_r$ | Coefficient of rolling friction |
| $\rho_p$ | Density of particle |
| $\nu$ | Poisson's ratio |

## References


1. Richard, P.; Nicodemi, M.; Delannay, R.; Ribiere, P.; Bideau, D. Slow relaxation and compaction of granular systems. *Nat. Mater.* **2005**, *4*, 121–128. [CrossRef] [PubMed]
2. Ketterhagen, W.R.; am Ende, M.T.; Hancock, B.C. Process modeling in the pharmaceutical industry using the discrete element method. *J. Pharm. Sci.* **2009**, *98*, 442–470. [CrossRef]
3. Boac, J.M.; Ambrose, R.K.; Casada, M.E.; Maghirang, R.G.; Maier, D.E. Applications of discrete element method in modeling of grain postharvest operations. *Food Eng. Rev.* **2014**, *6*, 128–149. [CrossRef]
4. Blais, B.; Vidal, D.; Bertrand, F.; Patience, G.S.; Chaouki, J. Experimental methods in chemical engineering: Discrete element method—DEM. *Can. J. Chem. Eng.* **2019**, *97*, 1964–1973. [CrossRef]
5. Golshan, S.; Sotudeh-Gharebagh, R.; Zarghami, R.; Mostoufi, N.; Blais, B.; Kuipers, J. Review and implementation of CFD-DEM applied to chemical process systems. *Chem. Eng. Sci.* **2020**, *221*, 115646. [CrossRef]
6. Sawley, M.L.; Cleary, P.W. A parallel discrete element method for industrial granular flow simulations. *EPFL Supercomput. Rev.* **1999**, *11*, 23–29.
7. Norouzi, H.R.; Zarghami, R.; Sotudeh-Gharebagh, R.; Mostoufi, N. *Coupled CFD-DEM Modeling: Formulation, Implementation and Application to Multiphase Flows*; John Wiley & Sons: Hoboken, NJ, USA, 2016.
8. Golshan, S.; Munch, P.; Gassmoller, R.; Kronbichler, M.; Blais, B. Lethe-DEM: An open-source parallel discrete element solver with load balancing. *arXiv* **2021**, arXiv:2106.09576.
9. Norouzi, H.; Zarghami, R.; Mostoufi, N. New hybrid CPU-GPU solver for CFD-DEM simulation of fluidized beds. *Powder Technol.* **2017**, *316*, 233–244. [CrossRef]
10. Eibl, S.; Rüde, U. A systematic comparison of runtime load balancing algorithms for massively parallel rigid particle dynamics. *Comput. Phys. Commun.* **2019**, *244*, 76–85. [CrossRef]
11. Rettinger, C.; Rüde, U. Dynamic load balancing techniques for particulate flow simulations. *Computation* **2019**, *7*, 9. [CrossRef]
12. Tsuzuki, S.; Aoki, T. Large-scale granular simulations using Dynamic load balance on a GPU supercomputer. In Proceedings of the Poster at the 26th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 16–21 November 2014.
13. Fleissner, F.; Eberhard, P. Load balanced parallel simulation of particle-fluid dem-sph systems with moving boundaries. *Parallel Comput. Archit. Algorithms Appl.* **2007**, *48*, 37–44.
14. Kloss, C.; Goniva, C.; Hager, A.; Amberger, S.; Pirker, S. Models, algorithms and validation for opensource DEM and CFD–DEM. *Prog. Comput. Fluid Dyn. Int. J.* **2012**, *12*, 140–152. [CrossRef]
15. Weinhart, T.; Orefice, L.; Post, M.; van Schrojenstein Lantman, M.P.; Denissen, I.F.; Tunuguntla, D.R.; Tsang, J.; Cheng, H.; Shaheen, M.Y.; Shi, H.; et al. Fast, flexible particle simulations—An introduction to MercuryDPM. *Comput. Phys. Commun.* **2020**, *249*, 107129. [CrossRef]
16. Forgber, T.; Toson, P.; Madlmeir, S.; Kureck, H.; Khinast, J.G.; Jajcevic, D. Extended validation and verification of XPS/AVL-Fire™, a computational CFD-DEM software platform. *Powder Technol.* **2020**, *361*, 880–893. [CrossRef]

17. Blais, B.; Barbeau, L.; Bibeau, V.; Gauvin, S.; El Geitani, T.; Golshan, S.; Kamble, R.; Mikahori, G.; Chaouki, J. Lethe: An open-source parallel high-order adaptative CFD solver for incompressible flows. *SoftwareX* **2020**, *12*, 100579. [CrossRef]
18. Fleissner, F.; Eberhard, P. Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. *Int. J. Numer. Methods Eng.* **2008**, *74*, 531–553. [CrossRef]
19. Warren, M.S.; Salmon, J.K. A parallel hashed oct-tree n-body algorithm. In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 19 November 1993; pp. 12–21.
20. Berger, R.; Kloss, C.; Kohlmeyer, A.; Pirker, S. Hybrid parallelization of the LIGGGHTS open-source DEM code. *Powder Technol.* **2015**, *278*, 234–247. [CrossRef]
21. Cintra, D.T.; Willmersdorf, R.B.; Lyra, P.R.M.; Lira, W.W.M. A hybrid parallel DEM approach with workload balancing based on HSFC. *Eng. Comput.* **2016**, *33*, 2264–2287. [CrossRef]
22. Cintra, D.T.; Willmersdorf, R.B.; Lyra, P.R.M.; Lira, W.W.M. A parallel DEM approach with memory access optimization using HSFC. *Eng. Comput.* **2016**, *33*, 2463–2488. [CrossRef]
23. Markauskas, D.; Kačeniauskas, A. The comparison of two domain repartitioning methods used for parallel discrete element computations of the hopper discharge. *Adv. Eng. Softw.* **2015**, *84*, 68–76. [CrossRef]
24. Arndt, D.; Bangerth, W.; Blais, B.; Clevenger, T.C.; Fehling, M.; Grayver, A.V.; Heister, T.; Heltai, L.; Kronbichler, M.; Maier, M.; et al. The deal. II library, version 9.2. *J. Numer. Math.* **2020**, *28*, 131–146. [CrossRef]
25. Arndt, D.; Bangerth, W.; Blais, B.; Fehling, M.; Gassmöller, R.; Heister, T.; Heltai, L.; Köcher, U.; Kronbichler, M.; Maier, M.; et al. The deal. II library, version 9.3. *J. Numer. Math.* **2021**, *29*, 171–186. [CrossRef]
26. Gassmöller, R.; Lokavarapu, H.; Heien, E.; Puckett, E.G.; Bangerth, W. Flexible and scalable particle-in-cell methods with adaptive mesh refinement for geodynamic computations. *Geochem. Geophys. Geosystems* **2018**, *19*, 3596–3604. [CrossRef]
27. Burstedde, C.; Wilcox, L.C.; Ghattas, O. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.* **2011**, *33*, 1103–1133. [CrossRef]
28. Bangerth, W.; Burstedde, C.; Heister, T.; Kronbichler, M. Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes. *Acm Trans. Math. Softw.* **2012**, *38*, 1–38. [CrossRef]
29. Geuzaine, C.; Remacle, J.F. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *Int. J. Numer. Methods Eng.* **2009**, *79*, 1309–1331. [CrossRef]
30. Owen, D.; Feng, Y.; Han, K.; Peric, D. Dynamic domain decomposition and load balancing in parallel simulation of finite/discrete elements. In Proceedings of the ECCOMAS 2000, Barcelona, Spain, 11–14 September 2000.
31. Golshan, S.; Esgandari, B.; Zarghami, R.; Blais, B.; Saleh, K. Experimental and DEM studies of velocity profiles and residence time distribution of non-spherical particles in silos. *Powder Technol.* **2020**, *373*, 510–521. [CrossRef]
32. Alizadeh, E.; Dubé, O.; Bertrand, F.; Chaouki, J. Characterization of mixing and size segregation in a rotating drum by a particle tracking method. *AIChE J.* **2013**, *59*, 1894–1905. [CrossRef]