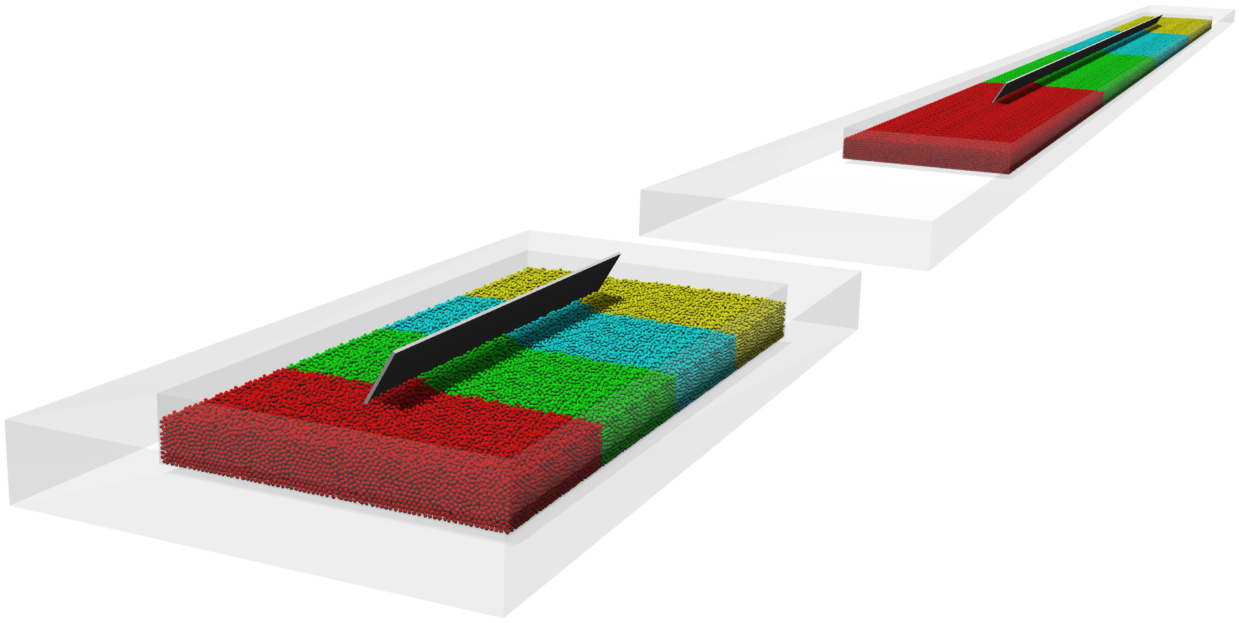
CHALMERS
UNIVERSITY OF TECHNOLOGY



Development of multi-GPU parallelization for a DEM solver

A parallelization extension for an existing state of the art DEM solver

Master's thesis in Engineering Mathematics and Computational Science

FREDRIK RASMUSSON

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Development of multi-GPU parallelization for a DEM solver

A parallelization extension for an existing state of the art DEM solver

FREDRIK RASMUSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

Development of multi-GPU parallelization for a DEM solver
A parallelization extension for an existing state of the art DEM solver
FREDRIK RASMUSSON

Supervisors: Klas Jareteg and Adam Bilock, Industrial Path Solutions AB
Examiner: Anders Logg, Department of Mathematical Sciences

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: DEM simulation performed with the presented parallelization method.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Development of multi-GPU parallelization for a DEM solver
A parallelization extension for an existing state of the art DEM solver
FREDRIK RASMUSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The thesis presents a multi-GPU parallelization extension for an existing single GPU Discrete Element Method solver. The implementation extends the solver's capability to simulate large particle populations, making it possible to decrease the difference between simulations and real-world particulate systems. The code is developed with HPC in mind, carefully minimizing the additional overhead as a consequence of the parallelization operations by minimizing total number of communication points between the GPUs.
The computational domain is divided amongst the GPUs by splitting physical space through one of the three Cartesian axes. Although topologically simplistic, it advantageously results in few communication points for each GPU as well as efficient transfers between GPUs as memory locality is trivially achieved. The HPC GPU clusters targeted by the solver generally have 4-8 GPUs which for most cases will be well suited for the one-dimensional domain decomposition.
A load balancing scheme have been developed which dynamically shifts the domain borders to distribute the computational load between the devices. The scheme is optimized for even simulation time between the GPUs. This is achieved by measuring and monitoring execution time of some key operations performed in the DEM algorithm and incrementally shift the domain borders to reach a state where all solvers have close to equal execution times for these operations.
Performance measurements have been performed through Amazon Web Services Accelerated Computing instances with systems ranging from 4 to 8 GPUs. The total cost of the parallelization in relation to total execution time ranges from 2.6% to 6.5% with increasing number of connected GPUs. Thus, the implementation of the parallelization scheme is deemed efficient and successful. The chosen and defined algorithm is verified and benchmarked on three cases. The verification shows that the physics of the single GPU solver is preserved for the multi-GPU solver. The dynamic load balancing is shown to give beneficial advantages over static decomposition and the optimization scheme for the balancing is verified on a simulation case with dynamic particle behavior. The overall scaling of the algorithm is studied by benchmarking and monitoring the cost associated with the different steps of the DEM algorithm. It is shown that for certain steps, part of the original single GPU solver, the scaling is worse than for the added implementation steps. This is analyzed and considered to be an effect of the memory schemes for the peer-to-peer mode on the GPUs and will require further attention in future work.

Keywords: Discrete Element Method, Parallelization, GPU, mulit-GPU, HPC, Domain decomposition, Dynamic domain decomposition.

Acknowledgements

First of all, I want to acknowledge my supervisors Klas Jareteg and Adam Bilock whom have supported me throughout the project. Klas, you always take the time for discussion with valuable and interesting input. After these discussions, I am left with inspiration and eagerness to continue working. Adam, your expertise in coding and DEM have been of great help. You have managed to give insightful answers to all my questions regardless of their difficulty. To both of you, I have time after time been astounded of your breadth of knowledge and capability to solve complex problems. I also want to show my gratitutde towards the DEM research group at FCC, with Johannes Quist as project lead. I admire your interest in DEM, modelling and research, and have enyojed our meetings. Furthermore, I want to acknowledge my examiner Anders Logg for valuable feedback and for devoting valuable time for this thesis.
I want to thank IPS for giving me the opportunity to complete my studies with an interesting and relevant topic. I have received a great welcome and look forward to start my professional career with you all.
Finally, with my 5 years of studies coming to an end, I want to thank family and friends for their never-ending support.

Fredrik Rasmusson, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AWS	Amazon Web Services
BVH	Bounding Volume Hierarchy
CUDA	Compute United Device Architecture
DEM	Discrete Element Method
FCC	Fraunhofer-Chalmers Centre
GPU	Graphics Processing Unit
HM+D	Hertz-Mindlin-Deresiewicz
HPC	High Performance Computing
MPI	Message Parsing Interface
PCIe	Peripheral Component Interconnect Express
SIMT	Single-Instruction, Multiple Threads
SM	Streaming Multiprocessor
SPH	Smoothed Particle Hydrodynamics
STL	Standard Template Library

Contents

List of Figures

List of Tables

1

Introduction

Granular flows are present in numerous different industries such as agriculture, mining and mineral processing. These industries generally have a big throughput in their production. In Sweden, the deliveries of aggregates from the mining industry year 2020 went up to just above 100 million tonnes [1]. For the same year that resulted in an energy consumption of just above 6000 GWh [2], which roughly corresponds to one third of the total energy consumption of the city of Gothenburg the same year [3].

The demand for aggregates are still increasing [1] but at the same time energy prices are surging and there is a general movement towards more sustainable industries. Because of this, improvement of efficiency has become increasingly interesting for many companies in related industries. However, shutting down the production to test different points of possible improvement is generally not feasible due to cost and long downtimes. Simulating the granular material in different parts of the production allows to thoroughly test and analyze improvements of the production. This allows for decisions to be well thought through and possible implementation to be carefully planned out.

A popular method of simulating granular materials is the Discrete Element Method (DEM), first implemented by Cundall and Strack [4] in 1979. DEM is a numerical method which considers each particle as a distinct geometric element with contact forces arising between the elements. The contact forces result in translational and rotational changes through Newton's second law of motion [4]. As each particle is distinct, a single simulation can contain particles of different sizes, stiffness and geometry etc. This gives great potential in accurately capturing the dynamics of granular flow.

DEM has been implemented for many applications where different shaped particles are of interest. Some examples include Cleary [5], who explored the influence of particle shape on granular systems such as vibrating screens to separate particle size fractions, a blade plow mixer, the process of filling a dragline bucket, flow through transfer chutes, a cement ball mill and a landslide. Liu et al. [6] investigated hopper flow with different particle shapes and verified with physical experiments. The result indicated that particle shape has a great impact on parameters such as the mixed region, stagnant zone and wall stress. Geng et al. [7] simulated slender particles in a rotating drum dryer and investigated the influence of rotational velocity and compared with spherical particles. Ketterhagen et al. [8] reviewed DEM models

for several common pharmaceutical processes such as material transport, storage, blending, granulation, milling, compression and film coating.

Resolving each particle has the potential of yielding very accurate simulations of complex granular flow, but this is also the burden of the method. Real world problems generally have extremely large particle populations which together with the computational intensity of the method results in long simulation times [9]. One approach to speed up calculations is to use simplified particle models such as spheres [10]. A simple particle model is less computationally intense but potentially does not capture the correct particle dynamics. On the contrary, more complex particle models such polyhedra [11] has a higher computational demand but has the potential of accurate dynamics. As an example of how long simulation times could be, Gan [12] states that a 1 million particle population could take weeks simulated on a single processor. Because of the computational demand and high particle populations, a useful DEM simulation framework must take High Performance Computing (HPC) into consideration.

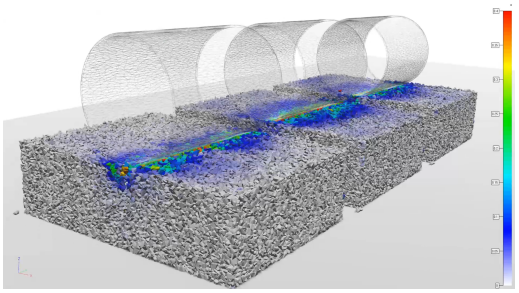
Graphics Processing Units (GPUs), with original purpose of rendering real-time graphics in computer games, has in the last few decades become available for more general purpose programming. This, along with a great increase in their computational capabilities and low cost has made them a core part in the HPC scientific community [13, 14]. NVIDIA is a leading actor in making this possible with their toolkit Compute United Device Architecture (CUDA) which makes the GPU accessible from a variety of high level programming languages such as C++ and Python. Proper GPU implementation in a DEM simulation framework can increase the simulation speed by several times [15, 16]. However, the GPUs have a limited amount of fast accessible memory which limits the size of the particle population over which the calculations are done. In order to extend this limit, the simulation should be parallelized over multiple GPUs.

Systems with multiple GPUs generally come in two shapes. Commercially available multi-GPU systems commonly have 4-8 GPUs with peer-to-peer connection. The peer-to-peer connection allows for fast data transfer in comparison to utilizing the data bus which connects the GPUs to the CPU. The second common way of connecting multiple GPUs is through a message-passing system such as the standardized Message Parsing Interface (MPI) [17]. MPI connects multiple CPU nodes which in turn can either execute the instructions themselves or pass it on to GPUs connected to the node.

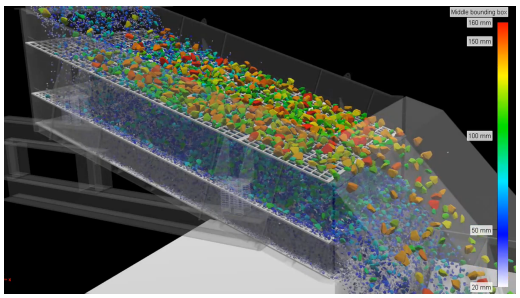
As examples from the literature, Yan and Regueiro [9] produced a DEM code for simulating complex-shaped granular particles and parallelized utilizing MPI. They ran simulations on up to 2048 compute nodes and showed that their communication time was a decreasing function of the number of compute nodes in strong scaling measurements. Rustico et al. [18] presented in 2012 an optimized multi-GPU version of a GPU Smoothed Particle Hydrodynamics (SPH) implementation. The platform on which they simulated had 6 GPUs connected to each other and host by PCIe 2.0. The implementation managed to closely follow the ideal speedup. Tsuzuki et al. [19] simulated on up to 512 GPUs connected via MPI and succeeded with a DEM simulation containing 129 million particles. The efficiency decreased with

increasing amount of GPUs with one cause being that the subdomains ran the risk of having extreme aspect ratios and thus a high communication load. Domìnguez et al. [20] wrote a multi-GPU implementation originating from the single-GPU DualSPHysics code. With 64 GPUs connected via an improved MPI they managed an SPH simulation with just over 1 billion particles. In 2020 Park et al. [21] created a parallelized multi-GPU SPH code to further explore nuclear safety. They implemented a simple one-dimensional domain decomposition and utilized peer-to-peer communication between the GPUs. Tian et al. [22] implemented a multi-GPU DEM code with asynchronous communication and a linked-cell list which was beneficial for communication. A speedup ratio of 10.39 was achieved with 16 GPUs versus 128 CPU cores. Most of the methods from these examples have been developed for large supercomputer clusters. These platforms are not the typical target for the software used and extended in this thesis, and there is no off the shelf method for DEM parallelization with a commercial aspect. The system for which Park et al. [21] developed their parallelization is most similar commercially available systems and is therefore of higher relevance.

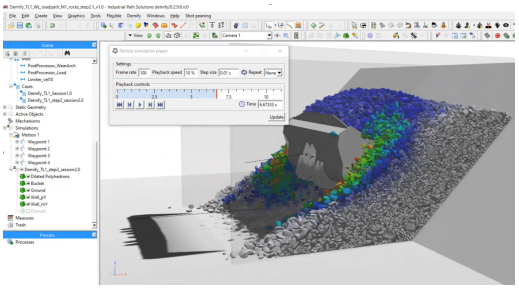
Demify® is a DEM software currently developed by Industrial Path Solutions Sweden AB in collaboration with Fraunhofer-Chalmers Centre (FCC). It is a state of the art single GPU implementation capable of simulating particles with complex polyhedron shapes [23]. Four examples of Demify® simulations can be seen in Figure 1.1.



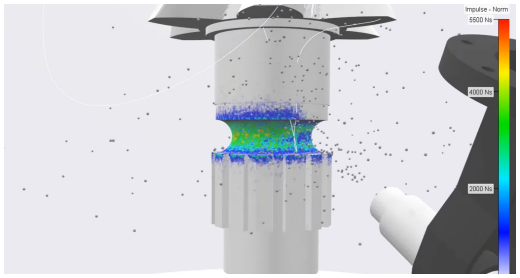
(a) Three particle beds compacted by rollers which apply a downward force.



(b) An industrial screen used for dividing the particle population into different fractions of sizes.



(c) A wheel loader bucket digging into a pile of gravel.



(d) Simulation of shot peening. A process used to harden metal components.

Figure 1.1: Four example cases of Demify® DEM simulations visualized in the IPS software.

Figure 1.1a is a compacting case where the rollers apply pressure to the particle bed below [24, 25]. Next, Figure 1.1b is an industrial screen which vibrates, separating

particles into three different size fractions. Figure 1.1c shows a simulation of a wheel loader bucket digging into a particle pile which used to evaluate optimal bucket geometry. Lastly, Figure 1.1d shows a simulation of shot peening which is a hardening process where particles are shot against a metal surface. These examples showcase the flexibility of the solver working with complex geometries and motions as well as large-scale problems with complex shaped particles.

To even further extend the single-GPU capabilities of Demify®, a multi-GPU extension will be implemented allowing for larger problems and short simulation times. It will be tested on multi-GPU systems with up to 8 GPUs.

1.1 Project scope and limitations

The project is conducted at Industrial Path Solutions Sweden AB with the purpose to *implement, verify and evaluate a multi-GPU extension to the company's existing DEM GPU framework with a close to ideal performance gain.* In order to reach the goal during the given time, some restrictions are put in place. This to ensure that the time spent are beneficial to the project. The three main restrictions are:

- Existing simulations cases will be used for the verification simulations.
- No new applications or particle models will be studied.
- No experimental validations will be performed, but rather the parallelized solver will be verified against already validated single GPU simulations.

As the parallelization should be general and not only work on special simulation cases, utilizing existing simulation cases is therefore sufficient. Demify® already has several particle models implemented and suitable for parallelization. Finally, experimental validations are unnecessary as the parallelized version can simply be compared to the single GPU version, which is experimentally validated.

1.2 Research questions

The main questions that the project aims to answer are:

- What are suitable decomposition schemes for distributing the particle population over multiple computational units?
- How should the decomposition be allowed to dynamically change as the particle population evolves over time?
- What is the overall cost, and what are the limiting operations?

As the time for the Thesis work is restricted, exhaustive tests on different possible algorithms and implementations for parallelization are not possible. Instead, the first two questions are in part answered by studying literature and analyzing methods suitable for implementing in Demify®. The last research question is answered by analyzing the performance of the implementation.

2

GPU architecture

In order to understand choices made for algorithms and the general implementation of the parallelization, it is necessary to have some background information on the hardware on which the implementation runs. GPUs are processing units specialized for parallel computing and thus have a different purpose than the CPU. On a transistor level, the difference is that a GPU has more transistors devoted to data processing, while the CPU has a more even balance with data processing, data caching and flow control [26]. In Figure 2.1 this difference is illustrated in a schematic. The schematic is not an illustration of any real hardware but the size and colors of the boxes depicts the main differences between CPUs and GPUs. The main difference being that the GPU has a larger amount of floating point units (green boxes) than the CPU. However, the floating point units of the CPU are individually more capable. CPUs can handle more complex workflows as a larger part of the chip is dedicated towards control units (blue boxes). A third main difference is that CPUs have different cache hierarchy. These differences combined make the GPU much more suited for highly parallelized workloads but limited in other aspects.

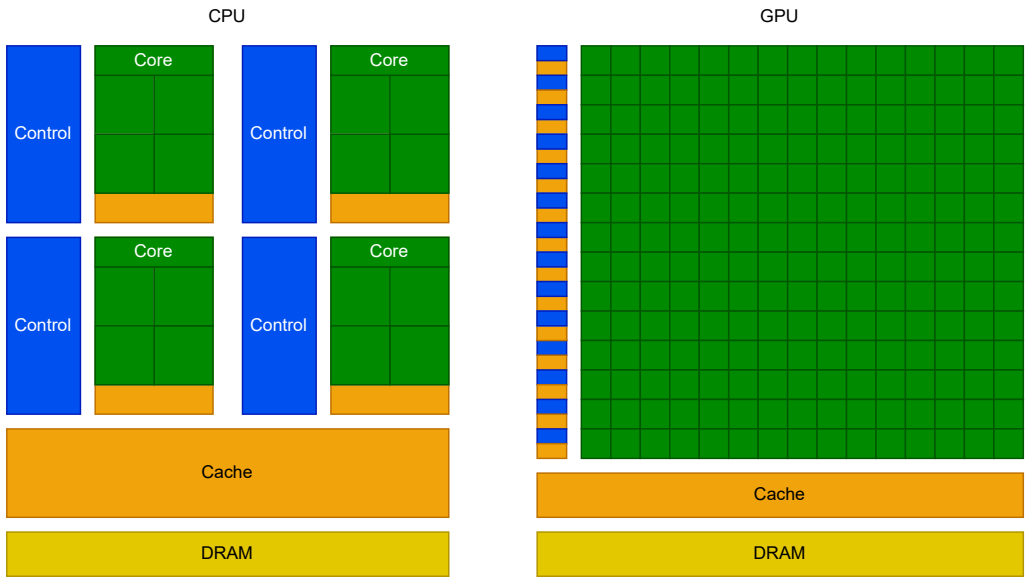


Figure 2.1: Illustration of the architectural differences between CPU and GPU. Green boxes represent floating point units, blue boxes control units, orange boxes layers of cache memory and yellow boxes the DRAM. This does not represent any real hardware and is only used for illustration purpose.

The hardware architecture to achieve this is built around a scalable array of multithreaded blocks called *Streaming Multiprocessors* (SMs). The architecture within the multiprocessors is called *Single-Instruction, Multiple-Thread* (SIMT) and manages the hundreds, or even thousands, of threads that the processor is designed to execute concurrently. The term SIMT originates from the fact that the smallest groups of threads that the multiprocessor creates, manages, schedules and executes called *warps* are 32 parallel threads executing one common instruction. If threads of a warp diverge on their execution path, all other threads in that warp are idle. Therefore, maximum efficiency is achieved when all 32 threads of a warp agree on the execution path [26].

The data bus which allows the CPU to instruct the GPU of what to execute is called the Peripheral Component Interconnect Express (PCIe) bus. However, compared to internal memory bandwidth on both GPU and CPU this bus is extremely slow and communication should therefore be kept at a minimum. An example of connections and bandwidths for a multi-GPU system can be seen in Figure 2.2. The values in the presented example are extracted from a system with two NVIDIA RTX 3090 GPUs connected via NVLink and an AMD Ryzen 9 7950X 16-Core processor CPU. The illustration is extended to a four GPU system to show how an even bigger system could be connected. The bus which connects the CPU to the GPU is in this case a PCIe 4.0 bus where a bandwidth of 9 GB/s was achieved. The shared memory bandwidth between the GPUs through NVLink was measured to 100 GB/s while the internal bandwidth of each card was measured to roughly 830 GB/s. This emphasizes the heterogeneity and complexity that have to be taken into consideration when utilizing single or multi-GPU systems.

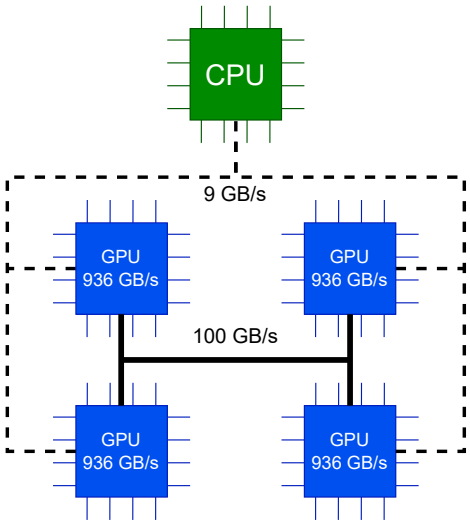


Figure 2.2: Schematic of how a GPU cluster could be connected. The green host (the CPU) has a shared connection to all blue GPUs with relatively low bandwidth. The GPUs are connected to each other separately with a significantly faster bus. The bandwidth of the internal memory of the GPUs are also included to reveal the relative differences. The values are extracted from a system with two NVIDIA RTX 3090 GPUs connected via NVLink and an AMD Ryzen 9 7950X 16-Core processor CPU.

The parts of Demify® that is run on the GPU use the general purpose parallel computing platform and programming model CUDA [26] which acts as an extension to the C++ programming language that the code is implemented with. For this thesis, the basics of the CUDA model is sufficient. The GPU and its SMs have no effective global synchronization and as mentioned earlier, the GPU is not designed to work efficiently for single threaded control flows. Because of this, the host runs the main program, dealing with flow and synchronization. The main program can then send data to the GPU and launch kernels which executes operations on the data. For example, Demify® stores all the particle data on the GPU and thus perform all particle operations on the GPU as well. The host deals with simulation initialization and controls the flow of which operations to be executed at what time. A visualization of the software flow can be seen in Figure 2.3.

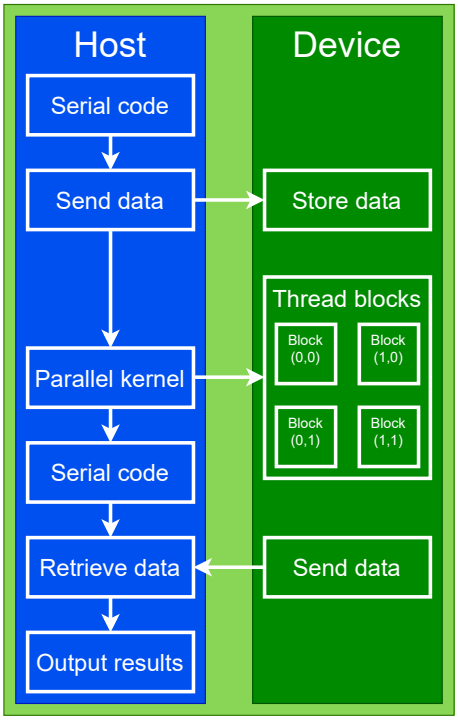


Figure 2.3: Visualization of what a software flow between host and device could look like. The host controls the serial part of the software and commands the device to execute parallelized kernels.

3

DEM Simulation Framework

3.1 DEM model

The particles are contained in a system of particles and a surrounding consisting of geometric objects with which the particles interact. Each particle is modeled as a rigid body with a state tracked over discrete time steps. For each time step the dynamics of each particle is determined by Newton's second law,

$$\begin{aligned} m_i \frac{\mathrm{d}^2 \boldsymbol{x}_i}{\mathrm{d}t^2} &= \boldsymbol{F}_i \\ I_i \frac{\mathrm{d}^2 \boldsymbol{\Theta}_i}{\mathrm{d}t^2} &= \boldsymbol{T}_i. \end{aligned} \tag{3.1}$$

Each particle i have a mass m_i, position at center of mass $\boldsymbol{x}_i$ acting force $\boldsymbol{F}_i$, inertia I_i, orientation $\boldsymbol{\Theta}_i$ and acting torque $\boldsymbol{T}_i$. Both implicit and explicit integration methods can be utilized to evaluate the time evolution of the particle system. Most commonly, explicit methods are preferred as the resulting equations are more suitable for numerical computations on massively scaling hardware, such as GPUs. In addition, the explicit methods have advantages in the formulation of the forces between the objects, allowing a direct use if e.g. Hertzian mechanics. Explicit methods are the chosen integration type for the framework in which this project is conducted and if not else stated, is the integration type throughout the report. The main steps of a DEM simulation can be seen in Figure 3.1.

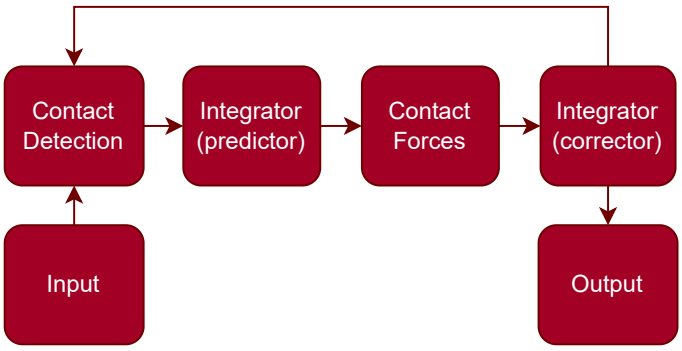


Figure 3.1: The core algorithmic flow performed by Demify®.

Input refers to the problem setup in terms of particle initial states and physical parameters e.g. friction coefficient. *Contact detection* covers the search steps to find pairs between particles and between particles and rigid objects that could exert forces on each other. A predictor-corrector integration scheme are used in this illustration and thus the predictor step is executed before the contact forces are calculated. The resulting loads from the contact model are then calculated in the step *Contact Forces* which gives input to the corrector step of the integration. At this point, the states are updated and then loops back to the *Contact Detection*. The loop continues until some termination condition is met. Such a condition would break the loop and lead to the *Output* step. Contact detection, integration and contact forces will be covered in the next subsections followed by an overview of different element representations.

3.1.1 Contact detection

In order to find contact pairs, a spatial search has to be executed. Then, with pairs identified the overlap of the pairs is resolved for the contact force equations. The search algorithm could for small populations simply be done by for each particle traverse all other particles and find those which meet the condition of contact. Such an algorithm would be $\mathcal{O}(n^2)$ in complexity where n denotes the number of particles. For populations of several thousands of particles, this would be practically infeasible, not to mention populations of millions or billions of particles. One alternative type of algorithm is a binning algorithm. Munjiza and Andrews [27] developed a method called *no binary search* with complexity $\mathcal{O}(nlog(n))$, however with limitation of only being applicable to systems of bodies of similar size. Williams et al. [28] presented an extended traditional binning algorithm to handle arbitrary sizes and shapes in two and three dimensions with complexity $\mathcal{O}(n)$. For the DEM-framework in which this work is implemented, the search algorithm is a tree-based algorithm based on [29, 30, 31, 32], called Bounding Volume Hierarchy (BVH). For this application, the tree will be used to find if volumes are overlapping. If a node have an overlap, so will all its parents. So, to find contact pairs is simply to go down the tree, discarding branches with no overlap and continuing down the ones with. According to Ericson [33], this method scales logarithmically for most use cases.

With pairs identified, each contact needs to be resolved to get all information needed for the contact force model. The contact model will be presented in the next subsection. Meanwhile, we will just care about the properties that the contact detection will need to output for the contact model. For spheres, the only pair-specific information needed is the center coordinates of the two colliding particles, their velocities and the normal direction of the interaction. The resolution of spheres is trivial as the spheres positions and velocities are stored for each time step and the normal direction is calculated using the coordinates. The resolution for polyhedra will not be covered as its force model as well is not covered and deemed out of scope. The approach is however similar to that of spheres and is covered in [23].

3.1.2 Contact Forces

Contact forces are necessary to evaluate the right-hand side of the two equations in 3.1. In this subsection, the Hertz-Mindlin-Deresiewicz (HM+D) model for spheres will be described.

The elastic force of the HM+D model has a normal elastic force corresponding to the Hertz contact law for spheres [34]

$$F_{n,e} = \frac{4}{3} E^* (R^*)^{1/2} \delta^{3/2}. \tag{3.2}$$

Here E^* refers to the effective modulus, R^* the effective radius and δ the indentation depth. These three parameters are defined as follows,

$$\frac{1}{E^*} = \frac{1-\nu_1^2}{E_1} + \frac{1-\nu_2^2}{E_2}. \tag{3.3}$$

where ν_i is the Poisson's ratio and E_i is the Young's modulus. Here, $i = 1, 2$ is the sphere's index. Similarly, the effective radius is defined as,

$$\frac{1}{R^*} = \frac{1}{R_1} + \frac{1}{R_2} \tag{3.4}$$

with R_1 and R_2 representing the spheres' respective radius. By letting $R_1 \to \infty$ the intersection between sphere 2 and a plane is achieved and, the effective radius becomes $R = R_2$. Given R_1 and R_2, the indentation depth can be calculated as,

$$\delta = \begin{cases} R_1 + R_2 - |\boldsymbol{c}_2 - \boldsymbol{c}_1| & \text{if } |\boldsymbol{c}_2 - \boldsymbol{c}_1| < R_1 + R_2 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where $\boldsymbol{c}_i$ is the center of each sphere $i = 1, 2$.

Additionally, the model contains a scalar dissipative normal force which is defined as,

$$F_{n,d} = 2\gamma \sqrt{m^* k_n} \nu_n \tag{3.6}$$

where γ is a damping coefficient, m^* is the effective mass and is defined as

$$m^* = \frac{m_1 m_2}{m_1 + m_2}, \tag{3.7}$$

where m_1 and m_2 are the mass of sphere 1 and 2 respectively. Furthermore, k_n is the normal spring stiffness and ν_n is the relative normal velocity. The normal spring stiffness is defined as,

$$k_n = 2E^* \sqrt{R^* \delta} \tag{3.8}$$

The HM+D model also contains a tangential spring force which is derived from the no-slip theory of Mindlin [35]. This is the tangential force used in the core solver

of Demify®. However, for this to work with the parallelization, the particle history needs to be transferred along with the forces, but this is not covered by the thesis. Therefore, for this project a different tangential force model is used, which does not rely on any history.

The tangential component instead comes from Deen et al.[36] and uses a Coulomb-type friction law

$$F_t = \begin{cases} -k_t\delta_t - \eta_t v_{r,t} & \text{if } |F_t| \leq \mu_f |F_n| \\ -\mu_f |F_n| \boldsymbol{t}_r & \text{if } |F_t| > \mu_f |F_n| \end{cases} \tag{3.9}$$

where k_t is the tangential spring stiffness, δ_t the tangential displacement, μ_f the friction coefficient, η_t the tangential damping coefficient, $v_{r,t}$ the tangential relative velocity and $\boldsymbol{t}_r$ the tangential unit vector. The tangential displacement is defined as

$$\delta_t = v_{r,t}\Delta t \tag{3.10}$$

and tangential velocity as

$$v_{r,t} = \boldsymbol{v}_r - v_{r,n} \tag{3.11}$$

$$v_{r,n} = (\boldsymbol{v}_r \cdot \boldsymbol{n}_r)\boldsymbol{n}_r \tag{3.12}$$

$$\boldsymbol{v}_r = \boldsymbol{v}_1 - \boldsymbol{v}_2 - (R_1\boldsymbol{\omega}_1 + R_2\boldsymbol{\omega}_2) \times \boldsymbol{n}_r \tag{3.13}$$

$$\boldsymbol{n}_r = \frac{\boldsymbol{c}_2 - \boldsymbol{c}_1}{|\boldsymbol{c}_2 - \boldsymbol{c}_1|} \tag{3.14}$$

where the relative velocity $v_{r,n}$, relative normal direction $\boldsymbol{n}_r$, relative velocity $\boldsymbol{v}_r$ also are presented. Here $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the velocities of the two particles in contact and $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ are the corresponding angular velocities. The tangential damping coefficient is expressed as

$$\eta_t = -2\ln(e_t)\frac{\sqrt{\frac{2}{7}m_r k_t}}{\sqrt{\pi^2 + (\ln(e_t))^2}} \tag{3.15}$$

where $0 < e_t < 1$ is the tangential coefficient of restitution and m_r defined as

$$m_r = \left(\frac{1}{m_1} + \frac{1}{m_2}\right)^{-1} \tag{3.16}$$

Finally, the tangential unit vector $\boldsymbol{t}_r$ can be calculated through the tangential velocity as follows

$$\boldsymbol{t}_r = \frac{v_{r,t}}{|v_{r,t}|}. \tag{3.17}$$

3.1.3 Integration

The equations of motion given in 3.1 are of the form of coupled ordinary differential equation. As mentioned, granular systems often contains many particles and therefore, computational efficiency while maintaining accuracy is of high importance. The integration of the equations only accounts for a small part of the over all computational time, but a wisely chosen integration scheme allows for larger time steps and thus lower simulation time. The integration scheme used for this work is Velocity Verlet setup as a predictor-corrector.

1. Update velocity and position (predictor)

$$\begin{cases} \boldsymbol{v}_i\left(t+\frac{1}{2}\Delta t\right) = \boldsymbol{v}_i(t) + \frac{1}{2}\boldsymbol{a}_i(t)\Delta t \\ \boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i\left(t+\frac{1}{2}\Delta t\right)\Delta t \end{cases} \tag{3.18}$$

2. Compute force $\boldsymbol{f}_i$ based on predicted velocity and position and update acceleration

$$\boldsymbol{a}_i(t+\Delta t) = \frac{\boldsymbol{f}_i}{m_i} \tag{3.19}$$

3. Update velocity (corrector)

$$\boldsymbol{v}_i(t+\Delta t) = \boldsymbol{v}\left(t+\frac{1}{2}\Delta t\right) + \frac{1}{2}\boldsymbol{a}_i(t+\Delta t)\Delta t \tag{3.20}$$

Subscript i refers to each particle in the population over which the integration scheme is applied, $i = 1, \ldots, n$, where n is the total number of particles. This scheme which performs computations based on positions and velocities half a time step apart have been empirically tested and are acceptable [37]. The orientation of the particles are integrated using explicit forward Euler integration.

3.1.4 Element representations

As mentioned in the introduction (section 1) the particles can be represented with different fidelity. As a general rule, one can assume that a simple representation can speed up calculations with the downside of being limiting for what particle shapes it can approximate. On the contrary, an advanced representation results in less modelling and calibration and a more realistic particle flow while being computationally more demanding. Govender et al. [38] showed that particle shape can have a big impact on simulation results. It is therefore important to choose particle representations in relation to the goal of the simulation. In the following paragraphs, four different representations will be presented with their advantages and disadvantages. The four representations are spheres, multispheres, dilated polyhedra and polyhedra. In Figure 3.2, four scanned particles are presented and in Figure 3.3 four different representations are visualized for these scanned particles.

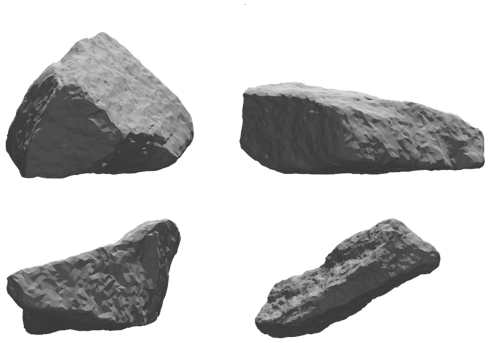
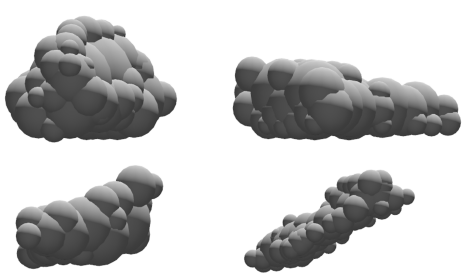
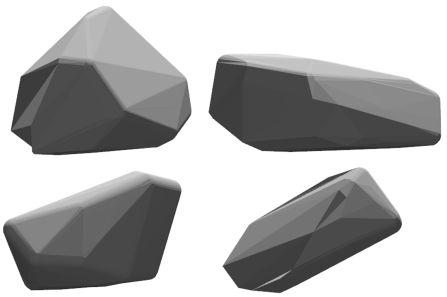


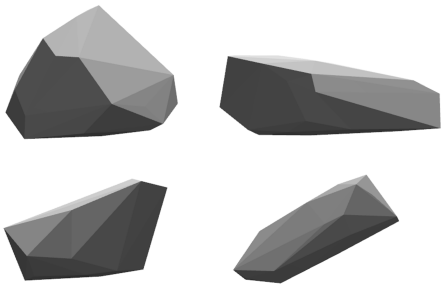
Figure 3.2: A scan of four gravel particles represented in software. See Figure 3.3 for examples of how these particles can be represented in a DEM simulation.



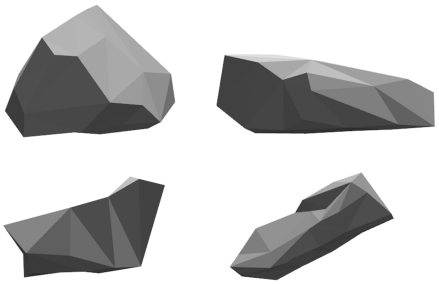
(a) Multispheres.



(b) Dilated polyhedrons.



(c) Convex polyhedrons.



(d) Non-convex polyhedrons.

Figure 3.3: Four methods of representing particle geometries in DEM. Each of the four methods are represent the four scanned particles presented in Figure 3.2.

Starting with spheres as it is the simplest way of representing a particle in three-dimensional space. Because of its simple and well studied topology there are some assumptions that can be made which can simplify some of the DEM algorithms. One such assumption is for example that there can only be one contact point between a particle-particle pair. Another advantage is that each particle takes up a minimal amount of memory which is advantageous when simulating large particle populations. The biggest disadvantage of the spheres is that granular flow often contain

irregularly shaped particles, which geometrically spheres can not represent. In an attempt to reduce this problem, Mehrdad et al. [39] used rolling friction that varied over the surface of the spheres. The results were good, but the parameters must be found empirically, and it is therefore more of an empirical rather than predictive approach.

Continuing with multispheres which are a compound of overlapping spheres, see Figure 3.3a. Some of the computational assumptions that can be made for the spheres still hold for this case while lending some possibility to represent irregular shapes. Mehrdad et al. utilized this method as well as the varying rolling friction method mentioned in the previous paragraph with good results. Since the irregularity of the granular particles to be simulated can be modeled prior to the simulation, this is a more useful approach. The main downside with this method is that each sub-sphere is independent in the contact detection and thus results in independent contact forces for each sub-sphere–sub-sphere pair. The accumulated contact force from all sub-spheres will therefore be higher than if the actual overlap volume could be resolved and used in the contact model [40]. Since a higher resolution representation implies an increasing amount of sub-spheres used for the particle representation, the method diverges.

A polyhedral particle representation could be used to capture more complex particle geometries, but as previously mentioned this is computationally demanding. The Minkowski sum theory combines the geometric simplicity of a sphere and complexity of a polyhedron. The resulting particle is often referred to as a dilated polyhedron and can be seen in Figure 3.3b. Efficient search algorithms for this representation has been derived [41, 42] and the geometric flexibility the method have over spheres are crucial for some problems [43].

Resolving the contacts of polyhedrons, see Figure 3.3c and Figure 3.3d, are computationally demanding and is therefore often avoided in DEM. Govender et al. [44, 45] has done work to reduce the computational cost by letting the contact forces between the combinations of faces and edges be independent. The decrease in computational cost is significant, however, similar to the multisphere approach where the contact forces also are independent, the accumulated contact forces might diverge. For polyhedrons, the exact volumetric overlap can be resolved and thus avoiding this issue. There have been plenty of work done based on this method. Early work was conducted by Nassauer et. al [46, 47], however with a particle population of not even 1000. Later, Govender et al. [48, 49, 50] managed simulations of convex and non-convex particles with populations of millions of particles. This is a significant improvement, and even better performance for polyhedrons were achieved by Bilock [23]. Although, there is still a big difference in computational cost between the simpler particle representations and the polyhedron representation is often avoided if possible. All four representations presented in this chapter are implemented in Demify®.

3.2 Demify architecture

The architecture of the code in Demify® is illustrated in Figure 3.4. Starting from the bottom is the *Core solver* which includes the backbone of the DEM-framework allowing for the flow shown in Figure 3.1 thus containing the governing equations and algorithms for neighbor search and integration. The squares inside the core solver are points of extra functionality to increase the usefulness of the program. For example, the generator can be placed in space and generates particles at a rate chosen by the user which allows for continuous flows without having to do so manually. There also exists extensions to the solver which are designed to not alter the core solver behavior. Basically, there are a set number of points in the core solver where extensions are executed, and the goal is to keep the solver indifferent to which extensions, if any, are run. The parallelization implementation in this project will be implemented as such an extension. Moving on, the core solver has an interface layer which connects the core solver to a structure of more user-friendly nature. This interface layer can be accessed through the IPS suite which is a GUI and through Python as a scripting-interface.

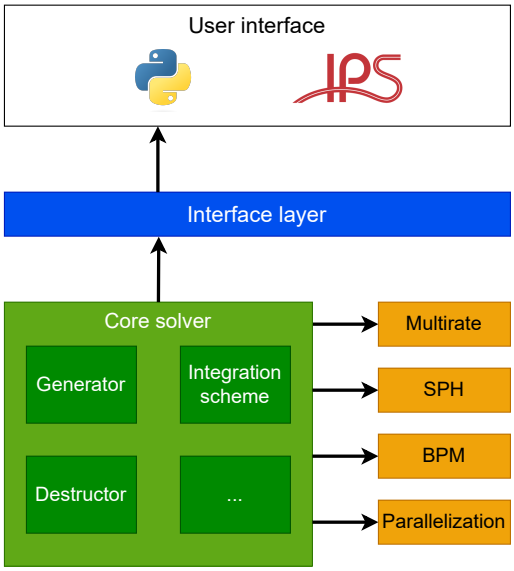


Figure 3.4: Illustration of the Demify® architecture and how the parallelization implementation fits into the structure. There is one-way communication between core solver and the orange extensions implying that the core solver is indifferent to the extensions.

The core solver is written using the programming language C++. A large part of the parallelized algorithms is implemented with the library Thrust [51]. The Thrust library contains parallel algorithms and resembles the C++ Standard Template Library (STL). It is a high-level interface which allows for both CPU and GPU implementations. The core solver in Demify® utilizes this interoperability and can be set to run on both the CPU and GPU.

To give further understanding for the core solver and the problems that will need to be addressed in order to implement the parallelization, a in depth flow-chart of the core solver can be seen in Figure 3.5. This main loop is entered when the solver has been set up and is ready for simulation.

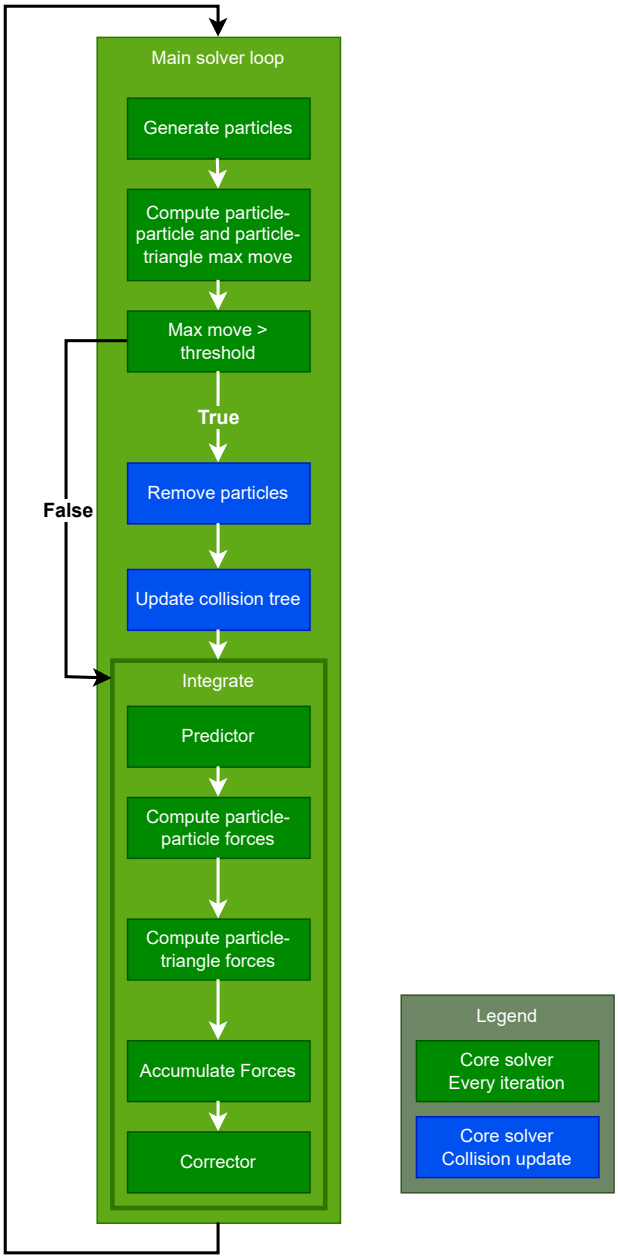


Figure 3.5: A schematic representing the main solver loop and its core operations. Note that there is a clause which is only run occasionally.

The generators, if there are any, are run early in the loop since new particles potentially will interact with the already existing ones. Next, the maximum relative movement between particles-particles and particles-triangles is calculated. The maximum movement is compared against a threshold labelled *move criterion*. At the

first time step, the clause will always be true. This clause is further covered in the next paragraph. The next step involves potentially removing particles that have exited the simulation domain, or entered a destructor volume. Next, in *Update collision tree* the collision tree is built and then the search for particle-particle and particle-triangle pairs is done. Finally, the integration is performed as described in 3.1.3. It is in *Compute particle-particle forces* and *Compute particle-triangle forces* that the calculations described in 3.1.2 are performed.

Now let us reconsider the step *Max move > move criterion*. If the maximum movement value is lower than the move criterion, it is guaranteed that the existing collision tree is still valid and there are no new potential particle-particle and particle-triangle pairs than the ones already captured. Therefore, the steps of the updating the collision tree is redundant. Contrary, if the maximum movement value is larger than the move criterion, there is no guarantee that the existing collision tree is representative of the particle population, and it needs to be updated.

4

Methods

The developed parallelization implementation will now be presented in the following order. First, the chosen domain decomposition is presented in two parts. Part one covering the static aspects which lay the foundation for the rest of the implementation. Part two covers how the decomposition is extended to be dynamic. At this point, the parallelization algorithm is covered and each part of it is explained and mapped into the core solver algorithm presented in the previous chapter.

4.1 Domain decomposition

The DEM equations and contact model given in the previous chapter shows that a particle only needs to consider other particles or triangles in its proximity. Because of this dependency on proximity, one method of dividing the computational work is to divide the problem in space. This gives an arbitrary amount of new smaller problems that can be distributed over equally many GPUs. However, the regions close to the domain boundaries must be considered carefully. Particles close to a domain border could have interactions both with particles internally, but also cross the border. Therefore, a buffer region where particles are shared is necessary.

First, one must decide how to divide the domain into smaller subdomains. Since it is in the buffer region that communication between devices will be necessary, it is of interest to keep the buffer region volume as small as possible. However, finding the smallest buffer region volume for most applications is not trivial. Assuming that such a boundary is found, transferring particles inside this volume gets more complex as the border is allowed to span over more dimensions in space. With this in mind, it was decided to implement a one-dimensional decomposition based on the work of Park et al. [21]. That is, divide the domain into smaller subdomains along a chosen axis, resulting in buffer regions being rectangular cuboids. This is the simplest way of dividing the domain and somewhat limiting in possibilities to minimize the volume of the buffer region. However, there are benefits in terms of low complexity of finding shared particles and easily sorting them in memory such that operations only affecting these particles can be done efficiently.

A visualization of the domain decomposition can be seen in Figure 4.1. As mentioned, the buffer region volume should be minimized to minimize the amount of particles that are shared between devices. Using this domain decomposition method there are two ways to minimize the buffer region volume. The most obvious one is to

minimize the width of the buffer regions. There are two parameters which decide its minimum size, the longest side of the biggest particle that can enter the buffer region (radius in the case of spheres) and the maximum movement criterion introduced in the previous chapter. To make sure that all particle-particle and particle-triangle pairs are identified at all times, the buffer region must expand minimum ΔL, see equation 4.1, in each direction from the subdomain border.

$$\Delta L = L_c + m \tag{4.1}$$

Here, L_c denotes the longest side of the biggest particle which could cause a collision and m is the maximum movement criterion. However, it is not always possible to achieve this value exactly because the sorting of the particles is not necessarily continuous, but rather they are sorted in bins along the same axis which the domain decomposition is done. The sorting step is more formally introduced in the next section. What should be mentioned here is that the smallest buffer region volume is achieved as the lowest multiple of the size of the sorting bins that is larger than ΔL.

The second way to affect the buffer region volume is by changing in which direction the domain is split along. The splits are allowed to be done along any of the three Cartesian axes and should thus be chosen such that the amount of particles inside the buffer region volume is minimal.

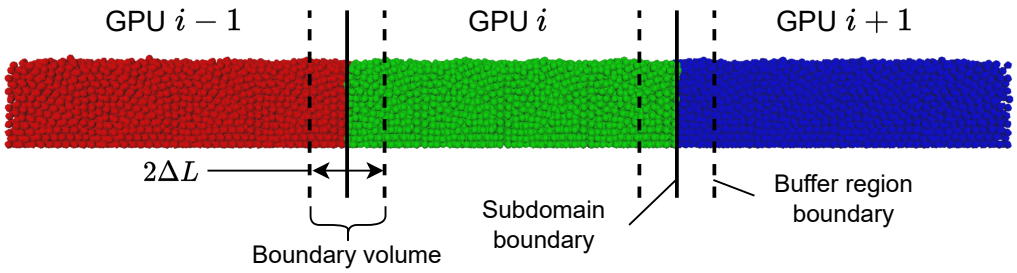


Figure 4.1: An example of a domain decomposition of a static particle bed. The size of the buffer regions are exaggerated for illustration purposes.

In Figure 4.2 the relation between the spatial decomposition and the particle memory is visualized. The simple nature of this illustration emphasize the point that the basic decomposition allows for good locality in memory which lessens the impact of the parallelization specific operations. The orange and red parts of the memory contain particles which are shared with the neighboring device and as can be seen there is a distinction between internal and external particles inside the buffer region. Internal being the particles which are in the half of the buffer region extending towards the not shared particles, and external the ones extending away. A device has full cover of its internal buffer and are responsible for calculating and communicating the particle states which lie inside it. Particle states in the external buffer are received from its neighbor.

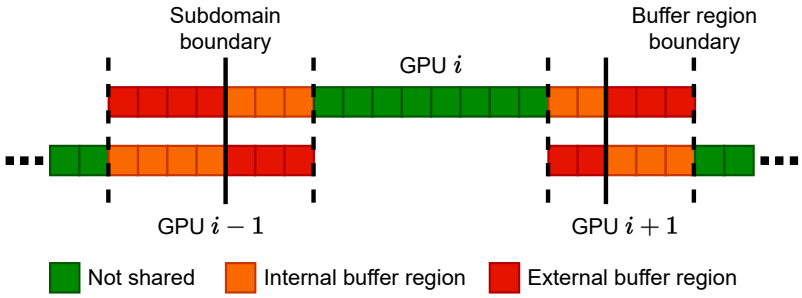


Figure 4.2: The relation between domain decomposition and particle placement in memory.

As the visualization in Figure 4.2 shows, there are duplicates of particles across the devices. If not considered, this would lead to duplicates of particle collision pairs and affecting the loads between particles. The particle collision search for the parallelization step is done such that one neighboring solver is responsible for the collisions occurring within the buffer region and each solver is responsible for collisions between the internal and not shared particles. Thus ensures that all possible particle collision pairs are found and that they are all unique.

4.2 Dynamic decomposition

With the domain decomposition method established there is a related problem to solve. To maximize efficiency, the workload between all GPUs should be evenly distributed. Therefore, some kind of compute load balancing is needed. For the one-dimensional decomposition this is done by moving the subdomain borders such that all devices have a similar load. One way to do this is to ensure that all devices have a similar amount of particles at all times, as is done by Tian et al. and Tsuzuki et al. [19, 22]. This is a straight forward metric to monitor, however it has some flaws. First, amount of particles is not a complete metric for simulation time as the behavior of the particle population can increase or decrease the amount of computations needed (for example, the amount of particle-particle and particle-triangle pairs can vary and how often the collision tree needs to be updated). It is also a bad metric if the GPUs are not identical, as the faster one will always be ahead.

Instead, the time the solvers take to perform the internal particle operations is used as a metric, inspired by the work of Domìnguez [20]. This metric accounts for how different particle behavior impacts the simulation time and does also take into account if a device is performing worse than another. However, it is no guarantee that the balance this method provides is the global optimum as it has no way of exiting a local optima if it enters one. However, there has yet to be seen a case where this balancing method is not sufficient for this decomposition method.

The particle partition to be sent during load balancing is related to the buffer regions. To decrease the load of the slow solver, the subdomain border is traversed by ΔL in the internal direction. This is illustrated in Figure 4.3. As can be seen,

previous external particles are no longer considered for the slow solver. The previous internal particles become external, and the new internal particles are particles which beforehand were not shared. By traversing one ΔL at a time, no extra search steps need to be conducted as the collision trees on the two impacted solvers are ensured to be complete within this delta.

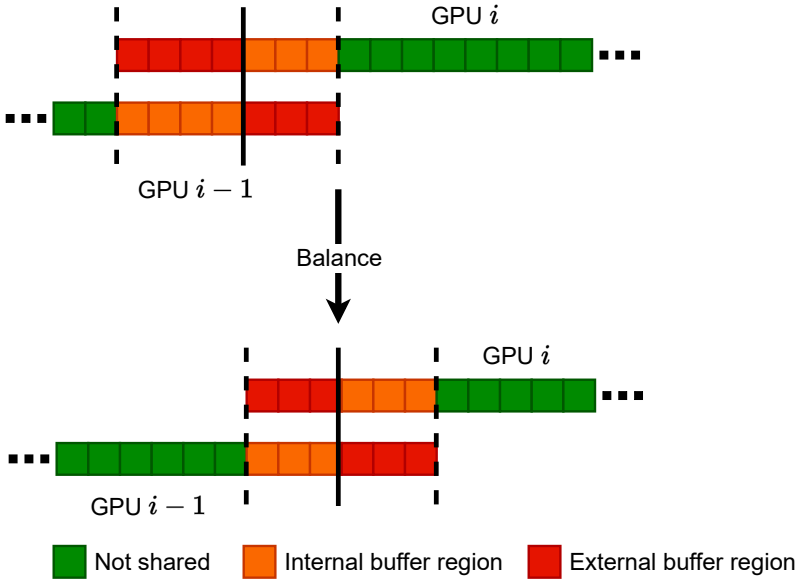


Figure 4.3: Visualization of how the dynamic decomposition shifts which particles are shared between the devices.

4.3 Points of extension

Expanding on the flow-chart in Figure 3.5 in the previous chapter, the points of extension for the parallelization implementation can be inserted. This is done in Figure 4.4. In this flow-chart the parallelization is added as outgoing calls from the core solver loop which represents the fact that the parallelization is merely an extension and does not alter the main solver loop in any way. Before diving deeper into this new flow-chart it helps to understand how the parallelization is initialized before starting the actual simulation.

For each GPU, one solver is initialized with its own material and simulation parameters. Particle- and object data for each respective solver is then distributed to the GPUs by the CPU which manages and launches each solver. Communication and synchronization in the serial flow of the main loop is done within the CPU, while transfer of particle data located on the GPUs is done via the peer-to-peer connection.

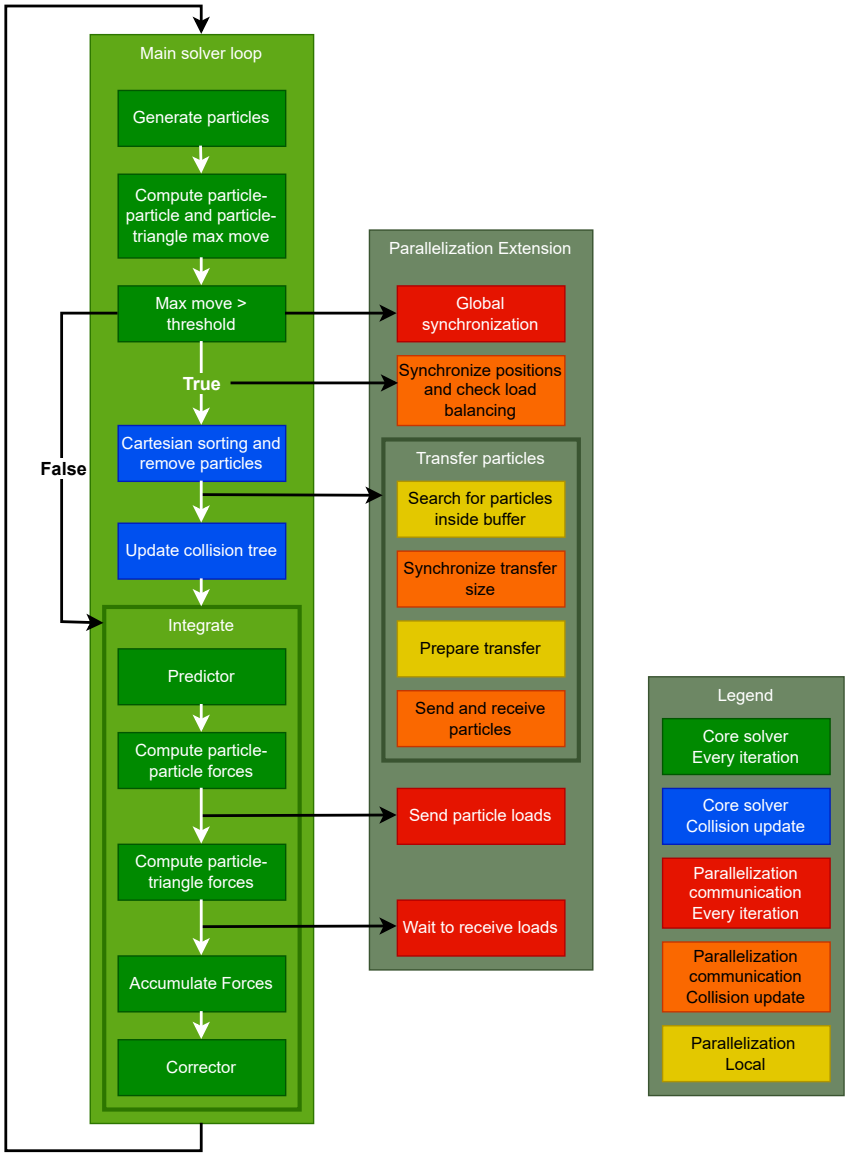


Figure 4.4: Schematic of the main solver loop with all parallelization steps added and their extension points going out from the main loop. Note the that the majority of parallelization operations does not occur on each time step but only occasionally when the collision tree is updated.

Returning to the flow-chart in Figure 4.4, the colors for the parallelization represent how often the step is done and if it involves any communication between solvers. Red indicates parallelization step that requires communication and that is executed at every time step. Orange also requires communication but is only executed at time steps where the collision tree is updated, corresponding to the blue steps in the core solver. The yellow steps are performed along with the orange ones, but does not require any communication between devices.

The purpose and implementation of each extension point in Figure 4.4 will now be presented. Starting from the top with *Global synchronization*, this step ensures

that all solvers agree whether their collision trees should be updated or not. This is the only global synchronization step. The benefit of this global synchronization is that it avoids cases where certain operations only should be executed on a subset of the particle range. Assuming that the load balancing scheme is in use, the global synchronization makes no difference on the total elapsed time.

The next step *Synchronize positions and check load balancing* establish a ground truth of the positions of particles inside each buffer region. Small deviations can occur as floating point numbers are not associative and there is no guarantee that the order of execution in the integration steps are identical across the GPUs. The position synchronization is essential to remove any possible conflict of which side of a subdomain border a particle is.

Whether load balancing should occur or not is also determined at this step. Each solver compares its own elapsed time to its neighbors, and if the difference is above a certain threshold, the subdomain borders will be shifted towards the slower solver. The value for this threshold is case specific, but a decent standard value of 0.9 will generally be sufficient as the time it will take to tune it most likely would be greater than the time saved.

At this point in the flow chart, there is a new operation added in the *Remove particles* box labelled *Cartesian sorting*. It sorts the particles in bins of a set size along the same Cartesian axis of which the domain is decomposed. There are two reasons why this step is added. First, since it is known that we will perform operations on particle ranges contained in set physical spaces (being the buffer regions), we can utilize good memory locality to speed up these operations. Secondly, it adds a point of memory defragmentation which otherwise would be a problem as the transfer of particles easily could lead to heavily fragmented memory.

The next parallelization step *Search for particles inside buffer* is done without any communication and categorize all particles whether they are not shared, or in the internal or external part of a buffer region. At this point, each solver has an updated value of their internal buffer region size which is communicated to all neighbors at the step labelled *Synchronize transfer size*. This allows each solver to do necessary memory preparations to ensure fast transfer of particle data. These preparations are done in step *Prepare transfer*. The details of this step will not be covered as it is of technical aspects not relevant to the method of the parallelization.

With all solvers prepared to transfer particle states, the actual transfer can begin. Even though this is the biggest step in terms of amount of data transferred, it is one of the more simple steps algorithmically. Each solver copies its internal buffer region particles into its neighbors prepared external buffer region memory, and waits to make sure its neighbors have done the same to itself.

A visualization of the buffer search and particle transfer can be seen in Figure 4.5. The first column represents how the particles were categorized at the last particle transfer. The search algorithm is performed, and the particles are recategorized to reflect the current state (column 2). Then finally the particles are transferred, and both buffer regions have identical states.

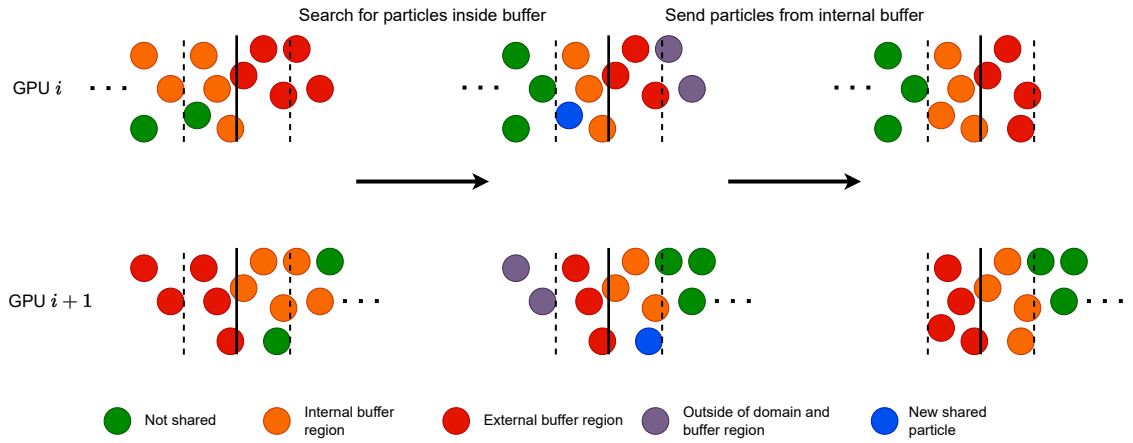


Figure 4.5: The core parts of the particle transfer are visualized starting with the particle search and how particles are classified to be sent, removed or changed to not being shared.

The step *Send particle loads* is the only operation which involves data transfer across GPUs at every time step. Therefore, sending and receiving particle loads is done at two separate steps to reduce the impact on total simulation time. When a solver has computed all particle-particle loads, it transfers the loads corresponding to each buffer region to temporary memory of the neighboring solver. At this point the solver can continue to calculate all particle-triangle loads. The next main solver step is to accumulate all loads to the particles, and therefore it must at this point wait to receive loads (step *Wait to receive loads*) from its buffer regions if these have not already been transferred. This adds asynchronous calculations of the particle-triangle loads to reduce the impact that the parallelization implementation have on total simulation time. However, this asynchronous behavior is not implemented, but the parallelization is set up exactly as described, thus allowing for the asynchronous advantage in the future.

The implementation is designed around the fact that the main solver loop only updates the collision tree occasionally. Since the integration step is performed at each iteration, it is sufficient to only transfer the least amount of data necessary to complete the integration steps, being the force and momenta. By construction, we know that as long as the collision tree does not need an update, there can not possibly be any particles in the buffer region that is not shared between the two devices, and we only need to transfer the complete particle states when the maximum movement criterion is met. The total amount of data per particle transfer at *Send particle loads* with double precision is $2(3 \cdot 8) = 48$ bytes. Compare this with *Synchronize positions* with $3 \cdot 8 = 24$ bytes and *Send and receive particles* with 169 bytes. The amount of data transferred at each time step (48 bytes) is roughly a quarter of the total $24 + 169 = 193$ bytes of the complete state. Additionally, it is worth to remind that the particles affected by the transfers are only the ones inside the buffer regions, which for general parallelized simulation cases only constitutes a small fraction of the total particle population.

5

Verification

The verification has three main purposes. It will ensure that:

- The physics have not changed.
- The dynamic load balancing works as intended.
- The parallelization algorithms work with key features in the core solver such as dilated polyhedrons and particle generators.

If these three points are fulfilled, we can be certain that the parallelization works as intended in terms of functionality, and it can be left to the benchmarking to evaluate its performance.

The first point requires more thorough verification as small deviations in the physics necessarily will not break the simulation. The other two points are essentially binary of either working or crashing the solver. The three points are verified with full simulation cases, one tailored for each point. Each case will be introduced and the result from the verifications presented. The first two cases are also used in the next chapter where the performance is evaluated, see chapter 6.

5.1 Case 1

Case 1 consists of a box filled with spheres over which a plate is dragged, see Figure 5.1. This is a decently static case with low velocities and angular velocities but many collision pairs. The plate both drags particles along with it, but also applies a pressure which is propagated through the particles. Because of the small movements, this case is well suited for examining velocities, angular velocities and kinetic energy. The simulation is easily scalable along the axis perpendicular to the trajectory of the plate, see 5.1. This will come in handy when evaluating performance later.

The physics is verified by comparing the behavior of the multi-GPU solution to that of a simulation calculated on a single GPU with identical initial parameters. This is sufficient since the parallelization steps does not introduce any new contact models or particle representations that should impact the behavior of the particles.

The simulation starts with a pre-generated bed of particles which for the single solver gives a deterministic solution however many times it is calculated. However, this is not the case for the parallel solvers as the order in which operations are executed

are randomized as a result of the randomness in communication order between the solvers. Different orders of execution leads to rounding errors as floating point numbers are not associative. Therefore, the single-GPU solver only realizes one of these variations, while the multi-GPU solvers realize all random variations.

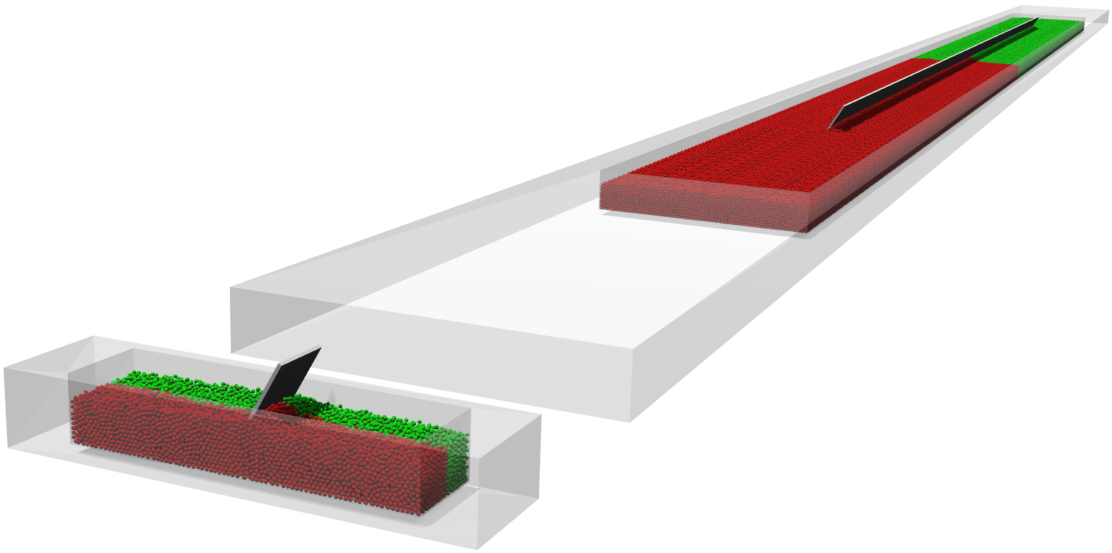


Figure 5.1: Case 1 visualized for two different problem scales. The two colors indicate which device that owns the particles.

All solvers are given an equal chance of resulting in different realizations by initializing each particle with a small randomized velocity. If both solvers solve for the same physics, then the difference in mean values and variations should be insignificant.

The resulting kinetic energy can be seen in Figure 5.2. This is with a population of $20,000$ particles and averaged over 30 runs. The material properties and simulations parameters can be seen in Table 5.1. The mean of the kinetic energy for both implementations are plotted with a band around them covering two standard deviations. As can be seen, there is no significant difference between the two solvers. The plots for velocity norm and angular velocity norm follow the same pattern and are therefore not included. This indicates that the physics have been left unchanged with the new parallelization implementation. More extensive comparisons were performed by closely monitoring the three parameters with the IPS software. The simulation was viewed through cross-sections to verify that any noise in the buffer regions was consistent with the noise elsewhere in the simulation and also compared to the single solver simulation.

Similar plots as in Figure 5.2 could be done for the remaining two cases as well. However, both these cases have much higher absolute velocities and the different realizations would indistinguishable.

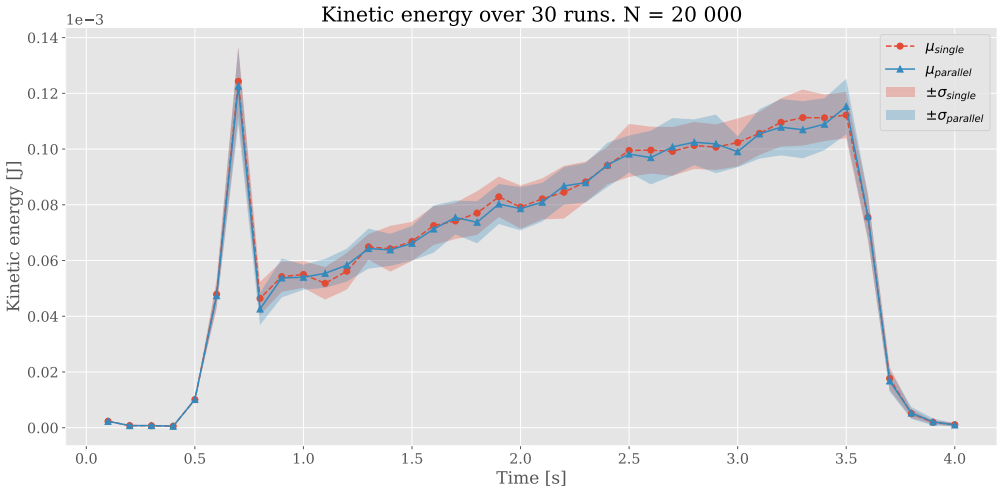


Figure 5.2: The mean and standard deviation of kinetic energy for case 1 with 20 tousand particles averaged over 30 runs. Each run have an initial particle population with small randomized individual velocities.

Table 5.1: Material properties and simulation parameters for case 1.

Material properties	Value	Unit
Particle Diameter	0.005	m
Particle Density	1000	kg/m^3
Wall Density	7000	kg/m^3
Restitution *particle* $\rightarrow$ *particle*	0.5	-
Restitution *particle* $\rightarrow$ *wall*	0.5	-
Particle Young's modulus	1	GPa
Wall Young's modulus	200	GPa
Particle Poisson's ratio	0.25	-
Wall Poisson's ratio	0.25	-
Friction *particle* $\rightarrow$ *particle*	0.2	-
Friction *particle* $\rightarrow$ *wall*	0.2	-
Simulation parameters		
Time step	10^{-5}	s
Total time	4	s
Particle representation	Sphere	-
Particle distribution	Mono distribution	-

5.2 Case 2

The second case have more dynamic particle behavior. Similar to case 1, it also consists of a pre-generated particle population represented as spheres. The particles are dropped from the top of a box, then guided by four inclined planes to the bottom of the box, see Figure 5.3. This stress tests the behavior of the dynamic decomposition as a split put along the vertical axis will have to follow the particles as they are falling down. To examine whether the dynamic decomposition works as intended, the total particle population of each solver is plotted over time. For this case, the number of particles on each solver is expected to be roughly the same.

This simulation was done with 480,000 particles and the material properties and simulation parameters can be seen in Table 5.2. The resulting particle distribution between the two solvers can be seen in Figure 5.4. The simulation is initialized with a split partitioning roughly the same amount of particles in each solver. As can be seen in the plot the amount of particles in each solver stays roughly the same. There seems to be three phases in the plot which in order correspond to free fall, flow between the inclined planes and the majority of particles laying still at the bottom of the box. Difference on computational time for static versus dynamic decomposition will be presented on the next chapter 6.

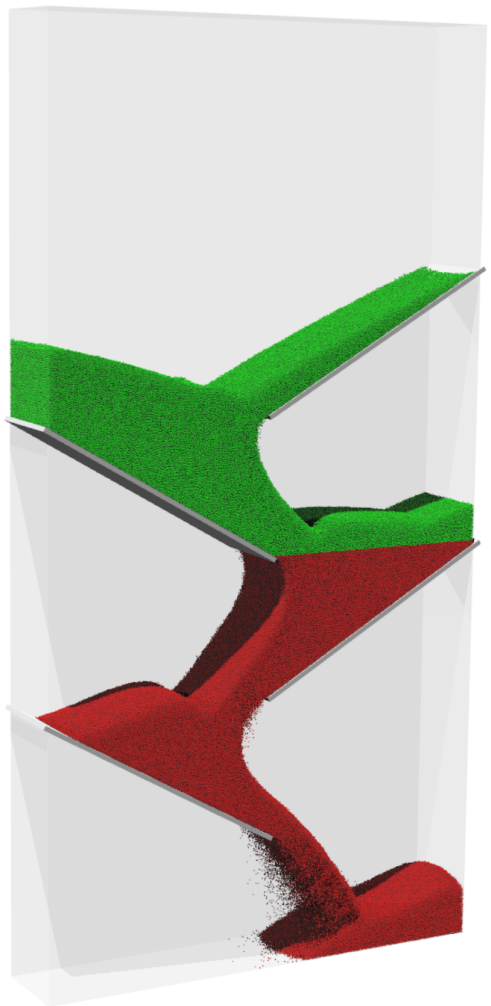


Figure 5.3: Case 2. The particles are generated in a grid at the top of the box and then fall down to the bottom guided by the four inclined planes. The total particle population for this visualization is 2 million spheres. The two colors indicate which device that owns the particles.

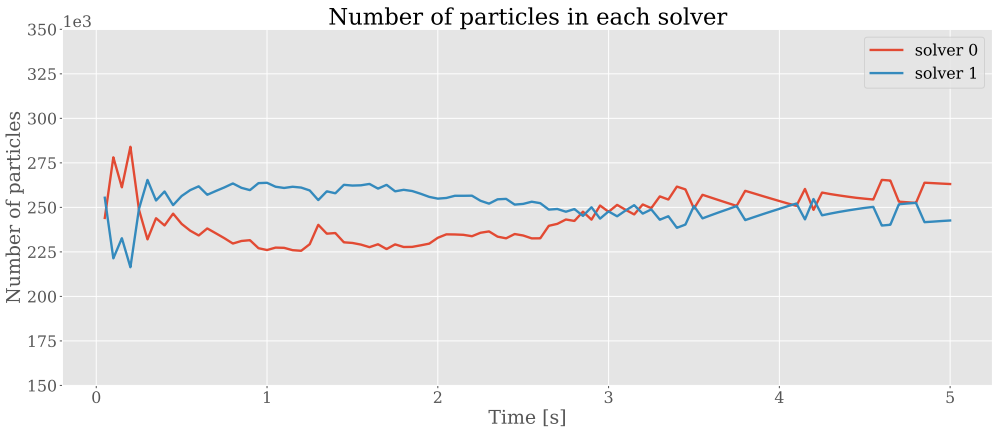


Figure 5.4: The distribution of particles between two solvers with dynamic load balancing. Total number of unique particles are 480, 000. Since there are duplicates of shared particles, the two values does not necessarily add up to exactly 480, 000.

Table 5.2: Material properties and simulation parameters for case 2.

Material properties	Value	Unit
Particle Diameter	0.005	m
Particle Density	1000	kg/m^3
Wall Density	1000	kg/m^3
Restitution *particle* $\rightarrow$ *particle*	0.5	-
Restitution *particle* $\rightarrow$ *wall*	0.5	-
Particle Young's modulus	1	GPa
Wall Young's modulus	1	GPa
Particle Poisson's ratio	0.25	-
Wall Poisson's ratio	0.25	-
Friction *particle* $\rightarrow$ *particle*	0.5	-
Friction *particle* $\rightarrow$ *wall*	0.5	-
Simulation parameters		
Time step	$5 \cdot 10^{-6}$	s
Total time	5	s
Particle representation	Sphere	-
Particle distribution	Mono distribution	-

5.3 Case 3

Case 3 contains complex and moving geometry and other core functions such as generators and destructors and complex shaped particles (dilated polyhedrons). The case can be seen in Figure 5.5 and represents an industrial screen. This type of machine sorts a particle population with a wide distribution of sizes into different fractions. For this machine, this is done by shaking the particles over three screens with different sized gaps allowing smaller particles to fall through. This machine outputs three different fractions and are funneled to their respective conveyor belts through chutes. The generators are placed at the top left of the simulation, inserting particles over the conveyor belt transporting them to the vibrating screens. There

are three destructors, one for each conveyor belt transporting the three fractions. The material properties and simulation parameters can be seen in Table 5.3. At this point we can verify that the approach of implementing the parallelization as an extension have preserved its core functionality and diversity.

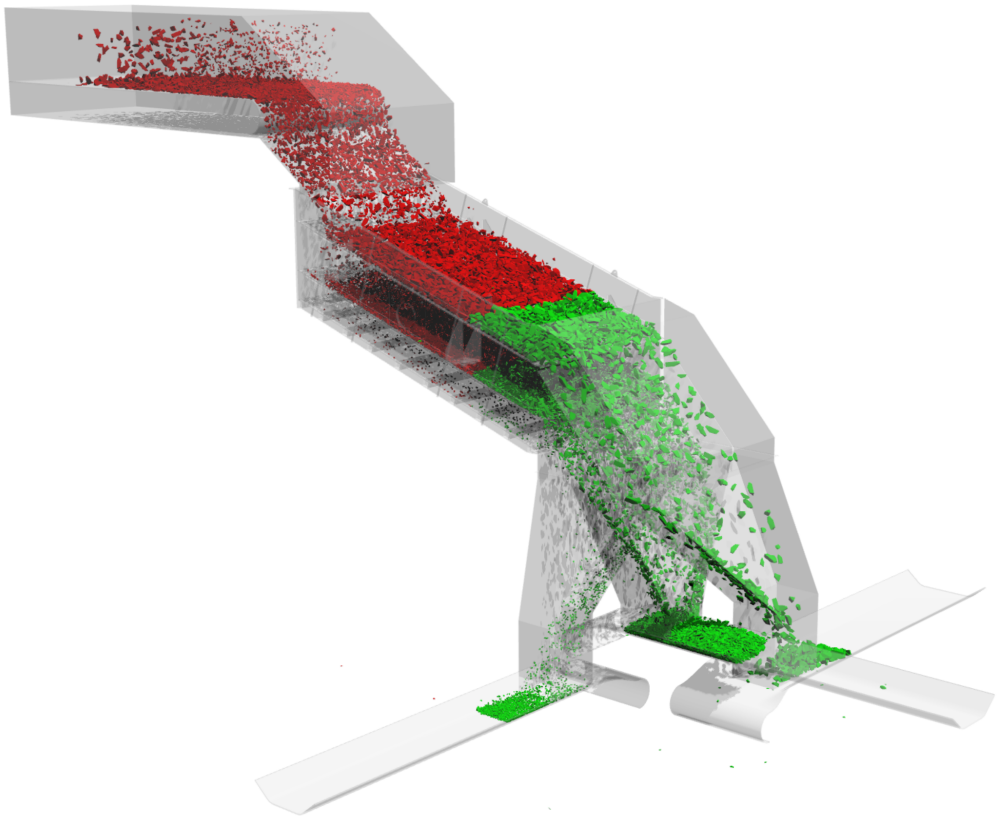


Figure 5.5: Particles are generated at the top left and conveyed to the shaking metal screens which sorts the particles by size. Each size fraction is then funneled through a chute to their respective conveyor belts where they are transported into a destructor, removing them from the simulation. The colors of the particles indicates which device that owns the particle.

This case is suitable to additional verification of physics as it has randomness introduced by the generators unlike case 1 and case 2 which simulates a pre-generated particle population. The comparison between single and parallel solvers are done on the size distributions of each of the three fractions. All sizes are recorded with a frequency of 1 second for each fraction and the resulting distributions can be seen in Figure 5.6. The medium size fraction have the largest number of particles with roughly $15,000$ particles and both solvers are very similar. The small fraction have around $6,000$ particles and the distributions are also very similar. The large fraction only have around 350 particles and therefore have more noise, however, the distributions are still similar with no obvious deviation.

Table 5.3: Material properties and simulation parameters for case 3.

Material properties	Value	Unit
Particle Density	2700	kg/m^3
Wall Density	7800	kg/m^3
Rubber Density	1000	kg/m^3
Restitution *particle* → *particle*	0.25	-
Restitution *particle* → *wall*	0.4	-
Restitution *particle* → *rubber*	0.65	-
Particle Young's modulus	4	GPa
Wall Young's modulus	4	GPa
Rubber Young's modulus	4	GPa
Particle Poisson's ratio	0.28	-
Wall Poisson's ratio	0.3	-
Rubber Poisson's ratio	0.35	-
Friction *particle* → *particle*	0.15	-
Friction *particle* → *wall*	0.2	-
Friction *particle* → *rubber*	0.1	-
Simulation parameters		
Time step	10^{-6}	s
Total time	15	s
Particle representation	Dilated polyhedron	-
Particle distribution 1	Uniform distribution [0.025 ; 0.06]	m
Particle distribution 2	Uniform distribution [0.06 ; 0.08]	m
Particle distribution 3	Uniform distribution [0.08 ; 0.1]	m

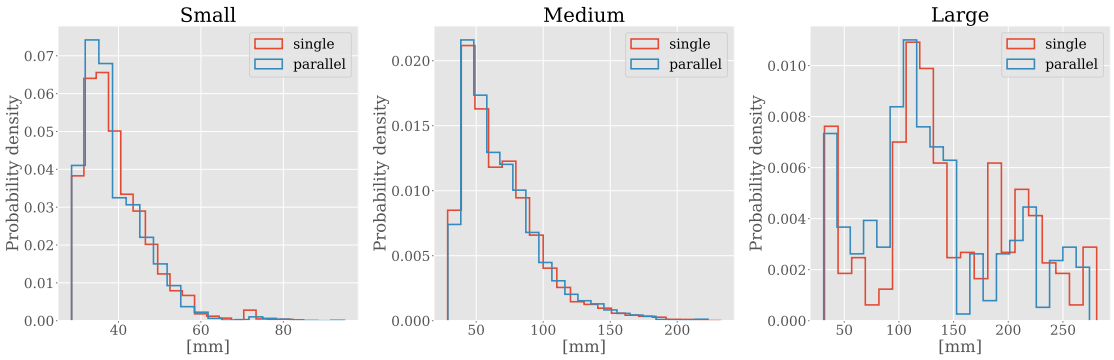


Figure 5.6: Distributions of the three resulting particle fractions from case 3. Total number of particles for each solver and each fraction is roughly: 6, 000 for the small size, 15, 000 for the medium size and 350 for the large size.

6

Performance Evaluation

The performance of the implementation is mainly evaluated using the scalable properties of case 1 presented in the previous chapter. The concepts of strong scaling speedup and weak scaling efficiency will be introduced and measured on the implementation. Analysis of the scaling behavior is then presented with an overview of execution time for some selected operations. There will then be performance comparisons for case 2 showcasing the differences between static and dynamic domain decomposition as well as performance difference for choice of axis to decompose along, and how it affects the different parts of the algorithm.

6.1 Case 1

The problem size of case 1 is easily increased in size by increasing the width of the box and the plate and inserting more particles. This problem is therefore used to get results of strong scaling speedup and weak scaling efficiency. For each size, the domain decomposition is done in such a way that all devices have as close to equal computational load as possible. This is done by turning off the dynamic decomposition and placing the subdomain borders such that all solvers have close to equal number of particles. Since the movement perpendicular to the subdomain borders are minimal, the amount of particles per solver is stable for the whole duration of the simulation.

Strong scaling speedup is the fraction of completing a set size of work for one processor and N processors [52]. More formally it can be defined as

$$\text{Speedup} = \frac{t(1)}{t(N)} \tag{6.1}$$

where $t(1)$ is the amount of time to complete the work for one processor and $t(N)$ for N amount of processors. Strong scaling shows for a fix problem size, how the amount of processors affects the speedup. Depending on the problem, the parallel overhead could cause the speedup to decrease past some number of processors and the strong scaling speedup would then indicate the amount of processors best suited for that case. Ideally, the strong scaling speedup should be completely linear.

Weak scaling efficiency, on the other hand, is the fraction of completing the same size of work for 1 processor and N processors. In other words, the problem size is scaled proportionally to the amount of processors used [52]. The efficiency is given

with the same fraction as the speedup (6.1), where $t(1)$ is the amount of time to complete a problem of size k with one processor and $t(N)$ is the amount of time to complete problem of size $N \cdot k$ with N processors. The efficiency would ideally be constant with value one and the real efficiency indicates how the additional parallel cost depends on the number of processors.

The simulations of case 1 are run on Amazon Web Services (AWS) Cloud Computing Services. Two systems are evaluated, p3.8xlarge and p4d.24xlarge. Their respective computing details can be seen in Table 6.1. The p4d.24xlarge instance have both more capable GPUs and peer-to-peer bandwidth. Due to time constraints reason with the access to the systems the results presented here are simulated on the p3.8xlarge instance. The difference in performance and relative difference between internal memory bandwidth and peer-to-peer bandwidth affects the scaling performance. Comparing the two instances can further highlight the limiting factors.

Table 6.1: Specifications of the three systems used for benchmarking. The first two rows presents the two clusters on AWS used for evaluation of case 1. Amazon provides specification of the cluster architecture[53] and the theoretical bandwidth values are gathered from NVIDIA [54, 55]. The third row presents the in-house system which have mainly been used for testing and evaluation of the decomposition scheme on case 2. The values in parentheses are measured values, while the other values are theoretical according to specifications.

Instance	NVIDIA GPU	GPU Memory	Memory bandwidth (Measured)	peer-to-peer bandwidth (Measured)
p3.8xlarge	4×V100	16 GB	900 (780) GB/s	300 (97) GB/s
p4d.24xlarge	8×A100	40 GB	1935 (1300) GB/s	600 (500) GB/s
in-house	2×RTX 3090	24 GB	(936) GB/s	(100) GB/s

A side note on the peer-to-peer bandwidth of the two instances. The topology of the system impacts the peer-to-peer bandwidth for the p3.8xlarge instance, i.e. depending on topology two devices could have the peer-to-peer bandwidth stated in Table 6.1, or roughly half. On the other hand, the topology of the p4d.24xlarge system is homogenous. The topology also have a similar impact on the latency. As for the results in this report, these parameters generally have a low impact as the difference in speed caused by the topology is often on another scale compared to total simulation time.

Starting with Figure 6.1, we present particle bed simulations on 1, 2, 3, and 4 GPUs for particle populations doubled from $20,000$ up to 5.12 million, for the p3.8xlarge instance. The parallel solvers perform substantially worse on small particle populations which will be discussed further on. It is first at 5.12 million particles that a proper scaling is achieved, still with a very minor difference between 3 and 4 GPUs. The speedup from 1 to 4 GPUs on the biggest case is 2.9 which can be compared to other commercial software of roughly 2.8 [56]. However, it is not suitable to directly compare these numbers as there are numerous different aspects that affect it, i.e. simulation case, population size and time step.

Note, that interface costs are removed from the overall cost for all performance measurements. This is done as some of the operations performed in the interface

are currently highly serial. These are operations such as writing the h5-files for the particle data. Parallelization of these operations are out of scope as they concern core parts of the solver and not strictly necessary for a DEM-solver.

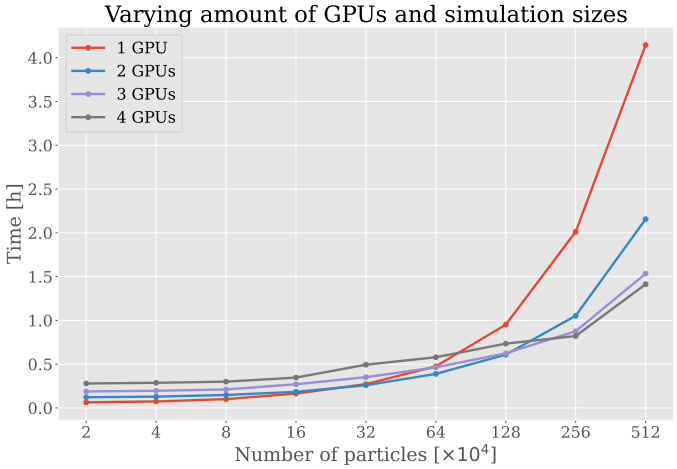
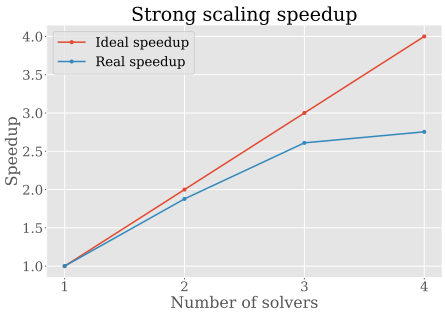
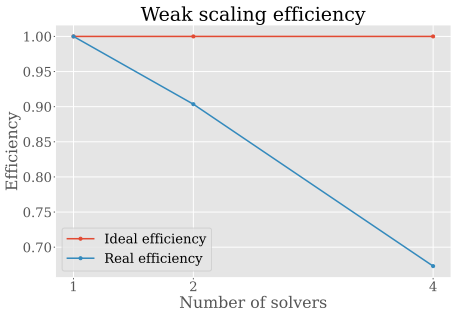


Figure 6.1: Scaling of the p3.8xlarge instance on case 1 varying from 1 to 4 GPUs and doubling problem size ranging from 20,000 particles to 5.12 million particles. Proper scaling is achieved for large enough problem sizes.

Both strong and weak scaling can be extracted from the data of Figure 6.1. The resulting curves can be seen in Figure 6.2. Figure 6.2a contains the strong scaling speedup and as can be seen, the first three data points lie quite close to the ideal scaling curve. However, it is apparent that the performance suffers when the relative problem size shrinks. This is clearly visible in Figure 6.2b where loss in efficiency looks to be close to proportional to number of GPUs.



(a) Strong scaling speedup with comparison to the linear ideal scaling. Case 1 with 5.12 million particles.



(b) Weak scaling efficiency with comparison to the theoretical ideal of perfect efficiency. Case 1 with 1.28 million particles per GPU.

Figure 6.2: Strong and weak scaling for simulation of case 1.

A similar plot to the one in Figure 6.1 is presented next. Here, the scaling performance of only the parallelization operations are plotted, see Figure 6.3. The single GPU has a constant value of 0 as it does not execute any of the parallelization operations. The non-smooth behavior most likely comes from the difference in the multi-GPU topology between runs as there is no guarantee that it is constant between reboots of the AWS instance. The scaling for increasing problem size should be close to linear which seems to be the case when not considering the smallest problem sizes and accounting for the topology. Most of the parallelization operations do not decrease in problem size with increasing number of GPUs as they mostly consider the particles inside the buffer regions. Therefore, the increase in time for constant problem size with increasing number of GPUs should also be linear, which seems to be the case.

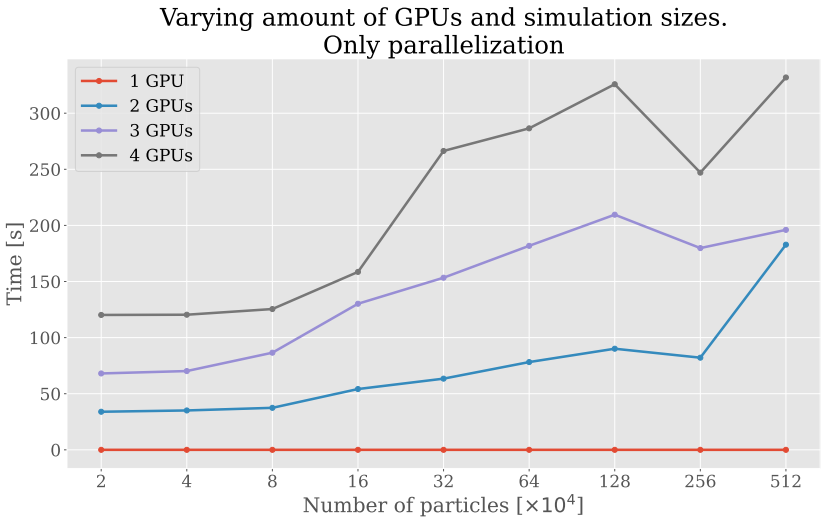


Figure 6.3: Scaling of the parallelization operations on the p3.8xlarge instance on case 1 varying from 1 to 4 GPUs and doubling problem size ranging from 20,000 particles to 5.12 million particles.

In order to analyze the performance more thoroughly, the total execution time of some selected algorithms are presented. Again, interface costs are not included. Execution times for the p3.8xlarge instance simulating 5.12 million particles with 2, 3 and 4 GPUs are presented in Figure 6.4. First, it is obvious that the parallelization steps (all steps between *Global sync.* and *Sync. loads*, inclusive) takes a small fraction of the simulation time, 2.6%, 3.6% and 6.5% for 2, 3 and 4 GPUs respectively. It is also noticeable that steps *Accumulate, Init, Overlaps, Integrate* and *Collision tree (search particles)* seems to scale well, close to linear. These operations are completely done on each GPU with no peer-to-peer communication or host communication. However, the remaining operations are more suspicious as they seem to quite drastically increase in execution time as more devices are added. Keep in mind that the workload for each device decreases as the total number of devices increase since the total simulation size is constant.

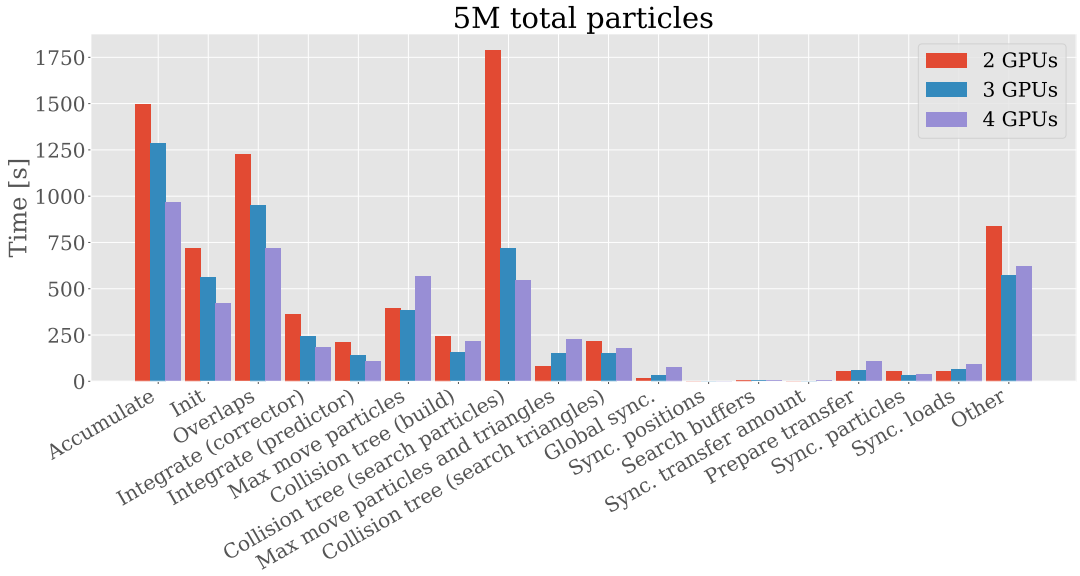


Figure 6.4: Execution time for selected algorithmic operations. The execution time for the operations implemented for the parallelization have a small impact. It is also visible that some of the core solver operations scale properly and some scale inversely with increasing number of GPUs. The total execution times were for 2 GPUs: 7762 s, 3 GPUs: 5521 s, 4 GPUs: 5086 s.

The difference between the operations with poor and good scaling is exaggerated in Figure 6.5. These measurements are done such that each device has a constant workload at $20,000$ particles. There are barely any difference in execution time for the first 5 operations even though amount of GPUs increase, which is the desired behavior. The parallelization operations do slightly increase with increasing number of GPUs which is expected as a bigger GPU-system requires more communication. However, for the operations that scale poorly there is a drastic increase in execution time with increasing amount of GPUs.

NVIDIAs software Nsight System can be used to analyze all calls and their respective execution time that is done through CUDA. When analyzing the problematic parts it seems like there are excessive allocations done in some of the Thrust calls within these operations. However, this would not explain the difference experienced when adding more GPUs. Further analysis revealed that each allocation were substantially slower and according to this blog post [57] the slowdown is proportional to amount of GPUs added and a result of how allocations are done on the shared peer-to-peer memory buffer. There are methods to reduce and even avoid this issue and will be discussed in the next chapter 7.

The p4d.24xlarge instance have similar behavior to the p3.8xlarge given that the problem size is larger to compensate for the better performance on the p4d.24xlarge instance. The difference between the operations with good and poor scaling is further emphasized on the p4d.24xlarge instance because of the more capable hardware. Similarly, the parallelization have a larger relative cost. However, further testing on the p4d.24xlarge is of great interest to investigate trends and the effect on scalability.

A final note on these results. The implementation done within the project scope performs well and have low impact on the bad scaling for small problem sizes. The approach of only synchronizing the absolute necessary information at every time step is successful in minimizing the parallel overhead. The local operation *Search buffers* is barely noticeable and while *Prepare transfer* clearly have an impact it is on par with the other more expensive parallelization operations. It is also noteworthy that *Prepare for transfer* have obvious improvement potential as it currently performs a temporary copy which with some work on the core solver could be avoided.

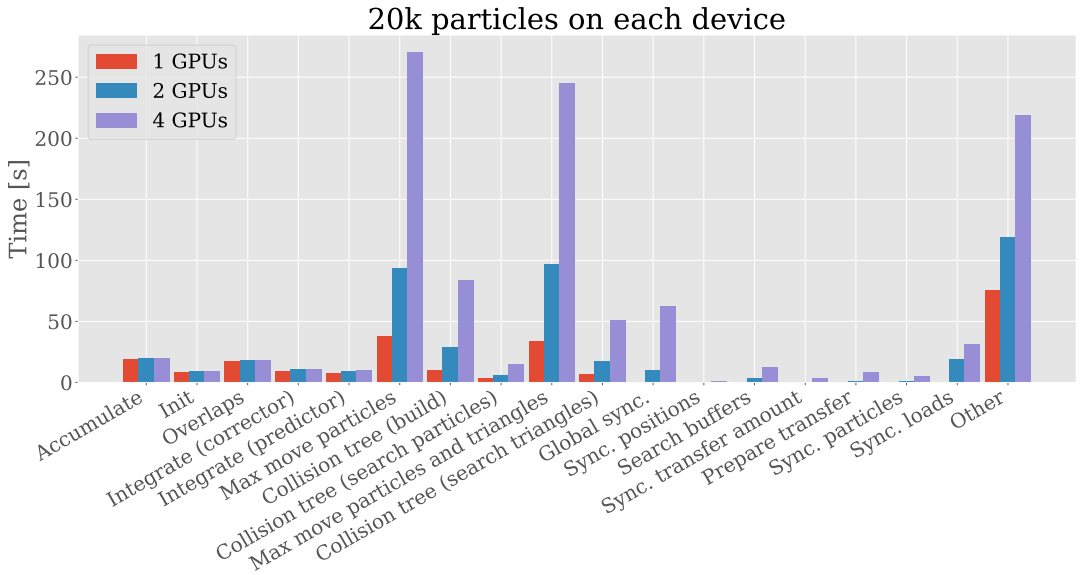


Figure 6.5: Execution time for selected algorithmic operations with a small constant work load for all devices. The parallelization operations are still comparatively small, however the problem with some of the core operations are clearly visible. Total execution times were for 1 GPU: 440 s, 2 GPUs 677 s, 4GPUs 1006 s.

6.2 Case 2

Two performance measurements are presented for case 2. First, the difference in execution time for the different operations depending on along which axis the domain is decomposed. Then, the difference between static and dynamic decomposition with a horizontal subdomain border. These measurements are done on the in-house system with two NVIDIA RTX 3090 GPUs, see Table 6.1.

Figure 6.6: Execution time with dynamic decomposition along different axes. Total execution time for axis 0: 3610 seconds, axis 1: 3367 seconds and axis 2: 3404 seconds. Particle population size of 480, 000.

The results of decomposing along the three different axes can be seen in Figure 6.6. Axis 0 splits the box (see Figure 5.3) along its depth, axis 1 along its width and axis 2 along its height. The simulation contains a total of 480, 000 particles. The overall worst execution time of 3610 seconds is achieved with axis 0, which is expected as this domain decomposition by far gives the largest number of particles in the buffer regions. Axis 1 and 2 have similar total execution time, 3367 and 3404 seconds respectively, with axis 1 being slightly ahead. The case was also simulated with a larger particle population of 2 million particles which put axis 2 slightly ahead. The choice of axis to split along is crucial to minimize execution time, but it is not always apparent what the correct answer is without testing.

The split along axis 2 relies completely on the dynamic decomposition as without it, only one device at a time would perform most of the calculations. The particle distribution between the two solvers with no dynamic decomposition can be seen in Figure 6.7. This is for a simulation with identical setup as the previous case with 480, 000 particles but with dynamic decomposition turned off. Compare this to Figure 5.4 which was presented as verification that the dynamic decomposition behaves as expected. This is not desired and the impact on execution time can be seen in Figure 6.8. Solver 1 with the static domain spends the majority of its execution time waiting for solver 0 at step *Sync. loads*. This because solver 1 have ownership of the particles for a shorter time as they quickly fall out of its static domain. Execution time between the two dynamically decomposed solvers are much more even and have a direct impact on the simulation time. Total execution time for the static decomposition was 4553 seconds while the dynamic decomposition took roughly 3/4 of the time with 3404 seconds.

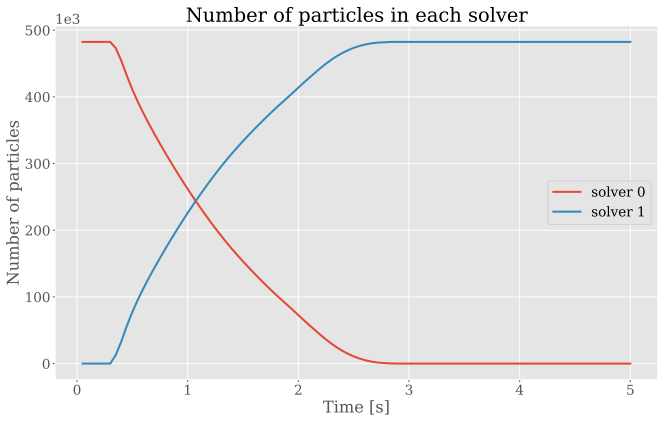


Figure 6.7: Number of particles on each solver with static decomposition along axis 2. Compare dynamic decomposition in Figure 5.4.

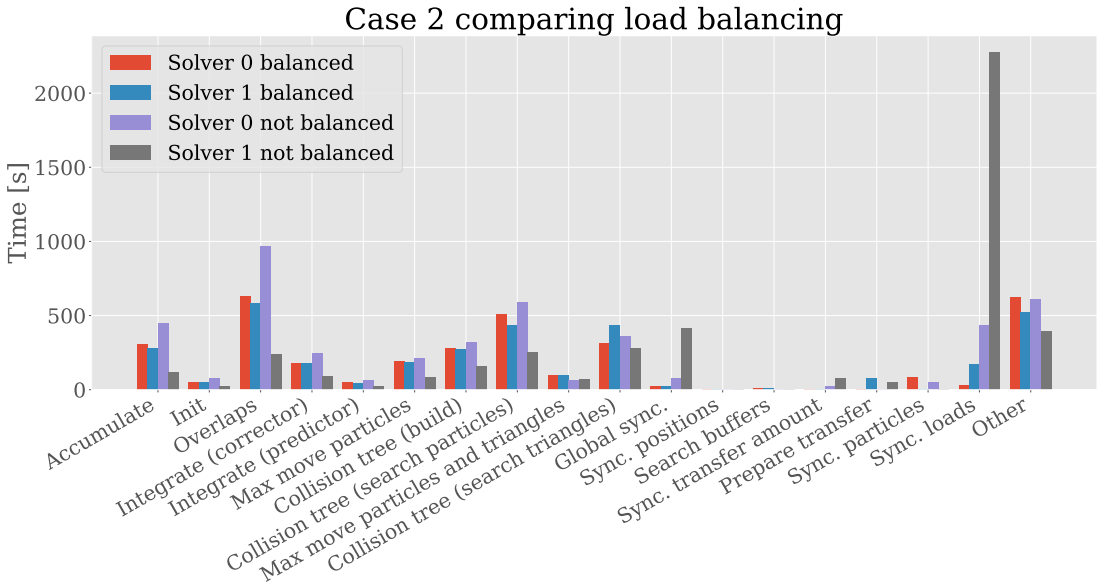


Figure 6.8: Execution time with dynamic and static decomposition along axis 2. Total execution time with dynamic decomposition: 3404 seconds, and with static decomposition: 4553 seconds. Particle population size of 480, 000.

7

Conclusion

As an overall conclusion, the presented method for static and dynamic domain decomposition is efficient and simplistic. The algorithms developed for the parallelization implementation only take as low as 2.6% of total execution time. Execution time for some operations on the core solver increase proportionally with increasing amount of GPUs in peer-to-peer mode.

On a more specific level, the chosen domain decomposition scheme is simplistic and therefore limited in its capability to minimize number of shared particles. Demify® is developed for GPU acceleration and commercially available multi-GPU systems with peer-to-peer connection commonly have 4-8 GPUs. Therefore, the topological limitation of the one-dimensional domain decomposition is less likely to be met compared to if the implementation was intended to be run on thousands of nodes. Additionally, the simple topology lends itself to trivial implementation of fast sort and search algorithms and memory locality in an otherwise very dynamic flow of data. This decreases the additional cost of the parallelization-specific operations.

The dynamic domain decomposition performs well with basically no additional algorithmic costs. The difference in the peer-to-peer data transfer is that all data is flowing in one direction, compared to a regular particle transfer step where particles states are synchronized in both directions. The dynamic domain decomposition is effective in spreading out the computational load across the GPUs, lowering the total execution time. Alternative methods were considered, such as equalizing computational load based on number of particles on each GPU. This was deemed as a worse approach as it does not capture any computational difference based on either particle dynamics or hardware. However, there is no guarantee that the chosen dynamic domain decomposition method converges to a global optimal solution as each GPU only consider the elapsed time of its neighboring GPUs and have no estimation of the global computational domain.

The most expensive operations performed in the parallelization steps are *Prepare transfer, Sync. particles* and *Sync. loads*. The fact that the latter two are expensive validates the intention of minimizing the amount synchronization steps that must be performed at every iteration. The per particle data amount transferred at every time step is roughly 1/4 of the complete particle state and the collision tree could remain valid for hundreds of iterations, depending on case. It is also worth to note that the synchronization of particle loads is implemented to allow for asynchronous particle-triangle calculations, further reducing the impact of the parallelization.

The local operation *Prepare transfer* also has a well known possible improvement. The current implementation must consider left and right neighbors differently because of the memory layout. The difference results in that solvers with neighbors to the left must perform a temporary copy of the particle states. The proposed improvement is to store the particle states centered in the allocated memory and thus having memory buffers extending in both directions, allowing for equally efficient transfer of particle states independent on left and right neighbor.

The performance analysis revealed, in addition to good performance of the parallel algorithms, that some parts in the main loop of the core scaled poorly on multi-GPU systems. The root cause could be found by measuring execution time of different CUDA calls with NVIDIAs software Nsight Systems. It was found that CUDA memory allocation scaled proportionally with increasing number of GPUs in peer-to-peer mode. Some functions in the Thrust library do memory allocations that can in some cases be avoided. There currently is work in progress to reduce the number of allocations and it seems like this approach greatly increase the overall scaling performance of the solvers. Results from the improvement is not included as it is not finalized and outside the scope of the project.

The remaining operations affected by the CUDA allocation scheme are done when updating the collision tree and a fix for this is out of the scope for this thesis. In the meantime the collision detection distance (maximum movement) should be considered carefully when using the parallelization implementation. A longer collision detection distance would cause the collision tree to be updated less frequently and reducing the number of allocations. A specialized memory optimized collision detection scheme could potentially be needed to completely remove the dependency on additional allocations during runtime.

The code was developed and implemented with the intention of reaching the production branch of the Demify® codebase. A not negligible part of the project has been dedicated to make sure that the code written fits in the standards and conventions of said codebase. As described in section 3.2 the parallelization implementation fits as an extension to the core solver and therefore follows certain set of rules that are universal for all extensions. Because of this, integration with core functionalities, such as generators and destructors, is trivial and validates the approach.

Continuing with technical code aspects, the parallelization can only be accessed through the Python text-based interface and not in the IPS graphical user interface. Regarding the interface, at the current state there are too many additional simulation parameters that must be specified when using the parallelizatione extension. Ideally, it would be possible to only demand the user to specify the axis to decompose the domain along for general cases.

In addition to user interface improvements, similar technical improvements are to: automatically adapt to GPU node topology, add asynchronous behaviour throughout the solver to decrease parallelization impact, improve how data files are written by the parallel solvers, add transfer of particle data structure for history effects. Particle history is necessary to use the standard contact model in Demify®.

Lastly, future work with a larger scope is to expand the parallelization to multi-node CPU systems via MPI. A more sophisticated domain decomposition would probably be necessary for large systems with hundreds or thousands of nodes. Operations specific to the one-dimensional decomposition and multi-GPU communication are well separated from the rest of the code, which is written to deal with general communication patterns. Thus, functionality such as a two-dimensional domain decomposition and multi-CPU communication would build upon the implemented framework.

Bibliography

[1] Sveriges geologiska undersökning (2021) Grus, sand och krossberg 2020.

[2] Sveriges geologiska undersökning (2022) Bergverksstatistik 2021.

[3] Statistikmyndigheten SCB (2020) Slutanvändning (MWh) efter region, förbrukarkategori, bränsletyp och år.

[4] P. A. Cundall and O. D. L. Strack A discrete numerical model for granular assemblies, In: *The Essence of Geotechnical Engineering: 60 years of Géotechnique*, pp. 305-329. https://doi.org/10.1680/geot.1979.29.1.47

[5] Paul W. Cleary, DEM prediction of industrial and geophysical particle flows, Particuology, Volume 8, Issue 2, 2010, Pages 106-118, ISSN 1674-2001, https://doi.org/10.1016/j.partic.2009.05.006.

[6] S.D. Liu, Z.Y. Zhou, R.P. Zou, D. Pinson, A.B. Yu, Flow characteristics and discharge rate of ellipsoidal particles in a flat bottom hopper, Powder Technology, Volume 253, 2014, Pages 70-79, ISSN 0032-5910, https://doi.org/10.1016/j.powtec.2013.11.001.

[7] Fan Geng, Zhulin Yuan, Yaming Yan, Dengshan Luo, Hongsheng Wang, Bin Li, Dayong Xu, Numerical simulation on mixing kinetics of slender particles in a rotary dryer, Powder Technology, Volume 193, Issue 1, 2009, Pages 50-58, ISSN 0032-5910, https://doi.org/10.1016/j.powtec.2009.02.005.

[8] William R. Ketterhagen, Mary T. Am Ende, Bruno C. Hancook, Process Modeling in the Pharmaceutical Industry using the Discrete Element Method, Journal of Pharmaceutical Sciences, Volume 98, Issue 2, 2009, Pages 442-470, ISSN 0022-3549, https://doi.org/10.1002/jps.21466

[9] Yan, B., Regueiro, R.A. A comprehensive study of MPI parallelism in three-dimensional discrete element method (DEM) simulation of complex-shaped granular particles. *Comp. Part. Mech.* 5, 553–577 (2018). https://doi.org/10.1007/s40571-018-0190-y

[10] Ferellec, JF., McDowell, G.R. A method to model realistic particle shape and inertia in DEM. *Granular Matter* **12**, 459–467 (2010). https://doi.org/10.1007/s10035-010-0205-8

[11] A. Hopkins, M. (2014), "Polyhedra faster than spheres?", *Engineering Computations*, Vol. 31 No. 3, pp. 567-583. https://doi.org/10.1108/EC-09-2012-0211

[12] J.Q. Gan, Z.Y. Zhou, A.B. Yu, A GPU-based DEM approach for modelling of particulate systems, Powder Technology, Volume 301, 2016, Pages 1172-1182, ISSN 0032-5910, https://doi.org/10.1016/j.powtec.2016.07.072. (https://www.sciencedirect.com/science/article/pii/S0032591016304648)

[13] Joshua A. Anderson, Chris D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, *Journal of Computational Physics*, Volume 227, Issue 10, 2008, Pages 5342-5359, ISSN 0021-9991, https://doi.org/10.1016/j.jcp.2008.01.047.

[14] Erich Elsen, V. Vishal, Mike Houston, Bijay Pande, Pat Hanrahan, Eric Darve, N-Body Simulations on GPUs, 2007, `https://doi.org/10.48550/arXiv.0706.3060`

[15] Jingwei Zheng, Xuehui An, Miansong Huang, GPU-based parallel algorithm for particle contact detection and its application in self-compacting concrete flow simulations, *Computers & Structures*, Volumes 112–113, 2012, Pages 193-204, ISSN 0045-7949, `https://doi.org/10.1016/j.compstruc.2012.08.003`

[16] Dong, Youkou and Yan, Dingtao and Cui, Lan, An Efficient Parallel Framework for the Discrete Element Method Using GPU, Applied Sciences, Volume 12, Issue 6, 2022, ISSN 2076-3417, `10.3390/app12063107`

[17] New Mexico State University, Message Passing Interface, Retrieved: 2023-05-11, `https://hpc.nmsu.edu/discovery/mpi/introduction/`

[18] Eugenio Rustico, Giuseppe Bilotta, Giovanni Gallo, Alexis Hérault, Ciro Del Negro, Robert A. Dalrymple, A journey from single-GPU to optimized multi-GPU SPH with CUDA, 2013

[19] Satori Tsuzuki, Seiya Watanabe, Takayuki Aoki, Large-scale DEM Simulations for Granular Dynamics, Tsubame esj., Volume 13, 2015, Pages 13-18, `https://www.gsic.titech.ac.jp/TSUBAME_ESJ`

[20] J.M. Domínguez, A.J.C. Crespo, D. Valdez-Balderas, B.D. Rogers, M. Gómez-Gesteira, New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters, Computer Physics Communications, Volume 184, Issue 8, 2013, Pages 1848-1860, ISSN 0010-4655, `https://doi.org/10.1016/j.cpc.2013.03.008.`

[21] Park S-H, Jo YB, Ahn Y, Choi HY, Choi TS, Park S-S, Yoo HS, Kim JW and Kim ES (2020) Development of Multi-GPU–Based Smoothed Particle Hydrodynamics Code for Nuclear Thermal Hydraulics and Safety: Potential and Challenges. Front. Energy Res. 8:86. `doi:10.3389/fenrg.2020.00086`

[22] Yuan Tian, Sheng Zhang, Ping Lin, Qiong Yang, Guanghui Yang, Lei Yang, Implementing discrete element method for large-scale simulation of particles on multiple GPUs, Computers & Chemical Engineering, Volume 104, 2017, Pages 231-240, ISSN 0098-1354, `https://doi.org/10.1016/j.compchemeng.2017.04.019.`

[23] A. Bilock, A GPU Polyhedral Discrete Element Method, M.S. thesis, Dept. Mathematical Sciences, Chalmers University of Technology, Gothenburg,

Sweden, 2020. [online]. Available: https://hdl.handle.net/20.500.12380/300900

[24] J. A. Göransson, A coupled multibody and discrete element approach for roller compaction dynamics, M.S. thesis, Dept. Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, Sweden, 2020, [online]. Available: https://hdl.handle.net/20.500.12380/301057

[25] J. Quist, K. Jareteg, A. Bilock, A. Persson, Undersökning av separationseffekter vid kompaktering av obundna material, Fraunhofer Chalmers, SBUF, ID: 13820, 2021, https://vpp.sbuf.se/Public/Documents/ProjectDocuments/005ad6c4-5b0a-4a78-a991-aadabad36a79/FinalReport/SBUF%2013820%20Slutrapport%20Unders%C3%B6kning%20av%20separationseffekter%20vid%20kompaktering%20av%20obundna%20material.pdf

[26] NVIDIA, CUDA, CUDA C++ Programming Guide, Version 12.0

[27] Munjiza A., Andrews K. R. F., NBS contact detection algorithm for bodies of similar size, International Journal for Numerical Methods in Engineering, Volume 43, 1998, Pages 131-149, https://doi.org/10.1002/(SICI)1097-0207(19980915)43:1<131::AID-NME447>3.0.CO;2-S

[28] Williams John R., Perkins Eric, Cook Ben, A contact algorithm for partitioning N arbitrary sized objects, Engineering computations, Volume 21 2004, Pages 235-248, https://doi.org/10.1108/02644400410519767

[29] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, Fast BVH Construction on GPUs, In: *Computer Graphics Forum*, Volume 28, 2009, Pages 375-384

[30] J. Pantaleoni, D. Luebke., HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry, In: *Proceedings of the Conference on High Performance Graphics*, HPG '10, Saarbrucken, Germany, Eurographics Association, 2010, Pages 87–95,

[31] K. Garanzha, J. Pantaleoni, D. McAllister, Simpler and faster HLBVH with work queues, In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, Association for Computing Machinery, New York, NY, USA, 2011, Pages 59–64, https://doi.org/10.1145/2018323.2018333

[32] T. Karras, Maximizing Parallelism in the Construction of BVHs, Octress, and K-d Trees, In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, Paris, France: Eurographics Association, 2012, Pages 33-37, http://dx.doi.org/10.2312/EGGH/HPG12/033-037

[33] Christer Ericson, *Real Time Collision Detection*, USA: CRC Press, Inc., 2004

[34] H. Hertz. "On the contact of elastic solids". In: *Z. Reine Angew. Mathematik* 92, 1881, Pages 156-171. https://archive.org/details/cu31924012500306/page/n193/mode/2up

[35] R. D. Mindlin. "Compliance of elastic bodies in contact". In: *J. Appl. Mech.* 16, 1949, Pages, 259-268. https://archive.org/details/cu31924012500306/page/n193/mode/2up

[36] N.G. Deen, M. Van Sint Annaland, M.A. Van der Hoef, J.A.M. Kuipers, Review of discrete particle modeling of fluidized beds, Chemical Engineering Science, Volume 62, Issues 1–2, 2007, Pages 28-44, ISSN 0009-2509, https://doi.org/10.1016/j.ces.2006.08.014.

[37] H. Kruggel-Emden, M. Sturm, S. Wirtz, V. Scherer, Selection of an appropriate time integration scheme for the discrete element method (DEM), Computers & Chemical Engineering, Volume 32, Issue 10, 2008, Pages 2263-2279, ISSN 0098-1354, https://doi.org/10.1016/j.compchemeng.2007.11.002.

[38] Nicolin Govender, Daniel N. Wilke, Chuan-Yu Wu, Raj Rajamani, Johannes Khinast, Benjamin J. Glasser, Large-scale GPU based DEM modeling of mixing using irregularly shaped particles, Advanced Powder Technology, Volume 29, Issue 10, 2018, Pages 2476-2490, ISSN 0921-8831, https://doi.org/10.1016/j.apt.2018.06.028.

[39] Mehrdad Pasha, Colin Hare, Mojtaba Ghadiri, Alfeno Gunadi, Patrick M. Piccione, Effect of particle shape on flow in discrete element method simulation of a rotary batch seed coater, Powder Technology, Volume 296, 2016, Pages 29-36, ISSN 0032-5910, https://doi.org/10.1016/j.powtec.2015.10.055.

[40] H. Kruggel-Emden, S. Rickelt, S. Wirtz, V. Scherer, A study on the validity of the multi-sphere Discrete Element Method, Powder Technology, Volume 188, Issue 2, 2008, Pages 153-165, ISSN 0032-5910, https://doi.org/10.1016/j.powtec.2008.04.037.

[41] F. Alonso-Marroquin and Yucang Wang. An efficient algorithm for granular dynamics simulation with complex-shaped objects. 2008. https://doi.org/10.48550/arXiv.0804.0474

[42] E. G. Gilbert, D. W. Johnson, S. S. Keerthi, A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space, In: *Journal of robotics and automation* Volume 4, Number 2, 1988, Pages 193-203

[43] Nye, B., Kulchitsky, A. V., & Johnson, J. B., Intersecting dilated convex polyhedra method for modeling complex particles in discrete element method, International journal for numerical and analytical methods in geomechanics, 38(9), 2014, Pages 978–990, https://doi.org/10.1002/nag.2299

[44] Nicolin Govender, Daniel N. Wilke, Schalk Kok, Rosanne Els, Development of a convex polyhedral discrete element simulation framework for NVIDIA Kepler based GPUs, Journal of Computational and Applied Mathematics, Volume 270, 2014, Pages 386-400, ISSN 0377-0427, https://doi.org/10.1016/j.cam.2013.12.032.

[45] Nicolin Govender, Daniel N. Wilke, Schalk Kok, Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture SoftwareX Volume

5, 2016, Pages 62-66, ISSN 2352-7110 https://doi.org/10.1016/j.softx.2016.04.004

[46] Nassauer, B., Liedke, T., Kuna, M. Polyhedral particles for the discrete element method. Granular Matter Volume 15, 2013, Pages 85-93, ISSN 1434-7636, https://doi.org/10.1007/s10035-012-0381-9

[47] Nassauer, B., Kuna, M. Contact forces of polyhedral particles in discrete element method. Granular Matter Volume 15, Pages 349–355 2013. https://doi.org/10.1007/s10035-013-0417-9

[48] Nicolin Govender, Daniel N. Wilke, Chuan-Yu Wu, Johannes Khinast, Patrick Pizette, Wenjie Xu, Hopper flow of irregularly shaped particles (non-convex polyhedra): GPU-based DEM simulation and experimental validation, Chemical Engineering Science, Volume 188, 2018, Pages 34-51, ISSN 0009-2509, https://doi.org/10.1016/j.ces.2018.05.011

[49] N. Govender, D. N. Wilke, C. Wu, R. Rajamani, J. Khinast, B. J. Glasser, Large-scale GPU based DEM modeling of mixing using irregularly shaped particles, In: *Advanced Powder Technology*, 29.10, 2018, Pages 2476-2490.

[50] N Govender, D. N. Wilke, BlazeDEM-GPU for simulations where particle shape matters, In: *Proceedings of the 8th International Conference on Discrete Element Methods (DEM8)*, 2019.

[51] Thrust, Retrieved 2023-05-17, https://thrust.github.io/

[52] hpc-wiki, Last updated: 2022-06-03, https://hpc-wiki.info/hpc/Scaling

[53] AWS instance types, 2023-06-08, https://aws.amazon.com/ec2/instance-types/

[54] NVIDIA V100 TENSOR CORE GPU datasheet, NVIDIA, 2020, https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf

[55] NVIDIA A100 TENSOR CORE GPU datasheet, NVIDIA, 2022, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf

[56] Rocky with Multi-GPU: which hardware is best for you?, Rocky, Published 2021-08-30, https://rocky.esss.co/blog/rocky-multi-gpu-which-hardware-is-best/

[57] Multi-GPU Programming, Georgii Evtushenko, Published 2020-07-20, https://medium.com/gpgpu/multi-gpu-programming-6768eeb42e2c

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY