computation

MDPI

Article

Dynamic Load Balancing Techniques for Particulate Flow Simulations

Christoph Rettinger [1,*] and Ulrich Rüde [1,2]

[1] Chair for System Simulation, Friedrich–Alexander–Universität Erlangen–Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany; ulrich.ruede@fau.de
[2] CERFACS, 42 Avenue Gaspard Coriolis, 31057 Toulouse, France
* Correspondence: christoph.rettinger@fau.de

Received: 30 November 2018; Accepted: 20 January 2019; Published: 23 January 2019

Abstract: Parallel multiphysics simulations often suffer from load imbalances originating from the applied coupling of algorithms with spatially and temporally varying workloads. It is, thus, desirable to minimize these imbalances to reduce the time to solution and to better utilize the available hardware resources. Taking particulate flows as an illustrating example application, we present and evaluate load balancing techniques that tackle this challenging task. This involves a load estimation step in which the currently generated workload is predicted. We describe in detail how such a workload estimator can be developed. In a second step, load distribution strategies like space-filling curves or graph partitioning are applied to dynamically distribute the load among the available processes. To compare and analyze their performance, we employ these techniques to a benchmark scenario and observe a reduction of the load imbalances by almost a factor of four. This results in a decrease of the overall runtime by 14% for space-filling curves.

Keywords: high performance computing; multiphysics simulation; lattice Boltzmann method; rigid particle dynamics; particulate flow; load balancing; parallel computing

1. Introduction

Simulations are becoming an increasingly prominent alternative to often expensive and time consuming laboratory experiments. Therefore, engineers are interested in introducing more and more physical effects to explore complex phenomena with the help of computers. These multiphysics simulations, thus, feature a combination of different algorithms to describe the physical phenomena, like multiphase fluid flow [1], particle motion [2,3], or free surfaces [4]. Typical applications can be found in environmental science [5], additive manufacturing [6], and petroleum engineering [7]. Due to their computational cost, it is often necessary to utilize thread-based and distributed memory parallelization techniques to obtain results in reasonable time and to make use of the immense capabilities provided by today's supercomputers. This is commonly achieved by a spatial domain partitioning where the computational domain is subdivided into smaller pieces which are then distributed among the available processes, see e.g., [8]. As the majority of the parallel numerical codes consist of computation and synchronization, the desired properties of such a distribution are that the overall workload (originating from the computations) is spread evenly among the processes and that the amount of communication (due to synchronization) is minimized. If the first property is violated, the simulation suffers from load imbalances and the processes must wait at the synchronization points for the slowest process, i.e., the one with the largest workload, until the simulation can proceed. As a result, the runtime of the whole simulation increases and the hardware resources at hand are utilized less efficiently due to the idle times. Acquiring and maintaining a distribution with balanced workloads is, thus, crucial to achieve efficient parallel simulations.

Conceptually, the task of load balancing can be split into two steps: At first, referred to as *load estimation*, each subdomain is assigned a weight that quantifies the workload present on that subdomain. Based on these weights, a *load distribution* routine is executed in a second step to reassign the subdomains to the processes. An illustration is given in Figure 1. Depending on the characteristics of the underlying algorithms and the targeted degree of parallelism, various load balancing strategies have been developed, as reviewed in [9]. Meshless methods, like the discrete element method (DEM) [10] or the discrete dipole approximation (DDA) [11], typically use the number of particles [12,13] or collisions [14] per subdomain as weight function. Consequently, this weight may vary significantly in space and time, requiring dynamic load balancing techniques. This can be achieved, e.g., by changing the geometric position of the subdomains [12,13] or further subdivision [14]. Contrary to that, in grid-based algorithms like finite volume or lattice Boltzmann methods, the weight of a subdomain is typically given as the number of grid cells [15–18]. As a special case, if the grid remains unchanged during the simulation and a constant number of cells per subdomain is chosen, a distribution of one subdomain per process is an obvious choice and, thus, encountered in many such programs. However, when adaptive mesh refinement (AMR) is applied to improve the accuracy and efficiency of these simulations [19,20], the workload is altered significantly and load balancing becomes an essential component. Apart from these techniques, load distribution routines have attracted a lot of attention throughout the years and various methods have been proposed, since they are also used for classical graph partitioning problems [9,21,22].

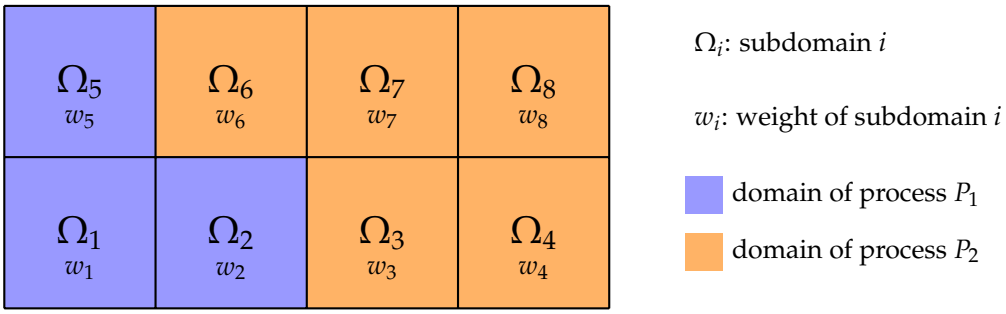


Figure 1. Illustration of domain partitioning with load balancing. The computational domain Ω is subdivided into subdomains Ω_i, such that $\Omega = \bigcup_i \Omega_i$. The load estimator assigns a weight w_i to each subdomain that quantifies the workload. The load distribution routine then assigns the subdomains to the available processes (here P_1 and P_2) such that the load per process is balanced.

It becomes apparent, however, that the applied techniques differ substantially for meshless and grid-based methods where in the first case the subdomains are modified solely for load balancing whereas this is not desired for the grid-based methods as it could affect the accuracy of the simulation. Determining a suitable load balancing approach for multiphysics simulations that incorporate both types of algorithms, thus, poses a challenge. Commonly, in cases where one of the algorithms is the workload-wise dominant one, the specific approaches of this algorithm are used, e.g., the grid-based fluid solver in particulate flow simulations [5,23], which comes at the cost of load imbalances. Alternatively, a different domain partitioning for each part can be used [12], which, however, requires expensive communication and complicated mapping mechanisms between the subdomains for distributed memory simulations. In this paper, we will therefore present a different strategy for dynamic load balancing of these simulations. We will use particulate flows as an illustrating example, which features a coupling between a fluid solver and a particle simulation. Our primary design objective is to enable massively parallel simulations on supercomputers for large physical systems. This restriction already excludes all algorithms that require global knowledge about process local quantities, e.g., the position of all particles, as they will inevitably not scale to several thousand or more processes. Additionally, we require that all parts of our simulation use the same domain partitioning

to avoid the previously mentioned difficulties and bottlenecks. Furthermore, the simulation result should be unaffected by the applied strategy. A variant that complies with these specifications is a static partitioning of the computational domain into blocks of constant size [8], as shown in Figure 1. Load balancing is achieved by dynamically distributing these blocks among the available processes with the goal to have a similar workload on each process by specifically assigning several blocks to each process. We develop a genuine load estimation approach to quantify the workload per block by taking into account all aspects of the coupled simulation and evaluate different load distribution routines in detail.

Ultimately, this results in an efficient simulation approach for large-scale particulate flow problems. It can be used for predictive studies and model development by exploring vast parameter spaces that would not be accessible with laboratory experiments. Such problems can be found in a wide range of applications, e.g., sediment transport in riverbeds [24,25], flow of proppant-laden fracturing fluid in petroleum engineering [7,26], swarms of microswimmers in biological flows [27], and fluidized bed reactors in chemical engineering [28,29].

The remainder of this paper is structured as follows: At first, we briefly describe the numerical methods we apply for geometrically fully resolved particulate flow simulations in Section 2, consisting of the lattice Boltzmann method, a hard contact particle solver and the fluid–particle coupling mechanism. Next, in Section 3, we present and calibrate our load estimation strategy that predicts the block's weight based on locally available quantities. This estimator is then applied in Section 4, where additionally the performance of several load distribution approaches is investigated and compared. We summarize our findings in Section 5.

2. Numerical Methods

In this section, we briefly describe the numerical methods that we apply in our simulations of particulate flows. The fluid flow is computed via the lattice Boltzmann method and a non-smooth granular dynamics simulation accounts for the motion and collisions of the suspended particles. A coupling of these two approaches is established based on the momentum exchange principle which features geometrically fully resolved particle shapes [2,5]. This coupling is illustrated in Figure 2. We note, however, that most of the observations and results obtained in the following sections also carry over to other CFD methods, like finite volume methods, other particle collision resolution methods, like the discrete element method [10], and also other coupling algorithms, like the immersed boundary method or the combined finite-discrete element method [30].

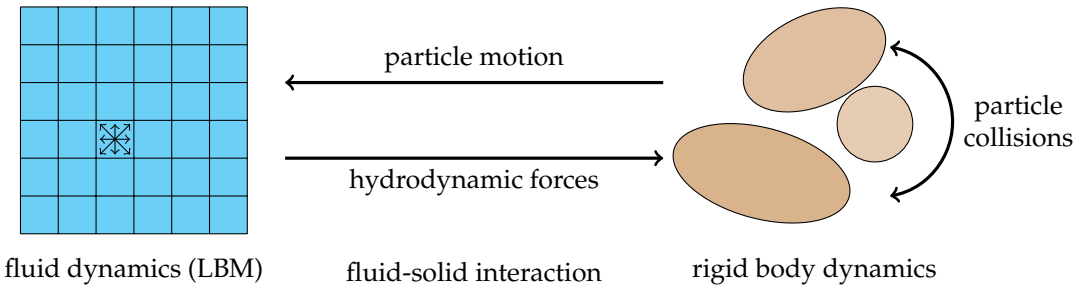


Figure 2. Schematic representation of the numerical methods presented in Section 2 that are used for particulate flow simulations in this work.

2.1. Lattice Boltzmann Method

In the lattice Boltzmann method (LBM) [31,32], the three-dimensional domain is discretized with a uniform lattice. Each cell features q different particle distribution functions (PDF) f_q where each one is associated with a lattice velocity $\boldsymbol{c}_q$. We employ the *D3Q19* variant such that $q \in \{0...18\}$.

The macroscopic quantities, density $\rho_f = \rho_0 + \delta\rho_f$, with a mean density ρ_0 and a fluctuation $\delta\rho_f$, and fluid velocity $\boldsymbol{u}_f$, are obtained as moments of the PDFs in each grid cell $\boldsymbol{x}$:

$$\delta\rho_f(\boldsymbol{x},t) = \sum_q f_q(\boldsymbol{x},t),\ \boldsymbol{u}_f(\boldsymbol{x},t) = \frac{1}{\rho_0}\sum_q \boldsymbol{c}_q f_q(\boldsymbol{x},t) \tag{1}$$

The evolution of the PDFs is described by alternating between a collision and a stream step. We use the two-relaxation-time (TRT) collision operator [33] that results in the collision step

$$\tilde{f}_q(\boldsymbol{x},t) = f_q(\boldsymbol{x},t) - \frac{1}{\tau_+}\left(f_q^+(\boldsymbol{x},t) - f_q^{\mathrm{eq},+}(\rho_f,\boldsymbol{u}_f)\right) - \frac{1}{\tau_-}\left(f_q^-(\boldsymbol{x},t) - f_q^{\mathrm{eq},-}(\rho_f,\boldsymbol{u}_f)\right), \tag{2}$$

where the PDFs are split into their symmetric f_q^+ and anti-symmetric f_q^- parts with their respective collision times τ_+ and τ_-. This collision step relaxes the PDFs linearly towards their equilibrium values

$$f_q^{\mathrm{eq}}(\rho_f,\boldsymbol{u}) = w_q\left(\rho_f + \rho_0\left(3\boldsymbol{c}_q\cdot\boldsymbol{u}_f + \tfrac{9}{2}(\boldsymbol{c}_q\cdot\boldsymbol{u}_f)^2 - \tfrac{3}{2}\boldsymbol{u}_f\cdot\boldsymbol{u}_f\right)\right), \tag{3}$$

where the lattice weights w_q are taken from [34]. The succeeding streaming step is then given as

$$f_q(\boldsymbol{x}+\boldsymbol{c}_q\Delta t, t+\Delta t) = \tilde{f}_q(\boldsymbol{x},t). \tag{4}$$

The kinematic fluid viscosity ν is determined by the collision time and we use the following relations to define τ_+ and τ_- [33]:

$$\nu = \frac{1}{3}\left(\tau_+ - \frac{\Delta t}{2}\right),\ \left(\frac{1}{2} - \frac{\tau_+}{\Delta t}\right)\left(\frac{1}{2} - \frac{\tau_-}{\Delta t}\right) = \frac{3}{16}. \tag{5}$$

2.2. Rigid Body Dynamics

The particles suspended in the fluid interact with each other and walls via collisions. In order to account for these particle–particle or particle–wall collisions, we apply a rigid body solver which determines the collision forces and integrates the particles' position and velocity in time. The individual particles are, thus, represented by their exact geometric shape. Each particle is described by its position $\boldsymbol{X}_p$, orientation $\boldsymbol{Q}_p$, as well as its translational and rotational velocity, $\boldsymbol{V}_p$ and $\boldsymbol{W}_p$. The dynamics are described by the Newton-Euler equations for rigid bodies, given acting forces $\boldsymbol{F}_p$ and torques $\boldsymbol{T}_p$. These arise from external forces hydrodynamic interactions with the surrounding fluid and collisions. Here, the collisions are determined by a contact detection algorithm and are modeled as inelastic hard contacts [35], resulting in non-smooth functions for position and orientation. This effectively resolves overlaps between the particles. Additionally, the contact model includes Coulomb friction. The integration of the particle trajectories in time is carried out by a semi-implicit Euler method. As a result, a non-linear system of equations has to be solved for each particle and each contact [35]. More details about the algorithm and its implementation can be found in [35,36].

2.3. Fluid-Solid Interaction

In order to realize a hydrodynamic coupling between the fluid and the geometrically fully resolved particulate phase, we make use of the momentum exchange method [37]. Here, the particles are explicitly mapped into the fluid domain, marking cells contained inside the particle as *solid*, in contrast to the *fluid* cells.

A no-slip boundary condition is applied along the surface of the particles which is given by the central linear (CLI) scheme as [33]

$$f_{\bar{q}}(\boldsymbol{x},t+\Delta t) = \tilde{f}_q(\boldsymbol{x},t) + \frac{1-2\delta_q}{1+2\delta_q}\left(\tilde{f}_q(\boldsymbol{x}-\boldsymbol{c}_q\Delta t,t) - \tilde{f}_{\bar{q}}(\boldsymbol{x},t)\right) - \frac{12\,w_q\rho_0}{1+2\delta_q}\boldsymbol{v}_b\cdot\boldsymbol{c}_q, \tag{6}$$

where the boundary velocity $\boldsymbol{v}_b = \boldsymbol{V}_p + \boldsymbol{W}_p \times (\boldsymbol{x}_b - \boldsymbol{X}_p)$ is the particle's velocity at the position $\boldsymbol{x}_b$. The interpolation weight δ_q is the distance between the cell center $\boldsymbol{x}$ and the actual surface position $\boldsymbol{x}_b = \boldsymbol{x} + \delta_q \boldsymbol{c}_q$, normalized by the distance of the cell centers, i.e., $\|\boldsymbol{c}_q \Delta t\|$. For spherical particles, it can be determined analytically as the result of a ray-sphere intersection problem. A sketch of the boundary-handling procedure is provided in Figure 3.

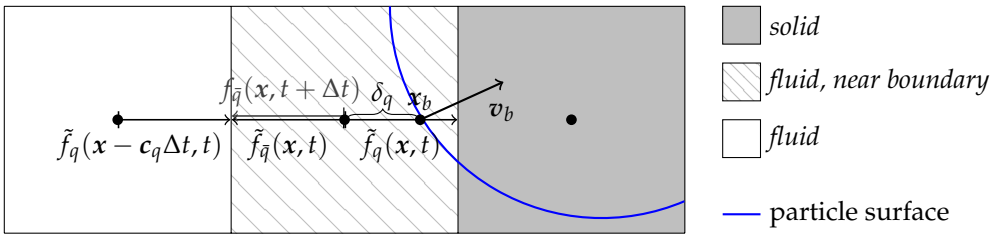


Figure 3. Sketch of the particle-mapping and the boundary treatment according to the CLI boundary scheme from Equation (6).

As a result of this boundary treatment, momentum is transferred from the fluid onto the submerged particle along the boundary link q, given as [38]:

$$\boldsymbol{F}_q(\boldsymbol{x}_b, t) = (\boldsymbol{c}_q - \boldsymbol{v}_b)\tilde{f}_q(\boldsymbol{x}, t) - (\boldsymbol{c}_{\bar{q}} - \boldsymbol{v}_b) f_{\bar{q}}(\boldsymbol{x}, t + \Delta t). \tag{7}$$

By summing up these contributions over all boundary links from all *near-boundary* cells (NB) next to a single particle, the total hydrodynamic force and torque are determined as

$$\boldsymbol{F}_p^{\text{hyd}} = \sum_{NB} \sum_{q} \boldsymbol{F}_q(\boldsymbol{x}_b)\,, \tag{8}$$

$$\boldsymbol{T}_p^{\text{hyd}} = \sum_{NB} \sum_{q} (\boldsymbol{x}_b - \boldsymbol{X}_p) \times \boldsymbol{F}_q(\boldsymbol{x}_b), \tag{9}$$

respectively.

Finally, due to the explicit mapping of the particles into the domain, cells that convert from *solid* to *fluid* due to moving particles need to restore PDF information before the simulation can continue. Here, we initialize the PDFs of such a transformed cell with its equilibrium values, Equation (3), using the particle's surface velocity and a spatially averaged density [2].

2.4. Parallelization

The direct numerical simulations of complex particulate flow scenarios can exhibit enormous computational costs that can only be tackled by parallel execution on a large number of processes. Such large computers are designed with a distributed memory so that typically the message passing interface (MPI) is used to implement the parallel algorithms. We here employ the open-source WALBERLA framework [8,39] where all the above-mentioned algorithms are implemented. It partitions the computational domain into *blocks*. They contain all the data for this subdomain and are then assigned to the available MPI processes to distribute the workload, see Figure 1. This requires the introduction of synchronization mechanisms for the fluid as well as for the particle simulation which both use the same domain partitioning as a result of the close coupling within the same framework.

The sketch in Figure 4 shows the applied techniques for two neighboring blocks. Each block is subdivided into uniform cells that are required for the fluid simulation by the lattice Boltzmann method, Section 2.1. To be able to carry out the streaming step, Equation (4), across block borders, an extra layer of cells, a so called *ghost layer*, has to be introduced around each block. They contain the corresponding PDFs from the neighboring blocks that are to be streamed to cells inside the block at hand and, thus, have to be communicated beforehand. Similarly, the particle simulation defines *local*

and *shadow* particles on a block [40]. The former are particles whose center of mass are contained inside this block. To check for particle–particle collisions, however, all possible collision partners have to be known on a block. This necessitates that copies of all particles that intersect with the block, with the center of mass being contained inside another block, are present on that block as well. These copies are here called *shadow* particles, where *ghost* or *proxy* are also commonly used labels.

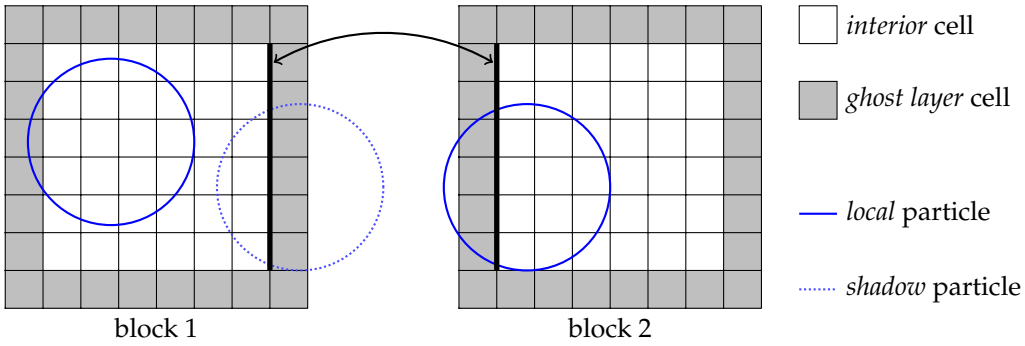


Figure 4. Domain partitioning into blocks. The cell-based fluid simulation requires a ghost layer for synchronization. The particle simulation uses shadow particles. The two bold vertical lines denote the common face of both blocks.

More details on the block-based domain partitioning and the communication routines can be found in [5,8,17,20].

2.5. Complete Algorithm

By combining the different components presented in the previous sections, we obtain the complete algorithm for coupled fluid–particle simulations with geometrically fully resolved particles, shown in Algorithm 1. Since the treatment of the boundary, Equation (6), and the link-based computation of the interaction force, Equation (7), require the same information, we fuse them to a single step in line 6 of the algorithm. Additionally, we note that the time step of the rigid body has to be chosen smaller than the one of the fluid simulation to obtain an accurate collision behavior [3]. Therefore, we introduce a sub cycling loop for the rigid body solver in which we subdivide the time step into S sub steps in which the hydrodynamic force acting on the particles is kept constant, see line 8 of Algorithm 1.

Algorithm 1: Coupled algorithm for particulate flow simulations.

```
1  for each time step t do
2      (Re-)map particle into fluid domain.
3      Reconstruct PDFs of converted cells if necessary.
4      Perform LBM collision step, Equation (2).
5      Communicate PDF ghost layers.
6      Apply boundary conditions, Equation (6) and compute hydrodynamic forces on particles,
         Equation (7).
7      Perform LBM stream step, Equation (4).
8      for each rigid body solver subcycle S do
9          Perform rigid body solver step (collision detection and resolution, force
             synchronization, time integration, position and velocity synchronization), Section 2.2.
10     end
11 end
```

3. Development and Calibration of a Workload Estimator

3.1. Description

The algorithm outlined in Algorithm 1 is a typical representative for a simulation procedure to solve multiphysics problems. One time step consists of contributions from the individual physical components, here fluid flow, Section 2.1, and rigid body dynamics, Section 2.2, as well as coupling algorithms for their interaction. Depending on the physical setup at hand, each of these parts generates a different workload within a block, used for the partitioning of the simulation domain, see Section 2.4. Taking particulate flows as an example, blocks with dense particle packings result in longer compute times, i.e., larger workloads, for the rigid body simulation in comparison to blocks without any particles. Additionally, in many physical problems, the location of these areas, and consequently the workload per block, changes over time. Since the simulation gets synchronized via communications in each time step, unbalanced workload distributions result in waiting times for the processes that own blocks with smaller workloads. These idle times reduce the overall efficiency of the simulation and result in a longer time to solution than ideally achievable. For successful load balancing, it is, thus, important to first estimate the workload per block. This is resembled by a scalar quantity and we denote it as *weight* of the block, see Figure 1. Once the weights of all blocks are known, a load distribution algorithm can then be used to reassign the blocks to the processes to establish a balanced state.

In this section, we present our approach to develop a load estimator for particulate flow simulations with Algorithm 1. The goal is to find a function that is able to predict the workload of a block, depending on local quantities that describe the state of this block. The workload is here given as the runtime of a single time step in this simulation, excluding communication routines. For that reason, simulations of a characteristic setup have to be carried out. During these simulations, the block local quantities and the corresponding runtimes of the different parts of the algorithm are continuously evaluated and stored. Based on this data, function fitting is applied to finally obtain a estimator function that can be used in all subsequent simulations to predict the workload for a block.

The simulation setup hast to be chosen such that different cases which can be encountered in a real application also occur in the simulation. This means that blocks with no particles and dilute as well as densely packed particles should be included in the setup and, thus, in the data. Only then, the function that is to be fitted to these measurements can later on predict these cases accurately enough. Specifically, we execute several simulations to obtain separate time measurements of the major parts of Algorithm 1 and vary some of the important parameters to increase the amount of data and the reliability. The block local quantities that are available in our fluid–particle simulation are given in Table 1 and their definition can be obtained from Figures 3 and 4. For function fitting, we first have to determine the form of the functions that map these quantities to the measured runtime. For each part, this is achieved by analyzing the structure of the individual algorithm. In a calibration step, those functions are then fitted to the timing measurements of the separate parts based on the block local quantities to determine the functions' coefficients. Finally, those functions are combined to obtain a complete estimator for the workload per block.

Table 1. Quantities that describe the state of a block.

Variable	Description
C	total number of cells on a block
F	number of cells flagged as *fluid*
B	number of cells flagged as *near boundary*
P_L	number of local particles
P_S	number of shadow particles
K	number of contacts between rigid bodies
S	number of sub cycles of the rigid body simulation

We note that this procedure must be carried out only once as a preprocessing step for all upcoming simulations. Once the load estimator is found, it is simply applied, i.e., the fitted function is evaluated, in the simulations whenever load balancing is carried out. This procedure also only involves simulations of small systems which keeps the computational effort small.

3.2. Workload Contributions

In the following, we establish the form of the functions we use to fit the measured times based on the local quantities from Table 1. The contributions are mainly gathered from the structure of the respective implementation, which are briefly outlined for each part.

3.2.1. LBM Module

The simulation of the fluid flow is carried out by the LBM, which consists of the collision (line 4 of Algorithm 1) and the stream (line 7) step. These are only carried out for cells marked as *fluid* and the pseudocode is given in Algorithms 2 and 3, respectively. The resulting workload heavily depends on the applied collision operator and the actual implementation [41,42] but it will generally be mainly determined by the number of cells C and the number of fluid cells F. Thus, a function that represents the workload generated by the two LBM steps is given by

$$\mathrm{WL}_{\mathrm{LBM}} = a_{1,\mathrm{LBM}} C + a_{2,\mathrm{LBM}} F + a_{3,\mathrm{LBM}} \tag{10}$$

Algorithm 2: Pseudocode for LBM (collision).

```
1 for each cell c do
2     if c is a fluid cell then
3         Perform LBM collision step.
4     end
5 end
```

Algorithm 3: Pseudocode for LBM (stream).

```
1 for each cell c do
2     if c is a fluid cell then
3         Perform LBM stream step.
4     end
5 end
```

3.2.2. Boundary-Handling Module

The boundary-handling procedure for LBM, line 6 of Algorithm 1, applies the chosen condition for each near boundary cell. More specifically, it does so for each fluid-boundary link of the near boundary cell As can be seen in the pseudocode of Algorithm 4, the resulting workload will, thus, be related to the number of these links. However, evaluating this number for each block is computationally more costly than simply counting the number of near boundary cells and the workload per link depends on the specific boundary condition, which we do not distinguish here. We therefore simply use a function of the form

$$\mathrm{WL}_{\mathrm{BH}} = a_{1,\mathrm{BH}} C + a_{2,\mathrm{BH}} NB + a_{3,\mathrm{BH}}, \tag{11}$$

where we additionally included C to represent the outer loop.

Algorithm 4: Pseudocode for boundary-handling.

```
for each cell c do
    if c is a near boundary cell then
        for each neighboring boundary cell do
            Perform boundary-handling according to applied boundary condition.
        end
    end
end
```

3.2.3. Particle-Mapping Module

Mapping the particles into the domain by marking the contained cells as boundary cells is an important part of the applied fluid–particle coupling algorithm, see line 2 of Algorithm 1. The pseudocode is outlined in Algorithm 5. The resulting workload, thus, depends on the number of particles, local and shadow, as well as the extent of their axis-aligned bounding box. Since the size of this box depends on the diameter of the particle, this information is not readily available. Instead, we can analyze the result of the mapping by including the number of non-fluid cells as a kind of block local solid volume fraction. Since shadow particles are only partially present on the block they will generate a smaller workload than local particle and we should, thus, distinguish workload contributions originating from local and shadow particles in our function. We therefore propose the following function:

$$\mathrm{WL}_{\mathrm{Coup1}} = a_{1,\mathrm{Coup1}} C + a_{2,\mathrm{Coup1}} F + a_{3,\mathrm{Coup1}} P_L + a_{4,\mathrm{Coup1}} P_S + a_{5,\mathrm{Coup1}} \tag{12}$$

Algorithm 5: Pseudocode for particle-mapping (Coup1).

```
for each particle p do
    Obtain an axis-aligned bounding box (AABB) which fully contains the particle.
    for each cell c in this AABB do
        if cell c is contained in particle p then
            Set flag to boundary (solid) flag.
        end
    end
end
```

3.2.4. PDF Reconstruction Module

The second part of the coupling algorithm, line 3 in Algorithm 1, reconstructs the PDF values in cells that have changed from being solid to being fluid due to the motion of the corresponding particle, see Algorithm 6. The generated workload, thus, depends on the position, orientation and velocity of the individual particles and is therefore difficult to predict in general. Assuming that chances for these cell state changes are higher if there are more particles around and the solid volume fraction is larger, we estimate its workload by:

$$\mathrm{WL}_{\mathrm{Coup2}} = a_{1,\mathrm{Coup2}} C + a_{2,\mathrm{Coup2}} F + a_{3,\mathrm{Coup2}} P_L + a_{4,\mathrm{Coup2}} P_S + a_{5,\mathrm{Coup2}} \tag{13}$$

Algorithm 6: Pseudocode for PDF reconstruction (Coup2).

```
for each cell c do
    if c has switched from solid to fluid cell then
        Reconstruct PDFs in c.
    end
end
for each cell c do
    if c has switched from solid to fluid cell then
        Set fluid flag in c.
    end
end
```

3.2.5. Rigid Body Simulation Module

The rigid body simulation part, line 9 of Algorithm 1, consists of several components which for simplicity will all be included in a single function. Algorithm 7 outlines these different sub steps [35]. The first part, contact detection, is typically of squared complexity since all particles have to be checked against all other particles to find possible contacts. Our implementation makes use of hash grids for optimizing this routine to obtain linear complexity. This, however, is only activated if a reasonable number of particles is present on a block since otherwise the computational overhead renders it less efficient. In a second step, the contact resolution treats each contact according to the applied collision model in order to resolve the overlaps between particles. Determining the needed information for these two steps is simple for spherical particles but can become complex and costly for other shapes. The last step uses a time integrator scheme to update the local particles' position and velocity. In the simulation algorithm, the whole rigid body algorithm is embedded into a loop over S number of sub cycles, see line 8 of Algorithm 1. Combining these steps, we use the following function to estimate the workload:

$$\mathrm{WL}_{\mathrm{RB}} = S\left(a_{1,\mathrm{RB}}(P_L + P_S)^2 + a_{2,\mathrm{RB}}P_L + a_{3,\mathrm{RB}}P_S + a_{4,\mathrm{RB}}K + a_{5,\mathrm{RB}}\right) \tag{14}$$

Algorithm 7: Pseudocode for rigid body simulation.

```
for each particle p do
    if p is in contact with another particle then
        Generate a contact pair.
    end
end
for each contact pair k do
    Compute reaction to resolve contact.
end
for each particle p do
    Integrate p in time using the acting forces.
end
```

3.2.6. Total Workload Estimator

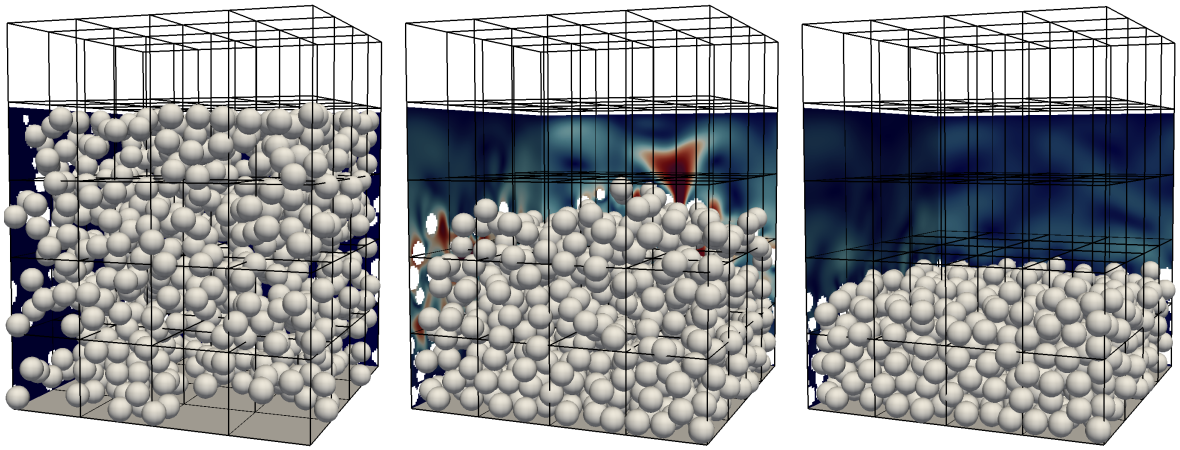
Since, in the general case, only a single weight per block has to be provided, the aforementioned contributions must be combined in order to obtain a estimator function for the total workload

per block. A natural and simple choice is to add up all individual functions, Equations (10)–(14), effectively combing the coefficients of the block quantities:

$$\mathrm{WL}_{\mathrm{tot}} = \mathrm{WL}_{\mathrm{LBM}} + \mathrm{WL}_{\mathrm{BH}} + \mathrm{WL}_{\mathrm{Coup1}} + \mathrm{WL}_{\mathrm{Coup2}} + \mathrm{WL}_{\mathrm{RB}}. \tag{15}$$

3.3. Simulation Setup

After having identified and defined the workload functions for the different modules of the algorithm, we now carry out multiple simulations of a characteristic setup to obtain timing information that are then used to fit the functions and determine the coefficients. To obtain a generally applicable estimator, the setup should be designed such that it contains various cases that are also encountered in typical applications involving particulate flows. Those are e.g., bounding walls, densely packed areas with many inter-particle collisions as well as dilute regions with only few or even no particles. Additionally, due to complex bounding geometries, some parts of the computational domain could be completely excluded from the simulation as neither fluid nor particles can enter these regions. We therefore chose a horizontally periodic setup with an initially random distribution of particles. Those particles are then set into a settling motion and then pack at the bottom plane until all particles have settled. Visualizations of the initial, intermediate and final state of the simulations with spherical particles can be seen in Figure 5.



(**a**) (**b**) (**c**)

Figure 5. Workload evaluation simulation with $b_s/\Delta x = 32$ and $D/\Delta x = 10$. Black boxes show the $4 \times 4 \times 5$ blocks used for domain partitioning. The magnitude of the fluid velocity is shown in two slices at the back of the domain. (**a**) Initially random particle distribution ($t^* = 0$). (**b**) Ongoing settling ($t^* = 1.3$). (**c**) Final, completely settled state ($t^* = 2.5$).

The characteristic parameters of this setup are the Galileo number, $Ga = u_s D/\nu = 30$, with a characteristic settling velocity $u_s = \sqrt{(\rho_p/\rho_f - 1)gD}$, the diameter D and kinematic viscosity ν. The particles have a density ratio of $\rho_p/\rho_f = 2.5$ and are subjected to acceleration g in the vertical direction, resulting in buoyancy forces. The domain size is $L_x \times L_y \times L_z$ with $L_x = 4b_s$, $L_y = 4b_s$ and $L_z = 5b_s - o_t$. Here, b_s is the block size and we introduce a constant offset of the top wall $o_t = 1.05b_s$. The global solid volume fraction of the domain is 0.2. The simulation is run for 2.5 unitless time steps, where one time step is $T = L_z/u_s$. As the definition of the domain size already suggests, we subdivide the whole domain into $4 \times 4 \times 5$ blocks of size $b_s \times b_s \times b_s$.

As a result, we obtain five vertical layers of blocks with different characteristics throughout the simulation. The bottom layer experiences a continuous increase in the number of particles until it is densely packed. The second layer features the interface between the particle packing and the bulk fluid

region at the end of the simulation. Blocks of the third layer are traversed by all upper particles during the settling phase and end up without any particles or boundaries. Similarly, the fourth layer ends up with no particles but, due to the offset in the top wall, with a constant number of boundary and near-boundary cells. The blocks of the topmost row contain neither fluid cells nor particles throughout the complete simulation since they are completely overlapped by the top wall.

Those simulations are carried out for different typical block sizes $b_s/\Delta x = \{24, 32, 48, 64\}$ and diameters $D/\Delta x = \{10, 20\}$ to obtain a large enough variance in the samples. Each simulation is executed in parallel on 80 processes of the SuperMUC Haswell cluster at the LRZ such that each process is assigned one block. This allows to obtain separate measurements for each block with 2000 samples each. In total, this results in around 1.3×10^6 data points that will be used for the function fitting. In the first 50 time steps, no measurements are taken to exclude possible warm-up effects of the hardware. Furthermore, we make use of thread pinning provided by the LIKWID tool [43,44] to obtain reliable measurements. Each sample consists of the current values for the block-local quantities from Table 1, where $S = 10$ is kept in all simulations, and additionally timing measurements of the five algorithm parts from Section 3.2, i.e., m_{LBM}, m_{BH}, m_{Coup1}, m_{Coup2} and m_{RB}.

3.4. Results of the Calibration

An example of the quantity evaluation and time measurement for the LBM module of the aforementioned simulation setup is shown in Figure 6. From the temporal evolution of F, the curves can be matched to a block of one of the five rows. Clearly, a correlation between F and m_{LBM} can be seen which is in agreement with the assumption made in Equation (10).

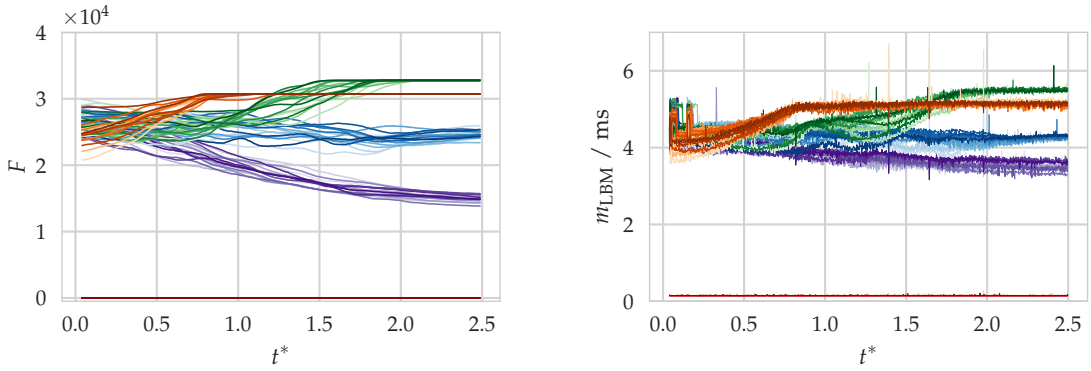


Figure 6. Number of fluid cells (**left**) and time measurements (**right**) of the LBM part per block over time ($t^* = t/T$) of the workload evaluation simulation with $b_s/\Delta x = 32$ and $D/\Delta x = 10$. The coloring resembles the affiliation of the block to one of the five rows in the setup.

Before we evaluate the result of the calibration process of the workload estimator, we want to obtain insight into the proportion of the different parts of the total runtime, given by

$$m_{\mathrm{tot}} = m_{\mathrm{LBM}} + m_{\mathrm{BH}} + m_{\mathrm{Coup1}} + m_{\mathrm{Coup2}} + m_{\mathrm{RB}}. \tag{16}$$

These proportions are different for each sample and are, thus, visualized in a compact format as a box-and-whiskers plot in Figure 7, showing the median, the upper, and lower quartile values as a box, as well as whiskers that extend to the 5th and 95th percentiles. It can be seen that the LBM and the BH part make up most of the workload with an average of 45% and 35%, respectively. This is followed by the rigid body simulation and the coupling parts with median values below 5%. The relatively large spread in the whiskers is introduced by the empty blocks of the fifth block row in the setup where total runtimes are very low and the different parts, except for Coup1, contribute similarly to it.

We can conclude that for spherical particles with the chosen coupling algorithm, it is most important to accurately predict the workload from the lattice Boltzmann and the boundary-handling routines.

With all samples available, we make use of the curve fitting functionality provided by the Python library SciPy in a post-processing step to obtain the coefficients from Equations (10)–(14). The outcome is presented in Table 2.

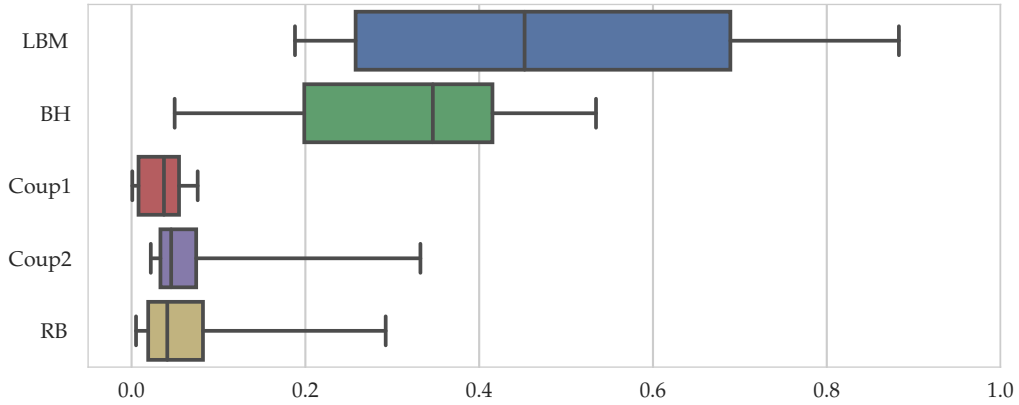


Figure 7. Box-and-whiskers plot of the proportion of the part's runtime of total runtime, m_X/m_{tot}, for each sample.

Table 2. Fitted values for the coefficients of Equations (10)–(14).

Coefficient	Value
$a_{1,LBM}$	9.99×10^{-6}
$a_{2,LBM}$	1.57×10^{-4}
$a_{3,LBM}$	-8.23×10^{-2}
$a_{1,BH}$	6.65×10^{-6}
$a_{2,BH}$	7.06×10^{-4}
$a_{3,BH}$	-1.09×10^{-1}
$a_{1,Coup1}$	3.08×10^{-6}
$a_{2,Coup1}$	2.42×10^{-7}
$a_{3,Coup1}$	1.41×10^{-2}
$a_{4,Coup1}$	2.78×10^{-2}
$a_{5,Coup1}$	-1.40×10^{-1}
$a_{1,Coup2}$	5.99×10^{-6}
$a_{2,Coup2}$	3.90×10^{-6}
$a_{3,Coup2}$	-8.80×10^{-3}
$a_{4,Coup2}$	2.51×10^{-2}
$a_{5,Coup2}$	-1.30×10^{-1}
$a_{1,RB}$	1.16×10^{-6}
$a_{2,RB}$	9.62×10^{-4}
$a_{3,RB}$	2.75×10^{-4}
$a_{4,RB}$	1.48×10^{-3}
$a_{5,RB}$	1.88×10^{-2}

To quantify the quality of the fit for each part, various measures can potentially be evaluated. One of them is the relative error of predicted workload by the five different fits for each individual sample, calculated as $E_X = (WL_X - m_X)/m_{tot}$. It is visualized as histograms for the different parts and the total workload in Figure 8. Additionally, the median and the median absolute deviation (MAD), which is a robust measure of the statistical dispersion, are stated. Note that we calculate the relative

error with respect to the total runtime instead of m_X to avoid the division by almost zero for the small runtimes of Coup1 and RB, see Figure 7.

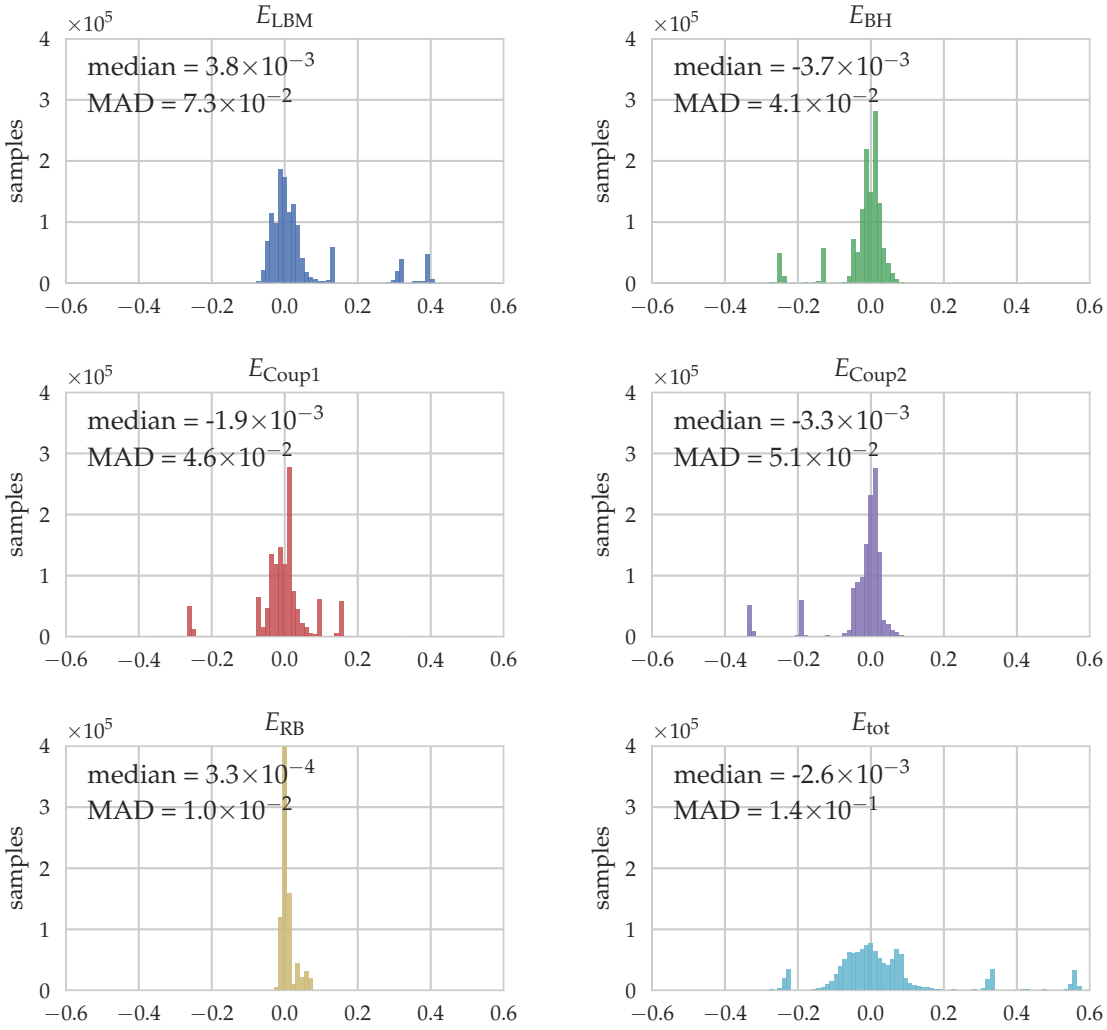


Figure 8. Relative errors E_X of predicted workload for the five different parts and the total runtime.

For the LBM part, we obtain good predictions of the workload with a median close to zero. Outliers can be seen at some distinct error values which originate from the empty blocks where the prediction is generally worse. A similar conclusion can be drawn for the BH part, where again the median is close to zero and some outliers can be found, this time on the negative side, i.e., due to underpredictions of the workload. These samples can again be attributed to those with empty blocks but also the ones of the third row which eventually end up with no boundary cells like the empty blocks. The Coup1 part shows an acceptable median value but a relatively large MAD value due to the outliers at around 30%. These are again the empty blocks where the assumption we have formulated when establishing our fit function in Equation (12), i.e., the mapping should be related to the number of boundary cells as a measure of the solid volume fraction, breaks down as no particles are present. Similarly, for Coup2 the median is close to zero but the MAD value is relatively large. As discussed when setting up the equation for the fit, Equation (13), predicting the workload for this part is especially difficult as it can not be directly related to any of the available quantities. This is then also the source of deviations from the predictions. At last, the RB part is well-predicted in average and also the variance is small. Considering the relative error for the total runtime, shown in the last

histogram, the obtained estimator from Equation (15) is able to predict the observed runtimes with good accuracy. Outliers originating from the empty blocks of different block sizes where the fits do not work as good can be seen. We also find that over 85% of all samples are predicted with a relative error of less than 10%.

3.5. Discussion

In this section, we have constructed a specific workload estimator for coupled fluid–particle simulations with the methods from Section 2. Here, the derivation of the specific form of the fit functions, Equations (10)–(14), was purely driven by the algorithms themselves. These functions can, thus, only be valid assumptions for the here applied algorithms. As they calibration process of the coefficients relies on data from actual runtime measurements, the obtained coefficients for these functions also depend on the hardware on which the corresponding measurements were obtained. However, it can be expected that they still remain good predictors if the hardware design is similar to the used one. In case of larger differences, like e.g., using a GPU instead of a CPU to run the simulation [45,46], this might no longer be the case and the measurements and fits should be redone. A possible improvement to overcome this drawback would be to also add hardware details, like cache sizes, clock frequency, etc., to the estimator and try to come up with a performance model for hardware-aware predictions. Since this is a topic of active research, even for simple, stencil-based algorithms like LBM [42,47], this is not further pursued here.

Generally, the here reported values for the coefficients from Table 2 should not be blindly reused in other programs even if the same algorithms and hardware are used, as also the actual implementation of the algorithms can have a large influence on the runtimes. Nevertheless, by following the procedure presented in this section as a general recipe, workload estimators for many different applications, including pure CFD or complicated multiphysics setups, can be obtained. They are then tailored and calibrated for the specific algorithms, hardware and implementation, and can be used in upcoming simulations of that kind to predict, and in a next step also distribute, the workload.

Besides the form of the fit functions, the simulation setup to generate the measurements has to be selected with care. As reported in Section 3.4, in our case the predictions of the empty blocks were generally worse than for other, regular blocks as some assumptions broke down. We still included them in the calibration procedure in order to obtain a general workload estimator and to avoid any special treatment of these empty blocks in our load estimation step. However, if empty blocks do not appear in the simulations in which this workload estimator will then be applied, we suggest to remove them from the test scenario as well. This improves the quality of the fits notably and a more accurate estimator can be obtained.

Future work will aim at further improvements of the load estimator. By including information available from performance models for the different algorithms, the workload estimator can be made more general and flexible. Tools like Kerncraft [48] automatically analyze the performance of a given implementation for the hardware at hand, which would render the estimator independent of these factors. Furthermore, a workload estimate based on the current runtimes is a natural alternative to the proposed predictor as it is able to use actual data from the currently running simulation. Such an estimator can not be used in an a-priori fashion which renders it less useful for adaptive grid refinement where new blocks are created and have to be distributed immediately to avoid huge imbalances. However, a combination of both strategies would result in a kind of predictor-corrector approach which could be a superior strategy.

4. Comparison of Load Distribution Methods

4.1. Description

Once the calibration process of the workload estimator has been executed, Equation (15) with coefficients from Table 2, it can be applied in the subsequent simulations to assign a weight to each

block, representing its workload. As a second step, these weights are now used by load distribution methods to reassign the blocks to the processes dynamically throughout the simulation. Besides the reduction of load imbalances, minimizing communication costs between the different processes and keeping the redistribution costs low are the key aspects of this step. We briefly review the different types of distribution methods in the next section. Next, we set up a simulation that is different from the one used to deduce the load estimator in Section 3.3. With this setup, we demonstrate the capabilities of our workload estimation approach and can compare the performance of different load distribution strategies.

4.2. Load Distribution Algorithms

4.2.1. Space-Filling Curve

Generally, a space-filling curve has the property that it covers the entire n-dimensional unit hypercube. Typically, these curves are constructed recursively from a sequence of piecewise linear continuous curves, following a specific construction pattern. Different versions of such curves exist, e.g., the Peano, Hilbert, or Morton curve, as reviewed in [21]. As a practical result, a curve is obtained that connects all blocks of the computational domain and, thus, determines a one-dimensional ordering of these blocks. For load distribution and while knowing the block weights as well as the total weight of all blocks, the blocks are assigned to the processes in a greedy manner. This means that one traverses the curve and picks the blocks until the sum of these block weights is approximately equal to the total weight divided by the number of processes. Due to the construction of these curves, the assigned blocks are usually geometrical neighbors and, thus, a reduction of the inter-process communication is implicitly achieved. More information about space-filling curves can be found in [21].

4.2.2. Diffusive Distribution

Load distribution via space-filling curves requires global information about the blocks and their weights. For massively parallel simulations, this procedure poses an upper limit on the applicable number of processes as it inherently does not scale and also might require too much memory on a single process [14,20]. For these cases, diffusive techniques become the only feasible approach to distribute the load. There, processes require only information about the load distribution in their direct neighborhood and will then try to even out possible imbalances by sending individual blocks to these neighboring processes. The load gets distributed in a diffusive manner and, thus, several iterations are required to obtain good results. However, a well-balanced load cannot be guaranteed and also the blocks on a process might get fragmented over time which increases communication costs.

4.2.3. Graph Partitioning

The primary goal of these approaches is to partition unstructured graphs such that the edge cut is minimized, corresponding to a reduced communication between the processes. This requires to determine weights for the edges that resemble the communication cost between neighboring blocks. As an additional optimization constraint, weights for the vertices, i.e., here the blocks, can be provided in order to balance these as well. Since graph partitioning is a common problem in various application fields, several algorithms are available [9,22,49]. The MPI-parallel library ParMETIS [50] is a commonly applied choice for this multi-objective optimization task. It offers various algorithms to construct, improve or update graph partitions that can be further tuned by specifying additional parameters. Furthermore, if provides functionalities to deal with multi constraint problems where several weights per block can be given to account for multiphysics applications [51,52]. Generally, it makes use of combinations of space-filling curves, multilevel algorithms and diffusive distributions to obtain a graph partitioning.

Specifically, the required edge weights EW between two vertices v_1 and v_2, i.e., here blocks with b_s^3 cells, are evaluated as follows to resemble the communication volume for the LBM part:

$$\mathrm{EW}(v_1, v_2) = \begin{cases} b_s^2, \text{ if } v_1 \text{ and } v_2 \text{ have common face,} \\ b_s, \text{ if } v_1 \text{ and } v_2 \text{ have common edge,} \\ 1, \text{ if } v_1 \text{ and } v_2 \text{ have common corner,} \\ 0, \text{ else.} \end{cases} \tag{17}$$

ParMETIS offers various algorithms and tuning parameters. We restrict ourselves to the algorithms `PartGeomKway` and `AdaptiveRepart`. The latter one is chosen since it is supposedly particularly well-suited for simulations with adaptive grid refinement, which is one natural use case of load balancing in general. The proposed default values for the load imbalance tolerance, `ubvec` $= 1.05$, and the ratio between inter-process communication time compared to data redistribution times, `itr` $= 1000$, are used.

4.3. Simulation Setup

The simulation domain is a rectangular box that is subdivided into $12 \times 12 \times 16$ blocks of size 32^3 cells. This results in a total of 2304 blocks. By using $N_p = 256$ processes of the SuperMUC cluster, a uniform block distribution is achieved with 9 blocks per process. We use four inclined planes to construct a fluid domain where the horizontal cross section is reduced by 60% towards the bottom plane, resembling a hopper as illustrated in Figure 9. As a result, several blocks remain empty throughout the whole simulation. The domain contains about 4300 spherical particles of diameter $D/\Delta x = 15$ that are initially densely clustered along the top plane and then settle down. The Galileo number is $Ga = 50$, the density ratio is $\rho_p/\rho_f = 1.5$, the number of rigid body solver sub cycles is set to $S = 10$, and the simulation is run for 80,000 time steps. The temporal evolution of the simulation can be seen in Figure 9.

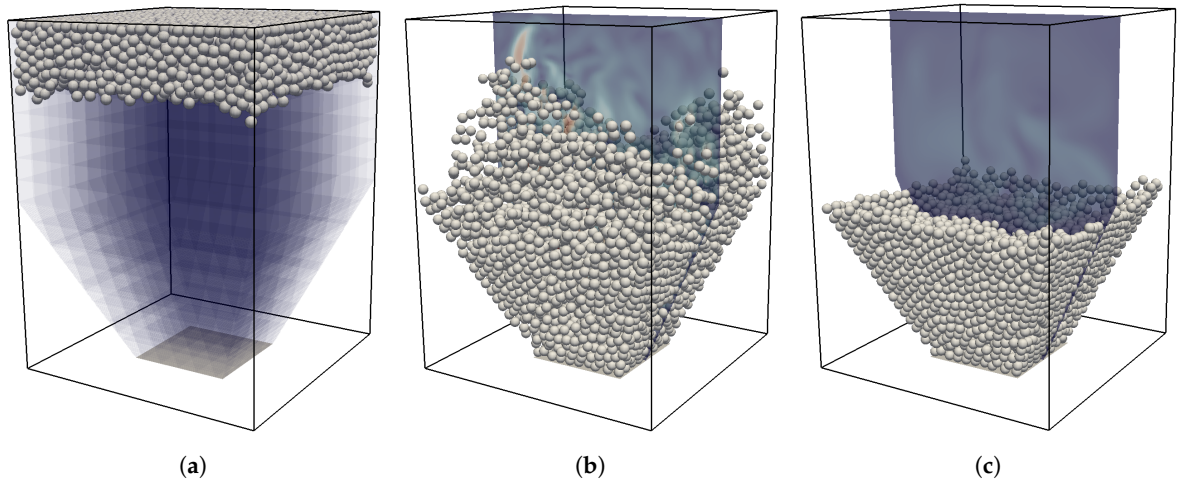


Figure 9. Visualization of the temporal evolution of the particles and the fluid velocity in the hopper clogging simulation. (**a**) Initial state with block structure. (**b**) Active settling phase. (**c**) Final, completely settled state.

In order to distribute the blocks among the available processes based on the predicted loads, we use several of the strategies presented in Section 4.2 and evaluate their performance. For space-filling curves, Hilbert and Morton orders are applied and the mentioned ParMETIS variants are also included. In all cases, the load balancing step that consists of load estimation and load

distribution is triggered every 100 time steps. These variants are compared against the case where no load balancing is applied at all. However, since the initial distribution of 9 blocks per process is non-trivial, we use the static partitioning given by the Hilbert curve for that case, which is called once at the beginning of the simulation.

4.4. Results of the Hopper Simulation

In Figure 10, the temporal evolution of the runtime for 100 time steps is shown for the four different load distribution variants and compared against the case with no load balancing. From these results, we see that without load balancing the runtime decreases until time step 40,000, which is shown in the middle of Figure 9. This corresponds to the transition from an initially densely packed bed to a fully suspended settling with only few inter-particle collisions. Afterwards, when the clustering at the bottom sets in, the simulation again gets slower. On the other hand, both cases that use space-filling curves for load distribution yield a relatively constant runtime throughout the whole simulation and are both very similar in their behavior. Except for some fluctuations halfway through the simulation, the space-filling curves are significantly faster than the default case. The two ParMETIS variants also feature low runtimes at the beginning which then steadily increase as the simulation advances, leading to larger runtimes than the default case. Again, the trend of both curves is rather similar but the `AdaptiveRepart` variant exhibits stronger oscillations in the runtime.

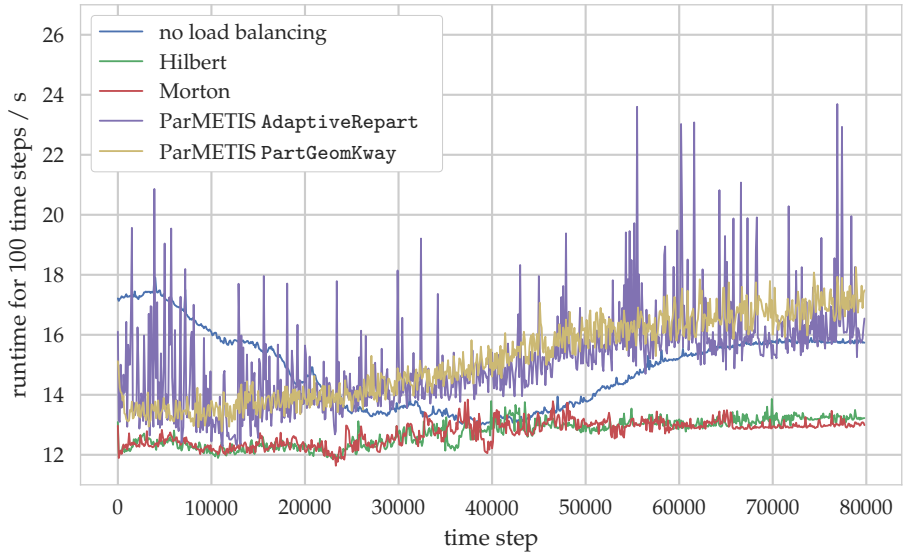


Figure 10. Runtime per 100 time steps in seconds over the time steps for simulation shown in Figure 9. Four load distribution variants using the load estimator from Section 3 are shown together with the case where no load balancing is applied.

The time to solution of the whole simulation is evaluated as the cumulated sum of the values depicted in Figure 10 and is summarized in Table 3. In total, load balancing with space-filling curves reduces the overall runtime by 14% whereas using ParMETIS even results in a slight increase of the runtime. Compared to the rather simple space-filling curves, the more complex load distribution algorithms of ParMETIS are also reflected in the larger fraction of the runtime spent in the load balancing procedure. As a result, however, the ParMETIS variants show the smallest edge cut, i.e., the sum of all edge weights from Equation (17) that are cut by process boundaries.

Table 3. Comparison of total runtime of the hopper simulation for the four load distribution variants relative to the case without load balancing. Additionally, the runtime of the load balancing step relative to the total runtime and the simulation-averaged edge cut is given.

Variant	Relative Runtime (%)	Load Balancing Fraction (%)	Mean Edge Cut
no load balancing	100.0	0.0	4.5×10^6
Hilbert	86.0	2.0	4.3×10^6
Morton	86.0	1.9	4.8×10^6
ParMETIS `AdaptiveRepart`	101.9	3.9	3.6×10^6
ParMETIS `PartGeomKway`	102.3	6.8	3.6×10^6

Since the overall goal of load balancing is to reduce possible load imbalances, the latter ones are again evaluated in intervals of 100 time steps and shown in Figure 11. The load imbalance LI, ideally close to zero, is here defined as the normalized difference between the maximum and the average runtime per interval, i.e.,

$$LI = \frac{\max_p m_{tot}(p)}{\frac{1}{N_p}\sum_p m_{tot}(p)} - 1, \tag{18}$$

where m_{tot} is the runtime of the workload generating parts, see Equation (16), summed up over the 100 time steps, and p denotes one of the $N_p = 256$ processes. This definition takes into account that the performance of the simulation is most affected by a few processes that are slower than average, resulting in many idle processes, whereas having some processes that are faster than average is less problematic, as only a few processes have to wait. Our definition, thus, differs from [12] who used the normalized difference between the maximum and minimum runtime. The evolution of the load imbalance for the case without load balancing resembles the behavior from Figure 10 with a drop from 90% to around 40%, followed by an increase to 70%. The space-filling curves exhibit the lowest imbalances with a median of 17%, and thus, almost four times smaller than the default case. Two distinct behaviors can be seen for ParMETIS: the `PartGeomKway` variant is slightly worse than the space-filling curves with an median of 22% whereas the `AdaptiveRepart` case shows larger imbalances together with strong fluctuations.

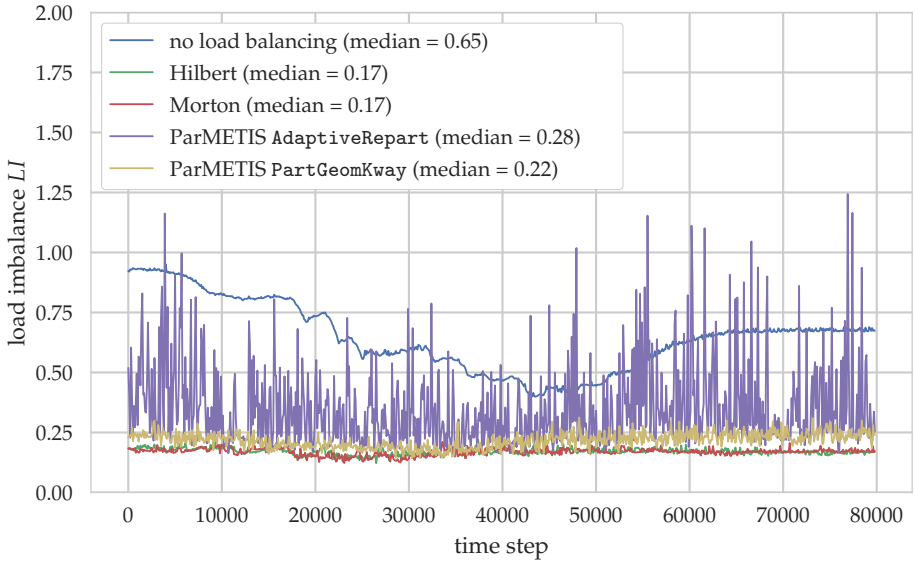


Figure 11. Load imbalance over the time steps for simulation shown in Figure 9.

4.5. Discussion

The results from this comparative study show the importance of choosing an appropriate load distribution technique. Space-filling curves achieve a reduction of the overall runtime by 14% compared to a simulation in which no load balancing is applied. On the other hand, ParMETIS, which relies on space-filling curves but combines them with more advanced partitioning strategies, does not yield an improvement in the runtime. Even though the ParMETIS variants show a reduction of the load imbalance and the edge cut, they ultimately fail in reducing the time to solution. One reason for that is certainly but only partly that the runtimes for the load balancing step are significantly higher than for the space-filling curves. Also, some improvements can be expected by optimizing the parameters that have to be specified for the ParMETIS routines. This specialization, however, runs contrary to a widely and readily applicable load distribution technique. Another reason is that ParMETIS is not designed to work with an average of, here, 9 blocks per process but usually expects around 1000 or more. This would e.g., be the case for cell-based domain partitioning, but unrealistic for block-based ones, and with this larger granularity the partitioning routines are expected to work faster and better. The observed strong fluctuations of the ParMETIS variants in the runtime and load imbalance graphs are, thus, possibly a direct cause of this difficulty. Furthermore, our choice of the edge weights, Equation (17), can be improved as it currently is solely based on the volume of the communication of the LBM data structures. Thus, the communication by the particle simulation is not regarded but might change the weights notably. One could, therefore, also try to find an estimator for the edge weights similarly to the procedure used to come up with a vertex weight estimator.

We note that these results are specific for the here simulated setup and cannot be generalized easily. They also show, however, that load balancing is a viable tool to reduce the time to solution if carefully applied and systematically compared. It is especially helpful for cases where strong heterogeneities exist in the simulation, as it is the case for the initial and end phase of the here carried out simulation. We explicitly point out that all presented load balancing techniques yield the same simulation result.

Besides ParMETIS, also other graph partitioning libraries are available, e.g., Zoltan [53], PT-Scotch [54], or Geographer [55]. In combination with a better estimator for the communication weights, their performance should, thus, be evaluated and again be compared to the space-filling curves in future work.

5. Conclusions

In this work, we present and evaluate different techniques to improve the performance of particulate flow simulations that exhibit spatially and temporally varying workloads during their parallel execution.

To this end, first a workload estimator is designed and calibrated that predicts the workload for each block of the domain using locally available information. It is based on analyzing the involved algorithms and setting up suitable functions that describe the generated load. The coefficients of the functions are determined by fitting them to timing measurements obtained from simulations of a small but representative setup. Choosing such a setup is a crucial step as all relevant phenomena and properties that can lead to varying workloads must be included to obtain a generally applicable estimator. Since timing measurements are influenced by various factors, like the hardware used or the actual implementation details, major changes in these might require a reevaluation of the fitted coefficients. Even though the article focuses on particulate flow simulations, the presented approach can be readily used for other multiphysics simulations.

In a second step, this workload estimator is applied in a more complex simulation scenario in conjunction with a load distribution technique to reduce load imbalances between the processes. We compare and evaluate load distribution via space-filling curves and with the help of the graph partitioning tool ParMETIS to the case without any load balancing. For the case of space-filling curves, a significant reduction of the time to solution by 14% can be observed, demonstrating how these techniques can be used to utilize hardware resources more efficiently by minimizing load imbalances.

On the other hand, the ParMETIS-based distributions are unable to improve the runtime for this specific scenario. This can be attributed to the relatively large computational overhead produced by the load balancing step but also generally to the different focus of these graph partitioning tools that would require a finer granularity.

All in all, this study shows that load balancing is an effective tool for parallel multiphysics simulations where workloads change in time and space. It is typically used in combination with adaptive mesh refinement, where it becomes a necessity due to the strongly varying work loads when the grid is updated. Thus, naturally, we aim to apply the here presented load balancing approach with AMR to enable efficient simulations of various engineering applications. But, as we have shown, it can also be advantageous to apply it with uniform grids to reduce the time to solution of the simulation.

Author Contributions: C.R. developed the presented algorithms, implemented them, realized the simulations and evaluated the results. U.R. supervised the work and contributed to the arrangement of this article.

Funding: This research received no external funding.

Acknowledgments: We thank Sebastian Eibl for valuable input and discussions. We acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (www.lrz.de). They also acknowledge support by Deutsche Forschungsgemeinschaft and Friedrich-Alexander-Universität Erlangen–Nürnberg (FAU) within the funding programme Open Access Publishing.

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

References

1. Kusumaatmaja, H.; Hemingway, E.J.; Fielding, S.M. Moving contact line dynamics: From diffuse to sharp interfaces. *J. Fluid Mech.* **2016**, *788*, 209–227. [CrossRef]
2. Rettinger, C.; Rüde, U. A comparative study of fluid–particle coupling methods for fully resolved lattice Boltzmann simulations. *Comput. Fluids* **2017**, *154*, 74–89. [CrossRef]
3. Biegert, E.; Vowinckel, B.; Meiburg, E. A collision model for grain-resolving simulations of flows over dense, mobile, polydisperse granular sediment beds. *J. Comput. Phys.* **2017**, *340*, 105–127. [CrossRef]
4. Anderl, D.; Bogner, S.; Rauh, C.; Rüde, U.; Delgado, A. Free surface lattice Boltzmann with enhanced bubble model. *Comput. Math. Appl.* **2014**, *67*, 331–339. [CrossRef]
5. Rettinger, C.; Godenschwager, C.; Eibl, S.; Preclik, T.; Schruff, T.; Frings, R.; Rüde, U. Fully Resolved Simulations of Dune Formation in Riverbeds. In *High Performance Computing*; Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 3–21, doi:10.1007/978-3-319-58667-0_1.
6. Markl, M.; Körner, C. Powder layer deposition algorithm for additive manufacturing simulations. *Powder Technol.* **2018**, *330*, 125–136. [CrossRef]
7. Long, G.; Liu, S.; Xu, G.; Wong, S.W.; Chen, H.; Xiao, B. A Perforation-Erosion Model for Hydraulic-Fracturing Applications. *SPE Prod. Oper.* **2018**, *33*. [CrossRef]
8. Godenschwager, C.; Schornbaum, F.; Bauer, M.; Köstler, H.; Rüde, U. A Framework for Hybrid Parallel Flow Simulations with a Trillion Cells in Complex Geometries. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13), Denver, CO, USA, 17–22 November 2013; ACM: New York, NY, USA, 2013; pp. 35:1–35:12. [CrossRef]
9. Hendrickson, B.; Devine, K. Dynamic load balancing in computational mechanics. *Comput. Methods Appl. Mech. Eng.* **2000**, *184*, 485–500. [CrossRef]
10. Cundall, P.A.; Strack, O.D.L. A discrete numerical model for granular assemblies. *Géotechnique* **1979**, *29*, 47–65. [CrossRef]
11. Yurkin, M.; Hoekstra, A. The discrete dipole approximation: An overview and recent developments. *J. Quant. Spectrosc. Radiat. Transf.* **2007**, *106*, 558–589. [CrossRef]
12. Owen, D.; Feng, Y.; Han, K.; Peric, D. *Dynamic Domain Decomposition and Load Balancing in Parallel Simulation of Finite/Discrete Elements*; ECCOMAS 2000: Barcelona, Spain, 2000.
13. Kloss, C.; Goniva, C.; Hager, A.; Amberger, S.; Pirker, S. Models, algorithms and validation for opensource DEM and CFD–DEM. *Prog. Comput. Fluid Dyn. Int. J.* **2012**, *12*, 140–152. [CrossRef]

14. Eibl, S.; Rüde, U. A Systematic Comparison of Dynamic Load Balancing Algorithms for Massively Parallel Rigid Particle Dynamics. *arXiv* **2018**, arXiv:1808.00829.
15. Deiterding, R. Block-structured adaptive mesh refinement-theory, implementation and application. *ESAIM Proc.* **2011**, *34*, 97–150. [CrossRef]
16. Lintermann, A.; Schlimpert, S.; Grimmen, J.; Günther, C.; Meinke, M.; Schröder, W. Massively parallel grid generation on HPC systems. *Comput. Methods Appl. Mech. Eng.* **2014**, *277*, 131–153. [CrossRef]
17. Schornbaum, F.; Rüde, U. Massively Parallel Algorithms for the Lattice Boltzmann Method on NonUniform Grids. *SIAM J. Sci. Comput.* **2016**, *38*, C96–C126. [CrossRef]
18. Qi, J.; Klimach, H.; Roller, S. Implementation of the compact interpolation within the octree based Lattice Boltzmann solver Musubi. *Comput. Math. Appl.* **2016**. [CrossRef]
19. Deiterding, R.; Wood, S.L. Predictive wind turbine simulation with an adaptive lattice Boltzmann method for moving boundaries. *J. Phys. Conf. Ser.* **2016**, *753*, 082005. [CrossRef]
20. Schornbaum, F.; Rüde, U. Extreme-Scale Block-Structured Adaptive Mesh Refinement. *SIAM J. Sci. Comput.* **2018**, *40*, C358–C387. [CrossRef]
21. Bader, M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*; Springer Science & Business Media: Berlin, Germany, 2012; Volume 9, doi:10.1007/978-3-642-31046-1.
22. Buluç, A.; Meyerhenke, H.; Safro, I.; Sanders, P.; Schulz, C. Recent Advances in Graph Partitioning. In *Algorithm Engineering: Selected Results and Surveys*; Kliemann, L., Sanders, P., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 117–158, doi:10.1007/978-3-319-49487-6_4.
23. Schneiders, L.; Grimmen, J.H.; Meinke, M.; Schröder, W. An efficient numerical method for fully-resolved particle simulations on high-performance computers. *PAMM* **2015**, *15*, 495–496. [CrossRef]
24. Vowinckel, B.; Jain, R.; Kempe, T.; Fröhlich, J. Entrainment of single particles in a turbulent open-channel flow: A numerical study. *J. Hydraul. Res.* **2016**, *54*, 158–171. [CrossRef]
25. Agudo, J.R.; Illigmann, C.; Luzi, G.; Laukart, A.; Delgado, A.; Wierschem, A. Shear-induced incipient motion of a single sphere on uniform substrates at low particle Reynolds numbers. *J. Fluid Mech.* **2017**, *825*, 284–314. [CrossRef]
26. Long, G.; Xu, G. The effects of perforation erosion on practical hydraulic-fracturing applications. *SPE J.* **2017**, *22*, 645–659. [CrossRef]
27. Pickl, K.; Pande, J.; Köstler, H.; Rüde, U.; Smith, A.S. Lattice Boltzmann simulations of the bead-spring microswimmer with a responsive stroke - from an individual to swarms. *J. Phys. Condens. Matter* **2017**, *29*, 124001. [CrossRef] [PubMed]
28. Buwa, V.V.; Roy, S.; Ranade, V.V. Three-phase slurry reactors. In *Multiphase Catalytic Reactors*; John Wiley & Sons, Ltd.: New York, NY, USA, 2016; Chapter 6, pp. 132–155, doi:10.1002/9781119248491.ch6.
29. Ge, W.; Chang, Q.; Li, C.; Wang, J. Multiscale structures in particle–fluid systems: Characterization, modeling, and simulation. *Chem. Eng. Sci.* **2019**. [CrossRef]
30. Munjiza, A.; Owen, D.; Bicanic, N. A combined finite-discrete element method in transient dynamics of fracturing solids. *Eng. Comput.* **1995**, *12*, 145–174. [CrossRef]
31. Chen, S.; Doolen, G.D. Lattice Boltzmann Method for Fluid Flows. *Annu. Rev. Fluid Mech.* **1998**, *30*, 329–364. [CrossRef]
32. Krüger, T.; Kusumaatmaja, H.; Kuzmin, A.; Shardt, O.; Silva, G.; Viggen, E.M. *The Lattice Boltzmann Method*; Springer: Berlin, Germany, 2017.
33. Ginzburg, I.; Verhaeghe, F.; d'Humieres, D. Two-relaxation-time lattice Boltzmann scheme: About parametrization, velocity, pressure and mixed boundary conditions. *Commun. Comput. Phys.* **2008**, *3*, 427–478.
34. Qian, Y.H.; D'Humières, D.; Lallemand, P. Lattice BGK Models for Navier-Stokes Equation. *EPL (Europhys. Lett.)* **1992**, *17*, 479. [CrossRef]
35. Preclik, T.; Rüde, U. Ultrascale simulations of non-smooth granular dynamics. *Comput. Part. Mech.* **2015**, *2*, 173–196. [CrossRef]
36. Preclik, T. Models and Algorithms for Ultrascale Simulations of Non-Smooth Granular Dynamics. Ph.D. Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, 2014.
37. Aidun, C.K.; Lu, Y.; Ding, E.J. Direct analysis of particulate suspensions with inertia using the discrete Boltzmann equation. *J. Fluid Mech.* **1998**, *373*, 287–311. [CrossRef]

38. Wen, B.; Zhang, C.; Tu, Y.; Wang, C.; Fang, H. Galilean invariant fluid-solid interfacial dynamics in lattice Boltzmann simulations. *J. Comput. Phys.* **2014**, *266*, 161–170. [CrossRef]
39. The WALBERLA Framework. Available online: http://walberla.net (accessed on 26 November 2018).
40. Iglberger, K.; Rüde, U. Massively parallel rigid body dynamics simulations. *Comput. Sci. Res. Dev.* **2009**, *23*, 159–167. [CrossRef]
41. Wellein, G.; Zeiser, T.; Hager, G.; Donath, S. On the single processor performance of simple lattice Boltzmann kernels. *Comput. Fluids* **2006**, *35*, 910–919. [CrossRef]
42. Wittmann, M.; Haag, V.; Zeiser, T.; Köstler, H.; Wellein, G. Lattice Boltzmann benchmark kernels as a testbed for performance analysis. *Comput. Fluids* **2018**, *172*, 582–592. [CrossRef]
43. LIKWID. Available online: https://github.com/RRZE-HPC/likwid (accessed on 26 November 2018).
44. Treibig, J.; Hager, G.; Wellein, G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, San Diego, CA, USA, 13–16 September 2010; pp. 207–216. [CrossRef]
45. Feichtinger, C.; Habich, J.; Köstler, H.; Rüde, U.; Aoki, T. Performance modeling and analysis of heterogeneous lattice Boltzmann simulations on CPU–GPU clusters. *Parallel Comput.* **2015**, *46*, 1–13. [CrossRef]
46. Riesinger, C.; Bakhtiari, A.; Schreiber, M.; Neumann, P.; Bungartz, H.J. A Holistic Scalable Implementation Approach of the Lattice Boltzmann Method for CPU/GPU Heterogeneous Clusters. *Computation* **2017**, *5*. [CrossRef]
47. Pohl, T.; Kowarschik, M.; Wilke, J.; Iglberger, K.; Rüde, U. Optimization and profiling of the cache performance of parallel lattice Boltzmann codes. *Parallel Process. Lett.* **2003**, *13*, 549–560. [CrossRef]
48. Hammer, J.; Eitzinger, J.; Hager, G.; Wellein, G. Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels. In *Tools for High Performance Computing 2016*; Niethammer, C., Gracia, J., Hilbrich, T., Knüpfer, A., Resch, M.M., Nagel, W.E., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 1–22, doi:10.1007/978-3-319-56702-0_1
49. Karypis, G.; Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* **1998**, *20*, 359–392. [CrossRef]
50. ParMETIS. Available online: http://glaros.dtc.umn.edu/gkhome/views/metis/ (accessed on 26 November 2018).
51. Karypis, G.; Kumar, V. Multilevel Algorithms for Multi-constraint Graph Partitioning. In Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (SC '98), Orlando, FL, USA, 7–13 November 1998; IEEE Computer Society: Washington, DC, USA, 1998; pp. 1–13. [CrossRef]
52. Kirk, S.; George, K.; Vipin, K. Parallel static and dynamic multi-constraint graph partitioning. *Concurr. Comput. Pract. Exp.* **2002**, *14*, 219–240. [CrossRef]
53. Boman, E.G.; Catalyurek, U.V.; Chevalier, C.; Devine, K.D. The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering, and Coloring. *Sci. Program.* **2012**, *20*, 129–150. [CrossRef]
54. Chevalier, C.; Pellegrini, F. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Comput.* **2008**, *34*, 318–331. [CrossRef]
55. von Looz, M.; Tzovas, C.; Meyerhenke, H. Balanced k-means for Parallel Geometric Partitioning. *arXiv* **2018**, arXiv:1805.01208.