Direct Numerical Simulation of Particulate Flows on 294,912 Processor Cores

Jan Götz*, Klaus Iglberger*, Markus Stürmer*, and Ulrich Rüde*:**

*Chair for System Simulation, University Erlangen-Nuremberg, Cauerstr. 6, 91058 Erlangen, Germany
**Cluster of Excellence “Engineering of Advanced Materials”, Nügelsbachstr. 49b, 91052 Erlangen, Germany

Abstract—This paper describes computational models for particle-laden flows based on a fully resolved fluid-structure interaction. The flow simulation uses the Lattice Boltzmann method, while the particles are handled by a rigid body dynamics algorithm. The particles can have individual non-spherical shapes, creating the need for a non-trivial collision detection and special contact models. An explicit coupling algorithm transfers momenta from the fluid to the particles in each time step, while the particles impose moving boundaries for the flow solver. All algorithms and their interaction are fully parallelized. Scaling experiments and a careful performance analysis are presented for up to 294,912 processor cores of the Blue Gene at the Jülich Supercomputing center. The largest simulations involve 264 million particles that are coupled to a fluid which is simultaneously resolved by 150 billion cells for the Lattice Boltzmann method. The paper will conclude with a computational experiment for the segregation of suspensions of particles of different density, as an example of the many industrial applications that are enabled by this new methodology.

Index Terms—Lattice Boltzmann method, rigid body dynamics, massively parallel simulation, fluid-structure interaction, fast frictional dynamics algorithm

I. INTRODUCTION

Many phenomena in nature require models in which different physical systems must be coupled. In this paper we will deal with such a multi-physics simulation and will present computational models for two phase flows when a solid phase is suspended in a fluid phase. Such systems are highly complex and are considered difficult to model and predict. Typical effects in suspensions are the segregation or sedimentation of the particles, or the fluidization when a densely packed collection of particles is subjected to an increasing flow rate of the fluid phase. Many current research efforts are still focused on improving the fundamental understanding of such systems. Since particulate flows also occur in many technical processes, such theoretical research has also immediate practical relevance.

In this paper we will propose an explicit fully resolved two-way coupling between two different physical systems in order to model particulate systems immersed in a fluid. For the flow simulation we will use the Lattice Boltzmann algorithm, a so-called kinetic method which operates on an Eulerian grid.

The suspended solid particles are represented by a rigid body simulation algorithm that uses a Lagrangian formulation for computing the trajectories of rigid objects including their rotation. This rigid body simulation includes efficient algorithms for detecting contacts between moving objects, plus the mechanisms for computing their response to collisions and frictional contacts. In our approach, the moving objects can have arbitrary shape and are not restricted to spheres.

Both programs are fully parallelized. They are coupled via an explicit exchange of momenta from the fluid onto the objects, and by modeling moving boundaries on the surface of the objects to transfer momenta back to the fluid.

Numerous numerical methods have already been suggested to simulate particulate flows. This includes for instance Stokesian dynamics [1], Euler–Lagrangian methods [2], distributed Lagrange multiplier methods [3] and discrete element methods [4, e.g.], just to name a few. Approaches based on the Lattice Boltzmann method have been presented by Ladd [5], [6], Aidun et al. [7] and Qi [8]. Most of these models do not attempt to implement a fully resolved simulation of the fluid-structure interaction between the solid particles and the fluid. Instead they must use approximations to reduce the computational complexity. For instance, some of these methods can only be applied in special flow regimes (for example potential or Stokes flow) or they choose to approximate the particles as dimensionless point masses, which can then e.g. not rotate in the flow. However, many physical effects depend on the shape and size of the particles. These effects cannot be simulated with these simplified models. Results for direct numerical simulation of particulate flows involving a large number of particles were e.g. presented by Yin and Sundaresan [9] with up to 5207 particles and Jin at al [10] with 21,336 spherical particles.

In our approach we extend and generalize these earlier methods. As described in a previous paper [11], our coupled fluid-structure-interaction method results in a detailed simulation of large ensembles of rigid, but geometrically resolved objects immersed in the flow. In this paper we demonstrate that current peta-scale supercomputers enable us to model particulate suspensions from first principles based on microscopic and mesoscopic models, but with ensemble sizes that the total system approaches macroscopic scale. These simulations are therefore of immediate technological relevance.

The paper is organized as follows. Section II will introduce the parallel LBM algorithm and Section III will present the parallelization of the rigid body dynamics simulation. The core of the paper is contained in Section IV, which will discuss...
the two-way coupling of these algorithms in the \textsc{walberla} (widely applicable Lattice Boltzmann from Erlangen) software framework. In Section V two example simulations of real-world scenarios are presented. The performance and scaling behavior of the coupled simulation on a Blue Gene system of up to 294,912 processor cores will be analyzed in detail in Section VI, before Section VII concludes the paper.

II. PARALLEL LATTICE BOLTZMANN SIMULATIONS

The Lattice Boltzmann method (LBM) can be used as an alternative to classical Navier–Stokes solvers for fluid flow. It uses an equidistant grid of lattice cells, which only interact with their direct neighbors in each time step. In our simulator we use the LBM on the same (block-)structured mesh even when the objects are moving. In other methods the automatic generation of a body fitted mesh during the simulation is a difficult problem. Additionally, eliminating the need for remeshing is in our experience a significant advantage with respect to performance on each single core and within a large-scale parallel system.

In this study we use the common three dimensional D3Q19 model originally developed by Qian, d’Humières and Lallemand [12] with $N = 19$ particle distribution functions (PDFs) $f_\alpha : \Omega \times T \mapsto [0; 1]$, where $\Omega \subset \mathbb{R}^3$ and $T \subset \mathbb{R}$ are the physical and time domain, respectively. Figure 1 gives an impression of the according 19-point stencil. The corresponding dimensionless discrete velocity set is denoted by $\{e_\alpha | \alpha = 0, \ldots, N - 1\}$. This model has been shown to be both stable and efficient [13]. For the work presented in this paper, we adopt a Lattice Boltzmann collision scheme proposed by Bhatnagar, Gross and Krook (LBGK) [14], [12]

\[ f_\alpha(x_i + e_\alpha \Delta t, t + \Delta t) = f_\alpha(x_i, t) - \frac{1}{\tau} [f_\alpha(x_i, t) - f_\alpha^{(eq)}(x_i, t)], \quad (1) \]

where $x_i$ is a cell in the discretized simulation domain, $t$ is the current time step whereas $t + \Delta t$ is the next time step, $\tau$ is the relaxation time in units of time step $\Delta t$ (which is set to be 1 here) and $f_\alpha^{(eq)}$ represents the equilibrium distribution.

Equation (1) is separated into two steps, known as the collision step and the streaming step, respectively:

\[ \tilde{f}_\alpha(x_i, t) = f_\alpha(x_i, t) - \frac{1}{\tau} [f_\alpha(x_i, t) - f_\alpha^{(eq)}(x_i, t)], \quad (2) \]

\[ f_\alpha(x_i + e_\alpha \Delta t, t + \Delta t) = \tilde{f}_\alpha(x_i, t), \quad (3) \]

where $\tilde{f}_\alpha$ denotes the post-collision state of the distribution function. The collision step is a local single-time relaxation towards equilibrium. While the collision is compute intensive, the streaming step advects all PDFs except $f_\alpha$ to their neighboring lattice site depending on the velocity, which is a memory intensive operation.

As a first order no-slip boundary condition often a simple bounce-back scheme is used, where distribution functions pointing to a neighboring wall are just reflected such that both normal and tangential velocities vanish:

\[ f_\alpha(x_f, t) = \tilde{f}_\alpha(x_f, t), \quad (4) \]

with $\alpha$ representing the index of the opposite direction of $\alpha$, $e_\alpha = -e_\alpha$, and $x_f$ explicitly denoting the fluid cell. More details on the Lattice Boltzmann algorithm and its derivation can be found in Succi et al. [15] or Chen et al. [16].

Due to the locality of the cell updates, the LBM can be implemented extremely efficiently (see for instance [17], [18], [19], [20], [21], [22]). For the same reason, the parallelization of LBM is comparatively straightforward [18], [23]. It is based on a subdomain partitioning that is realized in the \textsc{walberla} framework by a patch data structure as described in [24]. Generally, the flow domain is subdivided into patches of equal size, but several patches can be assigned to each process. The patches are surrounded by a ghost layer which is updated before each time step. The message sent from one process to another is composed from several smaller messages. To reduce the message startup time overhead, data is accumulated in buffers before sending it to neighboring processes.

III. PARALLEL RIGID BODY DYNAMICS

Particles incorporated in a flow are often realized using an approximation as point masses. This approach can be efficiently parallelized and allows for a large number of incorporated particles. However, the downside of this approach is the complete abstraction from the actual geometry of the particles. Therefore specific physical effects such as for instance particle-particle interaction due to collisions have to be modeled by potentials instead of direct momentum exchange between particles.

A different simulation approach is rigid body dynamics [25]. Unlike particle-based simulation approaches, the true shape of the granular objects is taken into account, i.e., all objects are fully resolved. This allows the simulation of arbitrarily shaped particles. In addition, contacts between objects are fully resolved, i.e., elastic and inelastic collisions as well as frictional contacts can be incorporated, if desired. In contrast to particle-based simulation approaches that use the (virtual) deformation of particles to calculate repelling forces, rigid body dynamics considers all objects as perfectly rigid and undeformable. Therefore, the forces are calculated as a result of motion constraints of contacting bodies to prevent interpenetration.

In [26] we have introduced the parallel fast frictional dynamics (PFFD) algorithm, which enables for the first time massively parallel rigid body dynamics simulations in distributed memory environments. The PFFD algorithm is based...
on the fast frictional dynamics (FFD) solver introduced by Kaufman et al. [27] and improved by Wengenroth et al. [28]. For a detailed description of the nonparallel version of this algorithm, see [27] and [28], for the parallel version [26]. The application of the PFFD algorithm for massively parallel granular media simulations with non-spherical particles has been awarded the PRACE award 2010 [29].

In the PFFD algorithm, the simulation domain is distributed among the processes. Each process only stores the rigid bodies that are located in its part of the simulation domain. Note, however, that since rigid bodies are volumetric objects they can be simultaneously in overlap with several subdomains (see Figure 2). We therefore associate each object with the subdomain that contains its center of mass. This process stores its attributes and is responsible to perform the necessary updates of locations, translational and angular velocities, etc. The position, orientation, and velocities of remote objects (i.e., all rigid bodies whose center of mass is not contained in the local process) must be updated via MPI communication. Due to the local collision treatment inherited from the FFD algorithm, the PFFD requires only communication via MPI with the directly neighboring processes.

Algorithm 1 The parallel FFD algorithm

1: \textit{MPI communication: force synchronization}
2: for each body \( B \) do
3: \hspace{1em} first position half-step
4: \hspace{1em} first velocity half-step
5: end for
6: \textit{MPI communication: exchanging rigid bodies}
7: for each body \( B \) do
8: \hspace{1em} find all contacts \( C(B) \)
9: \hspace{1em} for each violated contact \( k \) in \( C(B) \) do
10: \hspace{2em} add collision and friction constraints to \( B \)
11: \hspace{1em} end for
12: end for
13: \textit{MPI communication: exchanging collision constraints}
14: for each body \( B \) do
15: \hspace{1em} if \( B \) has constraints then
16: \hspace{2em} find post-collision velocity
17: \hspace{2em} select friction response
18: \hspace{1em} else
19: \hspace{2em} second velocity half-step
20: \hspace{2em} end if
21: \hspace{1em} second position half-step
22: end for
23: \textit{MPI communication: exchanging rigid bodies}

Algorithm 1 illustrates a schematic overview of the PFFD algorithm. In contrast to the nonparallel algorithm, the parallel version contains a total of four MPI communication steps to handle the distributed computation (see lines 1, 6, 13, and 23). We assume that in between two time steps it is possible to interact with the simulation, i.e., change the position, orientation, and/or velocities of rigid bodies as required by the simulation scenario. Additionally, it is possible to add external forces to each of the simulated particles. Therefore the very first step of the algorithm is a synchronization of the forces: each process needs to know the total forces acting on the objects (see Algorithm 1, line 1).

With these forces it is possible to perform the first half step to move the rigid bodies by half a time step. This half step consist of an initial position half step (line 3) followed by a velocity half step (line 4). This position and velocity update is only applied for local rigid bodies, i.e., for all objects, whose center of mass is contained in the subdomain. Remote objects, i.e., bodies whose center of mass is outside the subdomain and which are owned by another process, receive an update via MPI in the second MPI communication (line 6). Since the position update might result in new rigid bodies crossing a process boundary, this communication step also notifies neighboring processes of new objects.

After the second communication step, each process has all necessary information to perform a local collision detection (see 1, lines 7 to 12). All pairwise collisions between rigid bodies are detected and analyzed. In case a contact has to be handled, collision and friction constraints are added to both contacting rigid bodies. Next follows the collisions response phase. However, this phase is applied only for local rigid bodies. Sometimes collisions of objects, i.e., constraints, can only be detected on remote processes (see Figure 2 for an example). Therefore it is necessary for this phase to synchronize all constraints acting on the rigid bodies. This synchronization is handled in the third communication step (line 13).

Subsequently, the post-collision velocity can be calculated for each rigid body. This is performed strictly locally, resulting in feasible, yet not necessarily globally correct physical results (see [28]). The next step is the second half step, moving the rigid bodies to the end of the time step. For rigid bodies not involved in any collisions, a second velocity half step is performed, which updates their velocity depending on the acting forces, before the second position half step takes place, which updates the bodies’ position according to the new velocity. The final step of the PFFD algorithm is the fourth communication step in order to update remote bodies after the position update and to notify remote processes of rigid bodies newly crossing a process boundary.

IV. COUPLING THE LATTICE BOLTZMANN METHOD AND RIGID BODY DYNAMICS

The coupling between the Lattice Boltzmann flow solver and the rigid body dynamics simulation has to be two-way: rigid bodies have to be represented as (moving) boundaries in the flow simulation, whereas the flow corresponds to hydrodynamic forces acting on the rigid bodies (see Figure 3). We use an explicit coupling algorithm, as shown in Algorithm 2.

The first step in the coupled algorithm is the mapping of all rigid bodies onto the Lattice Boltzmann grid (see Figure 4(a) for an example). Objects are thus represented as flag fields for the flow solver. In our implementation, each lattice node with a cell center inside an object is treated as a moving boundary. For these cells, we apply the following boundary condition, which is a variation of the standard no-slip boundary conditions for moving walls [30]:

\begin{align}
\mathbf{u}_l &= \mathbf{u}_b \\
\mathbf{f}_l &= \mathbf{f}_b
\end{align}
Fig. 2. Setup of a simulation of two MPI processes. Rigid bodies are exclusively managed by the process their reference point (in our case the center of mass) belongs to. In case they are partially contained in the domain of a remote process, they have to be synchronized with the other process, where they are treated as remote bodies.

Fig. 3. Illustration of the two way coupling of WALBERLA fluid solver and PE rigid body dynamics solver.

Algorithm 2 Coupled LBM-PE solver

1: MPI communicate ghost layer of velocity and density
2: for each body B do
3: Map B to lattice grid
4: end for
5: MPI communicate ghost layer of PDFs
6: for each lattice cell do
7: Stream and collide
8: end for
9: for each surface cell do
10: Add forces from fluid to rigid objects
11: end for
12: Time step in the rigid body simulation

\[ f_\alpha(x_f, t) = \tilde{f}_\alpha(x_f, t) + 6w_\alpha \rho_w e_\alpha \cdot u_w, \]  

where \( \rho_w \) is the fluid density close to the wall and the current velocity \( u_w \) of each object cell corresponds to the velocity of the object at the cell’s position. Thus, rotational as well as translational velocities of the objects are taken into account on the object surface.

Furthermore, in this step flag changes due to the movement of the objects must be treated. Here, two cases can occur (see Figure 4(b)): First, fluid cells \( x_f \) can turn into object cells. This results in a conversion of the cell to a moving boundary. In the reverse case, that a boundary cell turns into a fluid cell, the missing distribution functions have to be reconstructed. In our implementation, the missing PDFs are set to the equilibrium distributions \( f^{eq}_\alpha(\rho, u) \), where the macroscopic velocity \( u \) is given by the velocity \( u_w \) of the object cell and the density \( \rho \) is computed as an average of the surrounding fluid cells. Further details are described in Iglberger et al. [31].

Note that for an efficient mapping step, and in order not to introduce additional communication steps for the mapping process, rigid objects have to reside on the same processes as the LBM patches they are contained in. Since the rigid body simulation is independent of any grid, the subdomain partitioning of the rigid body simulation is adapted to the LBM partitioning. Therefore each process knows about all particles that are (partially) inside its own subdomain.

During the subsequent stream and collide step, the fluid flow acts through hydrodynamic forces on the rigid objects: Fluid particles stream from their cells to neighboring cells and, in case they enter a cell occupied by a moving object, are reversed, causing a momentum exchange between the fluid and the particles. The total force \( F \) resulting from this momentum exchange [30] can be easily evaluated due to the kinetic origin.
of the LBM by

\[ F = \sum_{x_b} \sum_{\alpha=1}^{19} e_\alpha [2f_\alpha(x_f, t) + 6w_\alpha \rho_\alpha e_\alpha \cdot u_w] \frac{\Delta x}{\Delta t}, \]

where \( x_b \) are all obstacle cells of the object neighboring to at least one fluid cell.

The final step of the coupled algorithm is a time half step in the rigid body dynamics framework, which treats rigid body collisions and moves the objects according to the acting forces. This results in a position change of the objects, which have therefore again to be mapped to the LBM grid. A validation of this method is described in Iglberger et al. [31], whereas in Binder et al. [32], this method is used to simulate the drag force on agglomerated particles.

V. SIMULATION OF PARTICLE SEGREGATION

In this section, two simulations of real world scenarios with a moderate number of particles are shown. Please note, that the size of the particles and hence the number of particles was chosen such that single particles still can be identified in the resulting pictures. Additionally, both simulation and visualization are highly computational time consuming for scenarios with millions of objects. Thus, for larger number of particles only performance results are shown and discussed in Section VI. In all cases objects with density values of 0.8 kg/dm³ and 1.2 kg/dm³ are immersed in water with density 1 kg/dm³ and a gravitation field. Thereby light objects will rise to the top, while heavy objects will fall to the ground. Figure 5 shows a simulation with 242 200 objects of spherical shape and a lattice domain size of 384 \times 384 \times 800 cells. The simulation is running 12 hours on 32 768 processor cores.

These types of simulations enable a detailed and accurate analysis of physical phenomena like entrainment and mixing of particles. The simulation results will give important insight into physical phenomena. Using the coupled CFD- multibody solver, detailed parameter studies can be done, which are hardly possible with measurements and open entirely novel possibilities as far as simulations are concerned.

VI. PERFORMANCE RESULTS

WALBERLA is a portable framework and has already been used on a variety of different supercomputers, e.g. the HLRB II, a SGI Altix 4700 system in Garching, Germany, the HECToR, a Cray XT 4 system in Edinburgh, UK and the Juropa, a Bull and Sun system located in Jülich, Germany.

In this chapter we present performance and scalability results for the Jugene, an IBM Blue Gene/P system [33] located at the Jülicher Supercomputing Centre, Germany [34] with up to 294 912 processor cores. The Jugene provides 73 728 quad-core Power 450 processor located in 72 racks. Each Power 450 processor delivers 13.6 GFlop/s peak performance, summing up to a theoretical peak performance of 1 PFlop/s for the whole system. Each processor is attached to 8 MB cache and 2 GB of main memory, resulting in 144 TB of aggregated main memory for the overall system. The maximum achievable bandwidth of the main memory is 13.6 GB/s. The entire system is connected by different networks. For general-purpose, point-to-point and multicast operations, a torus network is used, which interconnects all compute nodes. Each node in the torus has six nearest-neighbor connections with a target hardware bandwidth of 425 MB/s in each direction for a total of 5.1 GB/s bidirectional bandwidth per node. This torus network also supports virtual cut-through hardware routing. Up to 256 compute nodes, the processors can only be connected in a mesh structure. A low power consumption compared to other supercomputers, the high-speed interconnect and the scalable design makes the Blue Gene architecture one of the most successful contemporary supercomputing architectures.
To compile the program, the GNU C and C++ compiler in version 4.1.2 is used. Tests with the standard IBM XLC/XLC++ compiler showed a lower performance compared to the GNU compiler when compiling with flag -O2. Higher optimization was not possible due to internal compiler errors.

In order to compare the performance values, we follow [35] and present the results in terms of million lattice updates per second (MLUPS). This allows for an estimate of the runtime for a given problem size. On architectures that perform a read for ownership before a write, WALBERLA transfers 524 Bytes per cell update (see [24]) neglecting cache effects and executing 213 floating point operations. However, in peak values, the Power 450 processor transfers one byte in the same time as it executes one floating point operation.

We evaluate two different scenarios, one labeled case A with sparsely packed particles, which appears in particulate flows, and one case B with densely packed particles representative for sedimentation or segregation processes as shown in Section V. In both scenarios, particles of a diameter of 6 lattice cells are used. For the simulations, the particles are arranged in two Cartesian directions on a regular grid with a given distance \( d \) and shifted by a random value, which is chosen to randomly distribute the objects without interpenetrating themselves. In the third Cartesian direction, layers are created with a distance of 7 lattice cells. For case A the distance \( d \) is 16 lattice cells, for case B \( d \) is 9 lattice cells, resulting in a fraction of particle volume of 6.3% and 19.8%, respectively. Additionally, two different domain sizes per core are used. One is \( 40 \times 40 \times 40 \) lattice cells, the other \( 80 \times 80 \times 80 \) lattice cells.
A. Performance and Tuning of Pure Fluid Simulations

Before we study the performance of complex scenarios we present results for a standard flow scenario, the simulation of a channel resolved by multiple subdomains of $40^3$ and $80^3$ lattice cells, respectively. Three different implementations of the stream collide loop are evaluated: A straightforward implementation, an optimized version, and an optimized version using C99 variable-length arrays (for more information on VLAs see the users journal of Randy Meyers [36], [37], [38]). Additionally, two different data layouts are tested:

- The Structure-of-Arrays (SoA) layout, where the first dimension is used to store the PDFs and the remaining dimensions for the 3D coordinate directions [17].
- The Array-of-Structures layout (AoS), where the first three dimensions are used for the 3D coordinate system, followed by the PDFs [17].

Both memory layouts show different characteristics, making one or the other more appropriate for a certain architecture. The SoA layout is optimized for streaming [17], providing pure linear access but requiring a large number of data streams. It is therefore more prone to cache thrashing effects. The AoS layout provides spatial locality only for the collision step, but the read operations are strided in the preceding streaming step. This can increase cache and memory traffic, but reduces the number of data streams significantly and simplifies address calculations.

Our results for a single node are summarized in Table I, which indicate the best performance for the AoS layout and the implementation using VLAs. These are also used in the following simulations of the coupled fluid-structure problem to achieve a maximal performance for the fluid solver. The fastest simulation achieves 4.65 GB/s main memory bandwidth and 1.17 GFlop/s on a single node, where values for GFlop/s and main memory bandwidth are measured using the automatically available performance counters [39].

An analysis of the assembly code generated for the SoA and AoS kernels using VLAs shows an advantage for the latter: In its innermost loop, it requires fewer instructions, mainly uses memory operations with relative addresses instead of separate computations, and needs to spill fewer registers. However, both kernels could provide much higher performance if accessing only data in L1 cache. More important seem to be the effects of the per-core L2 cache, which merely serves as an 14-entry prefetch and write-back buffer between L1 and shared L3 cache, which is detailed in [40]. Despite transferring less data from L3 to L1 cache, the SoA layout performs worse because it vastly exceeds the number of supported streams. Estimating the requirements of the AoS layout and using the maximum fill rate of the L1 cache, its performance is restricted to 2.9 MLUPS per core for line lengths beyond 20 cells. As the AoS kernel exceeds seven data streams and therefore cannot profit from prefetching by the L2 cache, the effective fill rate is even lower. Performance further degrades since a lattice cell update is dominated by reading in the beginning and writing at the end, with a phase of computation that cannot be overlapped with memory accesses in between. Summarizing, our best LBM performance on a Blue Gene/P node reaches 8.6% of peak Flop/s and 34.2% of peak memory bandwidth. While this may still be less than desirable in absolute terms, it presents a very significant improvement compared to the straightforward implementation and compares favorably with other Lattice Boltzmann implementations or PDE solvers on this architecture.

In the next step we present scaling results, keeping the subdomain size of $40^3 = 64$000 lattice cells per core fixed, see Table II. For 1000 time steps, the simulation takes around 45 seconds and shows excellent scalability over the whole range of processor cores considered. A similarly near-perfect scaling was reported by Clausen et al. [41], who benchmarked weak scaling on the Blue Gene/P system in Argonne with their LBM implementation. They employed a significantly smaller subdomain size of $32^3 = 32$768 lattice cells per core and achieved similar run times of 40-50 seconds for 1000 time steps with up to 32768 cores.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Time to solution (in s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>44.6</td>
</tr>
<tr>
<td>2048</td>
<td>44.6</td>
</tr>
<tr>
<td>4096</td>
<td>44.5</td>
</tr>
<tr>
<td>8192</td>
<td>44.5</td>
</tr>
<tr>
<td>16384</td>
<td>44.9</td>
</tr>
<tr>
<td>32768</td>
<td>44.9</td>
</tr>
<tr>
<td>65536</td>
<td>44.5</td>
</tr>
<tr>
<td>131072</td>
<td>44.6</td>
</tr>
</tbody>
</table>

TABLE II WEAK SCALING OF PURE LBM SIMULATION WITH $40 \times 40 \times 40$ LATTICE CELLS PER CORE AND 1000 TIME STEPS.

B. Node Performance and Tuning of Coupled Fluid-Structure Interaction Simulations

As a next step, we evaluate the node performance of the coupled fluid-particle interaction simulations, which will obviously depend on the domain size per core and the number of objects. The results are shown in Table III, where the GFlop/s and the memory bandwidth values are again measured using the automatically available performance counters [39]. For the smaller domain size of $40^3$ lattice cells per core, the performance is in general lower than for domain size of $80^3$ lattice cells due to a larger communication to computation ratio, although the performance of the pure Lattice Boltzmann solver is higher for a domain size of $40^3$ lattice cells per core (see Table I). When the number of rigid bodies is increased (case B), the computational effort for mapping the objects to the grid, the force evaluation, the movement and the collisions of the objects is higher compared to simulations with smaller number of objects (case A), resulting in a lower performance in terms of MLUPS for case B. Thus, the highest MLUPS value is obtained for a domain size of $80^3$ lattice cells and case A. For the same domain size and the same test case, also the highest measured values for GFlop/s and main memory bandwidth are achieved.
C. Multi-Node Performance of Coupled Fluid-Structure Interaction Simulations

In order to evaluate the run-time of the different computational tasks, simulations with 64 cores and 500 time steps are performed. For that number of cores and a domain decomposition of \(4 \times 4 \times 4\) the eight inner processes need to communicate to all neighbors. The results are shown in Figure 7. For case A, with sparsely packed particles, the LBM solver is the dominant part. When increasing the number of objects, the necessary computations for mapping, evaluating the force, moving and colliding the objects also increase. For case B, computing time for the force evaluation is nearly as high as for the stream and collide step. This is caused by the distributed access to the values in memory, because the force is evaluated on the surface of each object, where the cells, which are accessed are not locally aligned. For the simulations with smaller domain sizes, the ratio of communication is also higher compared to the larger domain sizes due to a larger communication to computation ratio.

The weak scaling is performed with both scenarios and two different domain sizes. The experiment is again started at 64 cores with a decomposition of \(4 \times 4 \times 4\), when 8 processes communicate in all neighboring directions. Note that placement of MPI processes needs to be done according to the requested torus shape and thus the underlying network shape on the system to maintain a high performance.

The weak scaling up to 294,912 cores of case A is presented in Figure 8. The reference values for calculating the efficiency are 44.07 MLUPS for a domain size of \(40^3\) lattice cells per core and 47.31 MLUPS for a domain size of \(80^3\) lattice cells per core for measurements with 64 cores. The lower efficiency for the smaller domain size results from a higher communication to computation ratio. Figure 9 shows the weak scaling up to 294,912 cores for test case B. The efficiency is based on measurements on 64 cores, which result in 23.06 MLUPS and 25.14 MLUPS for domain sizes of \(40^3\) and \(80^3\) lattice cells per core, respectively. Over the whole range of cores, the efficiency remains over 98% for a domain size of \(80^3\) lattice cells per core. Again, the efficiency of the simulation with smaller domain size is lower, compared to the simulation with larger domain size due to the communication to computation ratio. Note that the largest simulation on 294,912 cores utilizes more than 50 TB of main memory and contains 264,331,905 objects incorporated in the flow.

The strong scaling on Jugene ranges from 64 processes and a domain decoupling of \(4 \times 4 \times 4\) to 32,768 processes with a decoupling of \(32 \times 32 \times 32\) for a constant domain size of \(320 \times 320 \times 320\) lattice cells. By doubling the number of cores, the domain is successively cut in the Cartesian directions. The simulation time for both scenarios and the domain sizes per core are shown in Table IV. From 64 to 4,096 cores an efficiency of more than 50% is obtained. However, when using more than 4,096 processor cores the time to solution can not...
Fraction of compute time

LBM Communication
Stream Collide
Object Mapping
Force Evaluation
Physics Engine

Case A
Size 40

Case A
Size 80

Case B
Size 40

Case B
Size 80

Fig. 7. Fraction of computing time of different modules for two different scenarios and two different domain sizes per core on 64 cores.

Table IV

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Simulation time (in s) case A</th>
<th>Simulation time (in s) case B</th>
<th>Domain size per core</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>362.9</td>
<td>669.8</td>
<td>80 × 80 × 80</td>
</tr>
<tr>
<td>128</td>
<td>177.0</td>
<td>341.4</td>
<td>40 × 80 × 80</td>
</tr>
<tr>
<td>256</td>
<td>95.5</td>
<td>182.3</td>
<td>40 × 40 × 40</td>
</tr>
<tr>
<td>512</td>
<td>51.5</td>
<td>96.1</td>
<td>20 × 40 × 40</td>
</tr>
<tr>
<td>1024</td>
<td>28.6</td>
<td>52.6</td>
<td>20 × 20 × 40</td>
</tr>
<tr>
<td>2048</td>
<td>18.5</td>
<td>31.1</td>
<td>20 × 20 × 20</td>
</tr>
<tr>
<td>4096</td>
<td>12.1</td>
<td>20.0</td>
<td>10 × 20 × 20</td>
</tr>
<tr>
<td>8192</td>
<td>7.6</td>
<td>12.9</td>
<td>10 × 10 × 20</td>
</tr>
<tr>
<td>16384</td>
<td>5.7</td>
<td>8.8</td>
<td>10 × 10 × 10</td>
</tr>
<tr>
<td>32768</td>
<td>4.7</td>
<td>6.9</td>
<td>10 × 10 × 10</td>
</tr>
</tbody>
</table>

Simulation time and domain sizes for 500 time steps of strong scaling of coupled fluid-structure interaction simulations from 64 to 32 768 compute cores with scenarios A and B.

be reduced significantly anymore. At high processor counts, the number of lattice cells per core is low, but the fraction of communicated data to simulated data for both fluid simulation and simulation of rigid objects is high, which reduces the performance. This can be seen from Figures 10 and 11 for test cases A and B respectively. The time spent to communicate LBM ghost data and the time in the physics engine, which includes four steps of MPI communication in each time step, increases for larger processor counts. For an ideal scaling, the time in these routines should stay constant over the whole range of processes, as is nearly the case in the weak scaling test (see Figure 12). However, for case A and 32 768 cores, LBM communication and physics engine consume 87% of the time, compared to only 12% for the simulation with 64 cores.

VII. Conclusion

We have developed an efficient simulation system for the direct numerical simulation of particulate flows built from a combination of a Lattice Boltzmann fluid simulation and a rigid body physics engine. The results for the weak scaling on the Jugene Blue Gene/P system demonstrate a good parallel efficiency on the full machine with 294 912 cores. Using the
coupled simulation system, we are able to simulate millions of objects as fully resolved rigid bodies in a fluid flow including rotation and frictional contacts. This enables a detailed simulation of particulate flows and provides important insight into physical phenomena. We have provided results for a real-world simulation of 14 739 rigid objects of different shapes in the flow and for a simulation of 242 200 spherical objects in the flow, which is, to our knowledge, the largest direct numerical simulation reported in the literature. Future work will focus on further improvements of the method, including adaptive algorithms, dynamic load balancing, and hardware specific optimization.

a) Acknowledgments: The work was supported by the Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern (KONWIHR) under project waLBerlaMC, the European Union (EU) project DECODE (grant 213295) and the Bundesministerium für Bildung und Forschung under the SKALB project no. 01HH003A.

We would like to thank for their support: Bernd Mohr, Wolfgang Frings and Lukas Arnold, Jülich Supercomputing Centre (JSC), Germany; Robert Richter and Karl-Ernst Wirth, Chair of Particle Technology, Erlangen, Germany; Gerd Schröder-Turk, Chair of Theoretical Physics, Erlangen, Germany; Dominik Bartuschat, Christian Feichtinger and Stefan Donath, System Simulation Group, Erlangen, Germany.

REFERENCES


Information on the JUGENE. http://www.fz-juelich.de/jsc/jugene/.


