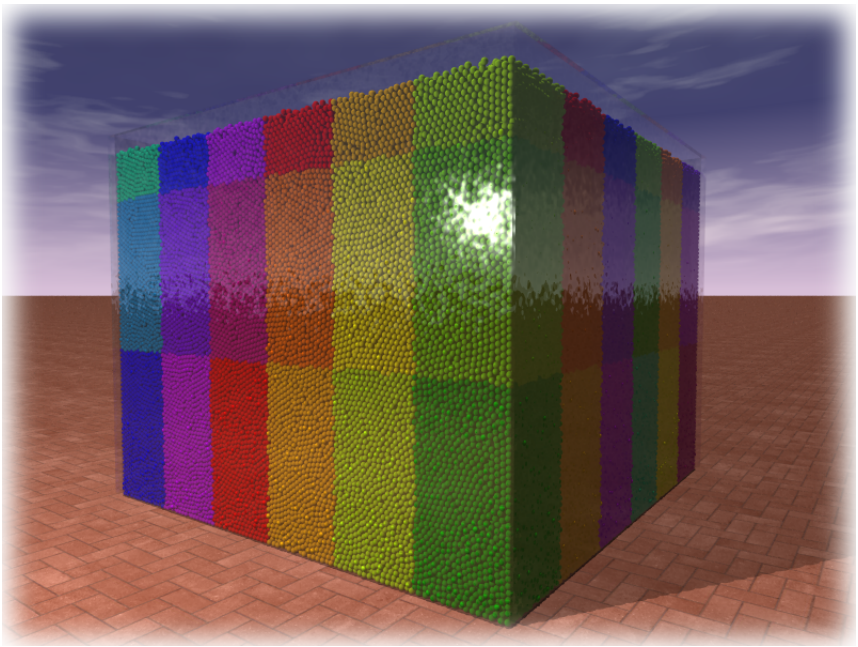
FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)

Massively Parallel Rigid Multi-Body Dynamics

Klaus Iglberger, Ulrich Rüde



Technical Report 09-8

Massively Parallel Rigid Multi-Body Dynamics

Klaus Iglberger, Ulrich Rüde

Lehrstuhl für Systemsimulation
Friedrich-Alexander University of Erlangen-Nuremberg
91058 Erlangen, Germany

klaus.iglberger@informatik.uni-erlangen.de

July 6, 2009

For decades, rigid body dynamics has been used in several active research fields to simulate the behavior of completely undeformable, rigid bodies. Due to the focus of the simulations to either high physical accuracy or real time environments, the state-of-the-art algorithms cannot be used in excess of several thousand to several ten thousand rigid bodies. Either the complexity of the algorithms would result in infeasible runtimes, or the simulation could no longer satisfy the real time aspects.

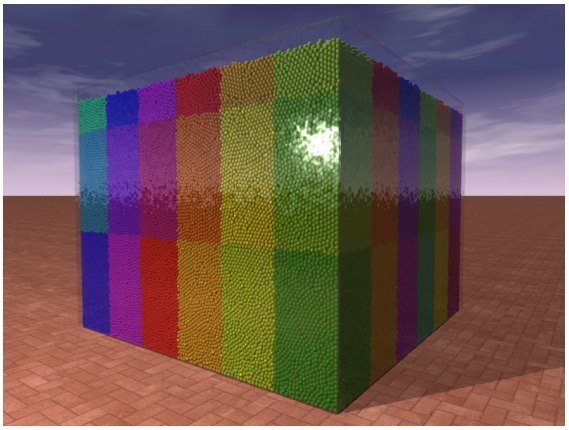
In this paper we present a novel approach for large-scale rigid body dynamics simulations. The presented algorithm enables for the first time rigid body simulations of more than one billion rigid bodies. We describe in detail the parallel rigid body algorithm and its necessary extensions for a large-scale MPI parallelization and analyze the parallel algorithm by means of a particular simulation scenario.

1 Introduction

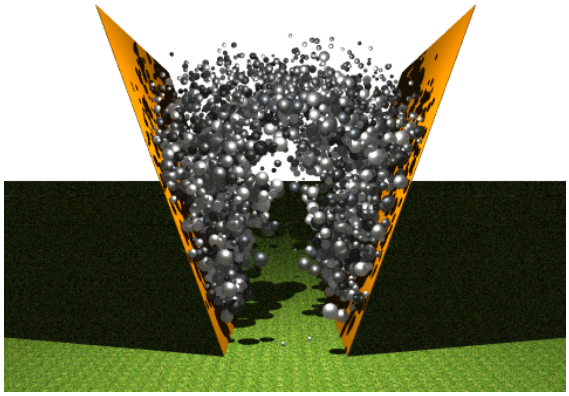
The field of rigid body dynamics is a subfield of the classical mechanics introduced by Isaac Newton in the 17th century. The only assumption made in this field is that the treated bodies are completely rigid and cannot be deformed by external forces. Though the rigidity assumption is a rigorous idealization of real materials, many phenomena in nature and problems posed in engineering can be well approximated in this way. In fact, a vast number of research communities is interested in the simulation of rigid bodies, each one focusing on different aspects of the general problem.

For instance, rigid body simulation has been used for decades in the robotics community that is primarily interested in the accurate prediction of friction forces. In material science, rigid body simulation is used for granular media simulations (see for example Fig. 1(a)) and in the computer graphics, computer games and virtual reality communities rigid body dynamics is used for authentic physical environments. For some years, rigid body dynamics has also been used in combination with lattice Boltzmann flow simulations to simulate the behavior of rigid bodies in a flow [ITR08]. This coupled simulation system is for example used to simulate fluidization processes with a high number of buoyant rigid bodies (see for example Fig. 1(b)).

The most obvious difference in the existing rigid body simulation frameworks is the applied solver for the treatment of collisions between rigid bodies. Currently, the available algorithms for the calculation of collision responses can be very coarsely subdivided into two categories: the accurate formulations based on linear complementarity problems (LCP) that are able to accurately predict frictional contact forces at higher computational costs [CPS92, RA05, Jea99, Ani06, Pre08] (they very often have a complexity of $O(NlogN)$ or even $O(N^2)$), and fast algorithms that scale linearly with the number of contacts but suffer from a reduced accuracy [Mil07, Ebe03]. The former algorithms are for instance favored in the robotics



(a) Medium-sized simulation of a granular media scenario consisting of 972 000 spheres. The colors indicate the domains of the 108 involved MPI processes.



(b) Coupled simulation between a a lattice Boltzmann flow simulation and the *pe* rigid body physics engine.

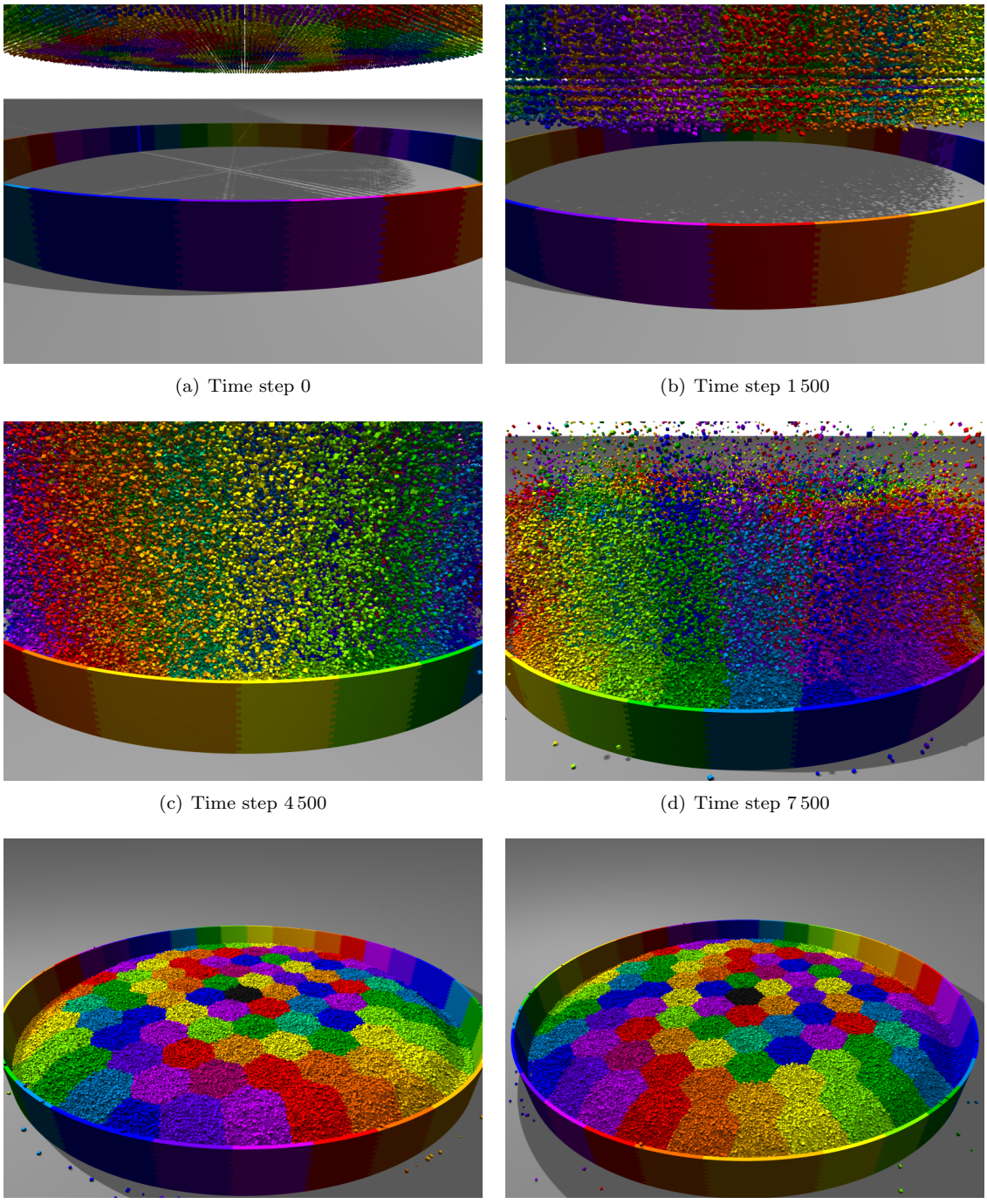
Figure 1: Simulation examples of rigid body simulations.

community because of the accuracy aspect, the latter are predominantly used for computer games and virtual reality environments.

Due to the computational costs of LCP solvers (for instance extensions of the Gauss-Seidel, Gauss-Jacobi or Conjugate Gradient methods), the maximum number of rigid bodies in such a simulation is limited to several thousand rigid bodies. A larger number of rigid bodies would result in completely infeasible runtimes. The faster algorithms, on the other hand, can handle more rigid bodies due to their linear complexity, but are often also limited to several thousand to ten thousand rigid bodies because of their real time requirements. Therefore both groups of algorithms are unsuited for the simulation of several hundred thousand or even a million rigid bodies as it would be necessary for a large-scale granular media simulation.

The algorithm presented in this paper represents the first large-scale rigid body dynamics algorithm. It is based on the fast frictional dynamics algorithm proposed by Kaufman et al. [KEP05] and is parallelized with MPI [GSL99]. Using this algorithm, our rigid body simulation framework named *pe* (the abbreviation for physics engine) enables the simulation of more than one billion interacting rigid bodies on an arbitrary number of cores. As a first example of a large-scale rigid body simulation, the example demonstrated in Fig. 2 shows the simulation of 500 000 spheres and boxes falling into a well build from 3 000 fixed boxes. The simulation domain is partitioned into 91 subdomains, where each subdomain is managed by a single MPI process. All bodies contained in a subdomain are exclusively known to this managing process. Therefore all processes have to communicate with neighboring processes about rigid bodies crossing a process boundary. Due to the hexagonal shape of each subdomain, each process has to communicate with a maximum number of six neighboring processes. Please note that this simulation only rudimentally demonstrates the full capacity of the parallel algorithm and was chosen such that individual objects are still distinguishable in the visualization.

This paper is structured in the following sections: Sec. 2 gives an overview of related work in the field of rigid body dynamics and large-scale simulations. Sec. 3 illustrates the setup of a large-scale rigid body simulation and highlights the necessary assumptions and extensions in order to be able to perform simulations with an arbitrary number of rigid bodies on an arbitrary number of processor cores. Sec. 4 takes a close look at the communication layer of our *pe* engine (see Fig. 4) and explains how heterogenous data types (integral data, floating point data, ...) and heterogenous geometry types (spheres, boxes, ...) are efficiently communicated among the MPI processes. In Sec. 5, the focus lies on the application layer. This section explains the MPI parallel algorithm of the rigid body dynamics simulation and will explain the extensions of the original fast frictional dynamics algorithm. Sec. 6 provides an analysis of the scaling behavior of the parallel algorithm, whereas Sec. 7 concludes the paper.



(a) Time step 0

(b) Time step 1 500

(c) Time step 4 500

(d) Time step 7 500

(e) Time step 10 500

(f) Time step 13 500

Figure 2: Simulation of 500 000 spheres and boxes falling into a well build from 3 000 fixed boxes. The colors indicate the domains of the 91 MPI processes. Due to the hexagonal setup of the domains, each MPI process has a maximum number of six neighboring processes.

2 Related Work

Tasora et al. [TNA08] presented a large-scale parallel multi-body dynamics algorithm for graphical processing units. Their algorithm is based on a cone complementarity problem for the simulation of frictional contact dynamics. Their approach is suited for more than one million rigid bodies. There exist several alternative rigid body simulation frameworks that basically offer the same rigid body functionalities as our framework. Two open source examples are the Open Dynamics Engine (ODE) [Homb] and the OpenTissue library [Homc]. However, currently none of them offers massively MPI parallel rigid body dynamics as introduced in this paper.

Fleissner et al. [FE07] focus on parallel simulation of particle-based discretization methods, such as the discrete element method or smoothed particle hydrodynamics, instead of rigid body dynamics. Although their approach offers the option to run large-scale particle simulations, they are only considering point masses instead of fully resolved rigid bodies.

3 Setup of a Parallel Rigid Body Simulation

The primary goal of the parallel algorithm is the simulation of an arbitrary number of rigid bodies on an arbitrary number of processor cores. In order to achieve this objective, it is strictly necessary to distribute the rigid bodies among the involved MPI processes due to basic computational and memory restrictions. Thus each process is assigned part of the global simulation domain along with all rigid bodies contained in this subdomain. However, this causes several additional problems for the parallel, distributed rigid body simulation in comparison to non-parallel simulations.

The first problem involves the volume of rigid bodies. In contrast to for example molecular dynamics simulations [GKZ08], where the particles are considered to be point masses that don't occupy any space and can therefore be uniquely related to exactly one process [1], rigid bodies have an expansion and can therefore not be considered point masses. As a result, rigid bodies may be physically present on an arbitrary number of processes at the same time as illustrated in Fig. 3. Therefore it is necessary to create a rule in order to be able to attach each rigid body to exactly one process that takes responsibility for the body. For this purpose, we are using the center of mass of each rigid body: in case the center of mass is contained in the domain of the local process, the body is considered to reside on this process. Otherwise it is a remote body. The choice of the center of mass (or any other point within the rigid body, as for instance the geometric center) makes it possible to uniquely relate each rigid body with exactly one MPI process.

Note the formulation "otherwise it is a *remote* body". In our framework, every MPI process knows only about his own local domain and about the process boundaries to his neighbors. The global partitioning of the simulation domain is therefore unknown to the processes and it is in fact never stored. From the point of view of a particular MPI process, a rigid body can therefore only be local (i.e. contained in the local domain) or remote (not contained in the local domain). It is not possible (and also not necessary) to relate a remote rigid body to a specific remote process, since the domain of remote processes is unknown. Fig. 3 illustrates the setup of two MPI processes from the point of view of both involved processes.

Another problem that is only encountered in a parallel rigid body simulation is that every rigid body in the simulation must be uniquely identifiable. In case a rigid body is overlapping a process boundary (i.e. is visible for several MPI processes), all involved processes should logically work with the same rigid body, although physically, due to the distributed memory parallelization, they work with copies of the same body.

On way to identify a rigid body is to create a unique ID for every single rigid body. The first idea for such an ID is a counter shared among all processes that is increased every time a rigid body is created. However, it is virtually impossible to efficiently create a unique ID all MPI processes agree upon in an environment with an arbitrary number of MPI processes and several million rigid bodies. This would result in an all-to-all communication for every single rigid body that is created. In our framework, this ID is created locally as a combination of the rank of the local MPI process and the number of locally created rigid bodies. This approach guarantees that every rigid body is assigned a unique ID within the whole rigid body simulation. The downside of this approach is that every rigid body may only be created on exactly one process, since there is no communication between processes during the setup of a rigid body

[1] This assumption is only valid in case the domains of the MPI processes don't overlap. However, this can be safely considered to be the usual rule.

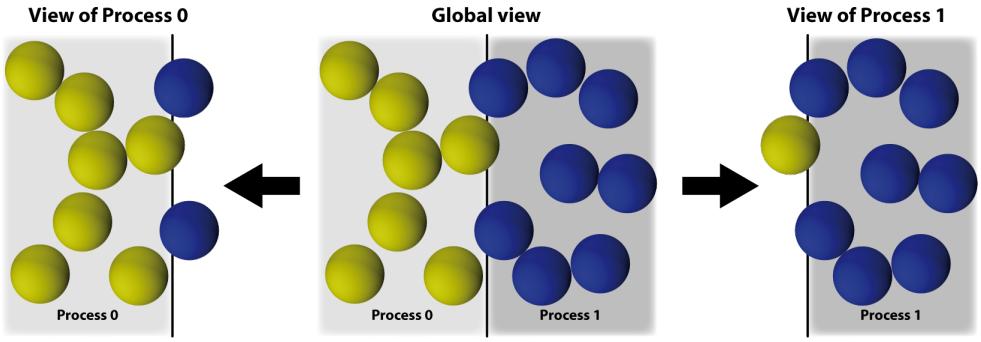


Figure 3: Setup of a simulation of two MPI processes. Both processes only know about the boundary to the neighboring process, but nothing about the domains these remote processes span. Rigid bodies are exclusively managed by the process their reference point (in our case the center of mass) belongs to. 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.

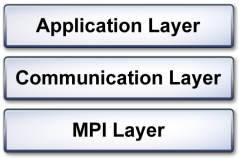


Figure 4: The communication layer is a wrapper around the MPI layer to offer rigid body related communication via MPI. It offers the functionality to send messages of arbitrary size containing heterogenous data types and heterogenous geometry types across an MPI channel. Based on its functionality, the application layer implements the parallel rigid body algorithm and knows which data has to be sent at which time.

[2]. In order to make all rigid bodies visible on all processes they are contained in, the local simulation domains are synchronized after the setup of all rigid bodies is complete.

4 The MPI Communication

Even more basic than the question of when to send which data (which is answered in Sec. 5) is the question of how to send data from one MPI process to another. In order to parallelize a rigid body simulation, it is necessary to handle several fundamental problems concerning the MPI communication itself. These problems, including heterogeneous data types, heterogeneous geometry types and arbitrarily sized messages, are implemented in the communication layer of the *pe* rigid body framework (see Fig. 4).

The first problem stems from the fundamental performance rule that the number of messages in a MPI communication should be kept as small as possible, since every MPI message involves a fixed, considerable latency. Therefore one goal for an efficient parallelization has to be to send a single MPI message to a remote process in every communication step. However, this results in the requirement to send different data types (i.e. integral data and floating point data) at the same time. The solution to this problem in our framework is the definition of a fixed message layout and to send all necessary data as raw bytes to the remote processes. Each remote process is now responsible to convert the bytes back to their original data types which is only possible due to the fixed message layout.

A second problem are the heterogeneous geometry types (as for instance spheres, boxes, or arbitrary triangle meshes). Different geometry types involve a different number of values to be sent. For example, for the geometric description of a sphere primitive, it is sufficient to send only the global position and the radius of the sphere. For a box primitive, the position, its orientation and the lengths of the three sides have to be sent. This problem can be easily solved by starting every data segment in a message with a particular message tag. Depending on this tag, the receiving process knows the content, size and structure of the following bytes and is able to convert them back to their original types and values.

[2]Note that a communication during the setup of a rigid body specifically between the processes the rigid body is created in could very easily result in a deadlock situation. Therefore this is also not considered a valid option.

The third problem is the variable size of a MPI message in a rigid body simulation. The number of rigid bodies that have to be sent to a neighboring process is only depending on the current position and orientation of the bodies. In combination with the heterogeneous data types and the heterogeneous geometry types, the size of a particular message is unknown for the receiving process. It is even possible that a message contains no data, i.e. the message size is zero (see for example Fig. 6(a)). However, since it is not possible to tell beforehand whether a message has to be sent or not, the receiving process will always expect a message. Therefore it is not possible to omit empty messages. In order to receive messages of arbitrary size, the receiving process has to acquire the size of a message before receiving it. This is performed by the `MPI_Probe()` function with a subsequent call to the `MPI_Get_count()` function.

For every communication step during the parallel rigid body simulation, the send buffers for the neighboring processes have to be filled individually with the according informations. This encoding step is succeeded by the message exchange via MPI, which is then followed by the decoding phase of the received messages. The complete MPI communication of the *pe* engine is shown in Alg. 1. The first step during communication is the initialization of the transfer of the according byte-encoded messages to the neighboring MPI processes via the non-blocking message passing function `MPI_Isend()`. While the MPI system handles the send operations, the receive operations are initiated. Since the size of the messages is unknown, the `MPI_Probe()` and `MPI_Get_count()` functions are used to test any incoming message. After that, the size of the receive buffer is adjusted and the blocking message passing function `MPI_Recv()` is used to receive the message. After all messages have been received, it is necessary to wait for all non-blocking send operations to finish. Only then the send buffers can be cleared for the next communication step.

Algorithm 1: MPI communication

```
1  for each neighboring process do
2  |  Encode the message to the neighboring process
3  |  Start a non-blocking send operation of the according byte encoded message via MPI_Isend()
4  end
5  for i from 0 to the number of neighboring processes-1 do
6  |  Check for any incoming message via MPI_Probe()
7  |  Acquire the total size of the message via MPI_Get_count()
8  |  Adjust the buffer size of the receiving buffer
9  |  Receive the message via MPI_Recv()
10 |  Decode the received message
11 end
12 Wait until all non-blocking send operations are completed
13 Clear the send buffers
```

5 The Parallel Rigid Body Algorithm

The parallel rigid body dynamics algorithm presented in this section is based on the fast frictional dynamics (FFD) solver first published by Kaufman et al. [KEP05] and improved by Wengenroth et al. [Wen07]. Due to its formulation, this algorithm belongs to the group of fast, real time collision response algorithms. For a detailed explanation of the nonparallel version of this algorithm, please refer to either one of the two previously mentioned references. This section will primarily focus on the parallelization of this algorithm. In the *pe* engine, the FFD solver is part of the application layer, that uses the communication capabilities of the communication layer (see Fig. 4).

The FFD solver is particularly suited for a large-scale parallelization due to its strictly local collision treatment: in order to calculate a post-collision velocity for a colliding rigid body, only the states of the contacting rigid bodies are considered. Therefore it is not necessary to set up and solve a LCP in parallel as it would be necessary in case of the LCP-based collision response algorithm. Additionally, the complexity of the FFD algorithm is linearly depending on the number of rigid bodies and the number of contact points between these bodies.

Alg. 2 shows the parallel FFD algorithm. In contrast to the nonparallel algorithm, the parallel version contains a total of four MPI communication steps to handle the distributed computation (see the lines 1, 6, 21 and 34). The rest of the algorithm remains unchanged in comparison to the nonparallel formulation.

Instead of immediately starting with the first position and velocity half-step for every rigid body, the first step in every time step of the parallel algorithm is the synchronization of the external forces (Alg. 2, line 1). In case a rigid body is overlapping a process boundary, part of the external forces can be contributed by the remote processes. This scenario happens for instance in case the rigid body is immersed in a fluid flow (as illustrated in Fig. 1(b)). Part of the hydrodynamic forces are calculated by the local process, another part by the remote processes. Therefore the involved processes have to synchronize their forces and calculate a total force for every rigid body.

Algorithm 2: The Parallel FFD-Algorithm

```
1  MPI communication step 1: force synchronization
2  for each body B do
3  |  position half-step (φ_B^+)
4  |  φ⃗_B^- = velocity half-step (B)
5  end
6  MPI communication step 2: update of remote and notification of new rigid bodies
7  for each body B do
8  |  find all contacts C(B)
9  |  for each contact k ∈ C(B) do
10 |  |  calculate twist n⃗_k induced by contact normal
11 |  |  calculate constraint offset d_k
12 |  |  if constraint is violated ( B, φ⃗_B^-, n⃗_k, d_k ) then
13 |  |  |  add collision constraint n⃗_k, d_k to T(B)
14 |  |  |  for m = 1 ... SAMPLESIZE do
15 |  |  |  |  calculate twist s⃗_m induced by m^th sample ⊥ to contact normal
16 |  |  |  |  add friction constraint s⃗_m, μ_m to S(B)
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 MPI communication step 3: exchanging constraints on the rigid bodies
22 for each body B do
23 |  if B has violated constraints then
24 |  |  find post-collision velocity φ⃗_B^τ on the border of T(B) closest to φ⃗_B^-
25 |  |  r⃗_B = φ⃗_B^- - φ⃗_B^τ
26 |  |  select friction response δ⃗_B from S(B) minimizing δ⃗_B + φ⃗_B^τ
27 |  |  φ⃗_B^+ = φ⃗_B^τ + ε · r⃗_B + δ⃗_B
28 |  end
29 |  else
30 |  |  φ⃗_B^+ = velocity half-step (B)
31 |  end
32 |  position half-step (B)
33 end
34 MPI communication step 4: update of remote and notification of new rigid bodies
```

After the force synchronization, all local rigid bodies (i.e. all rigid bodies whose reference point is contained in the local process) are updated with the first position half-step and first velocity half-step. Note that only local rigid bodies are treated. Every rigid body is updated by exactly one process, i.e. by the process its center of mass is contained in. Remote rigid bodies (i.e. all rigid bodies whose reference points are not contained in the local process) are updated in the second MPI communication step (Alg. 2, line 6). This communication step involves a synchronization of the position, the orientation and the velocities of remote rigid bodies, where the process that contains the reference point of the body sends the updated values to all neighboring remote processes that only contain part of the body. Due to the position and orientation change of all rigid bodies, it may also be the case that a rigid body now newly overlaps a process boundary to a particular neighboring remote process. In this case the entire rigid body has to be sent to the remote process, additionally including information about its geometry (as for instance the radius of a sphere, the side lengths of a box, the triangle mesh of an arbitrary geometry) and its material

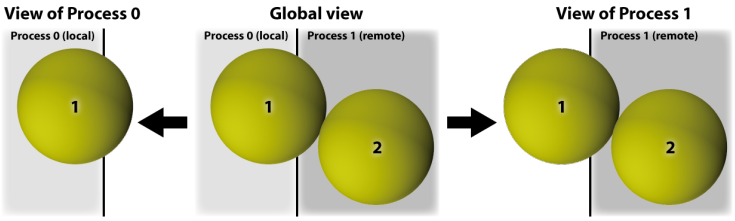


Figure 5: Some collisions can only be detected on remote processes. In the illustrated example, process 0 only knows the local sphere 1, for which collision constraints have to be calculated. Process 1, on the other hand, knows both sphere 1 and 2 (where sphere 1 is remote and 2 is local). Therefore only process 1 can setup the constraints for the collision between spheres 1 and 2 and has to send these constraints to process 0, where they are used to handle the collision between the two spheres.

Step	Purpose	After	Before
1	Force synchronization	Force calculation	1. velocity half-step
2	Update of remote and notification of new bodies	1. position half-step	Collision detection
3	Collision constraint synchronization	Constraint setup	2. velocity half-step
4	Update of remote and notification of new bodies	2. position half-step	Force calculation

Table 1: Dependencies of the MPI communication steps

(containing information about its density, the coefficient of restitution and frictional parameters). With this information, the remote process is now able to create a copy of the rigid body.

Note that, due to efficiency reasons, updates should be preferred to sending the complete rigid body. Only in case the remote process does not know the rigid body, sending the full set of information cannot be avoided. Also note, that it may be necessary to send updates for a rigid body that is no longer overlapping a particular boundary in order to give the remote process the information that the rigid body has left the domain of the remote process and can therefore be destroyed on the remote process.

The second communication step is followed by the collision detection that creates contact points for every pair of rigid bodies that is in contact. Directly afterwards, the setup of all collision constraints can be performed (see Alg. 2, line 7-20): for every contact point attached to the rigid body, a single collision constraint and several friction constraints are created in case the collision constraint is violated.

In the consecutive third communication step (Alg. 2, line 21), the MPI processes exchange collision and friction constraints among each other such that every process knows all active constraints on all local rigid bodies. This exchange is necessary since some constraints can only be detected on remote processes, as illustrated in Fig. 5. These constraints can now be used to treat all collisions and to update the velocities of the colliding rigid bodies. In case a rigid body is not involved in a collision, the velocity is updated in a second velocity half-step. After the velocity update (either due to collision handling or a velocity half-step), the second position half step is performed. In order to cope with this second change of the position and orientation of the rigid bodies, the fourth communication step (Alg. 2, line 34) performs the same synchronization as the second communication step. After this communication, all rigid bodies on all processes are synchronized and in a consistent state and the time step is finished.

Fig. 6 gives an example of a single moving rigid body that crosses the boundary between two processes. The example explains in detail, when the rigid body has to be sent by which process and when the responsibility for a rigid body is transferred to another process.

As discussed, the parallel version of the FFD algorithm contains four distinct MPI communications. We believe that this is the minimal number of MPI communications in case we don't want to change the fact that in-between two time steps the user may manipulate rigid bodies by for instance adding external forces to the bodies. Tab. 1 lists the dependencies of the individual communication steps from the steps in the non-parallel algorithm.

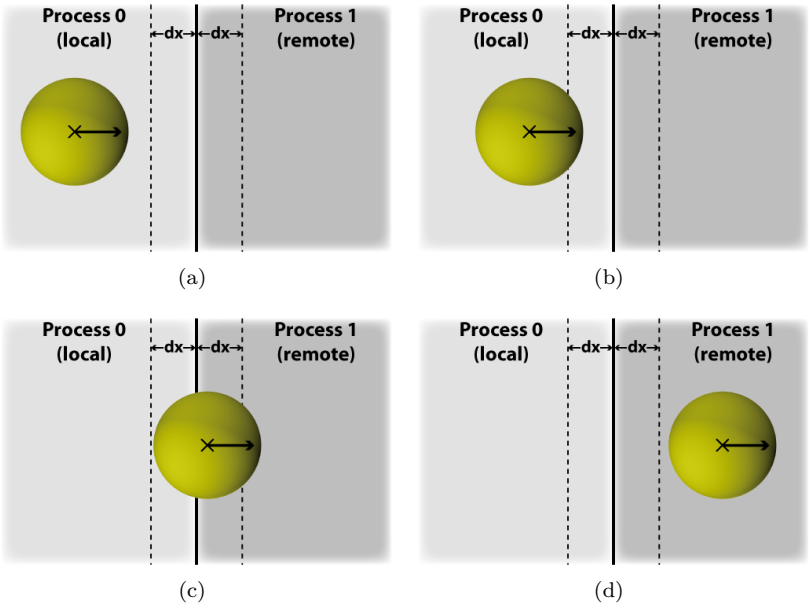


Figure 6: Example of the communication of a single rigid body. The reference point of the rigid body is indicated by the cross in its center, its linear velocity by the arrow. In Fig. 6(a), the rigid body is solely contained in process 0. Process 1 does not know the rigid body. In Fig. 6(b), the rigid body has moved into the overlap region between the two processes and therefore has to be sent to process 1. Only in the first communication, the entire rigid body, including information about its geometry and its material have to be sent. In subsequent communications, it is sufficient to send updates for the body. In Fig. 6(c), the rigid body's reference point is no longer contained in process 0, but has moved to process 1. Therefore process 1 now considers the rigid body as local, process 0 only as remote. In Fig. 6(d), the rigid body has left the overlap region between the two processes and therefore doesn't have to be communicated anymore. Therefore process 0 no longer contains any part of the body, so the body is locally destroyed. Note that the choice of the size of dx is arbitrary, but larger than the minimum distance between two colliding bodies.

6 Results

In this section, the scaling behavior of the parallel FFD algorithm is analyzed closely. The machine we are using for our scaling experiments is the HLRB-II supercomputer (SGI Altix 4700 platform) at the Leibnitz computing center Munich (LRZ) [Homa]. Due to the large number of cores (9 728) and its fast network, this machine is particularly suited for large-scale parallelizations. The HLRB-II features the following properties:

- 9728 Intel Itanium2 Montecito Dual Core cores running at 1.6 GHz, 9 MB Level 3 Cache per core, and 4 GByte of RAM per core
- 39 TByte of total main memory
- NUMAlink 4 interconnect fabric with 6.4 GByte/s bandwidth of one link (bidirectional)
- Overall peak performance of 62.3 TFlop/s (LINPACK result: 56.5 TFlop/s)

We are using up to 9 120 cores in order to simulate up to 1.14 billion rigid bodies in a single simulation. All executables are compiled with the Intel 10.1 compiler (using the optimization flag `-O3`) and we are using SGI MPI for ccNUMA Altix systems.

The scenario we have chosen for our scaling experiments is illustrated in Fig. 7: we are simulating the movement of spherical particles in an evacuated box-shaped volume without external forces. For this we create a specified number of uniformly distributed spheres, each with a random initial velocity. Although this scenario only demonstrates a fraction of the capabilities of the framework, it is well suited for our scaling experiments, since it guarantees an equally distributed load even without load balancing strategies and similar communication costs on all involved processes.

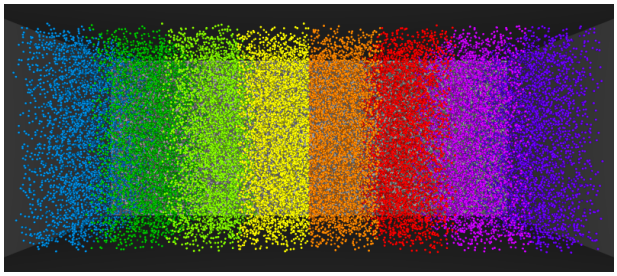


Figure 7: Simulation scenario for the scaling experiments: the specified number of spherical rigid bodies are moving randomly in a closed, evacuated, box-shaped domain without external forces.

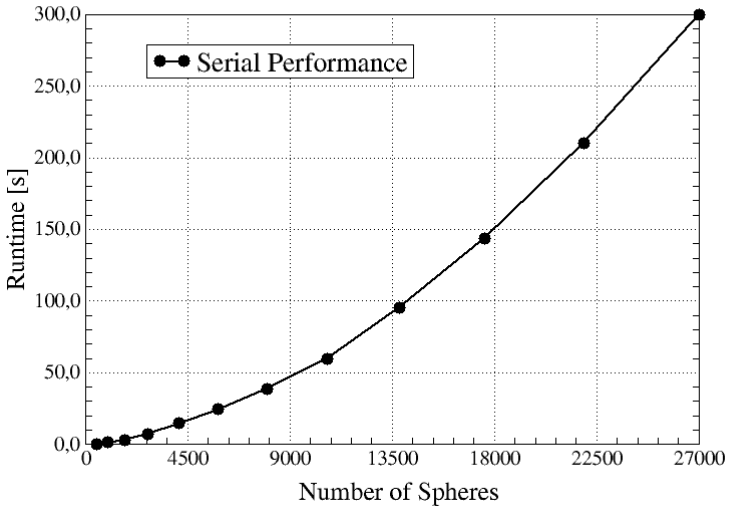


Figure 8: Serial performance of 1 000 time steps in the scenario for the strong and weak scaling experiments.

In order to be able to analyze the parallel performance of the algorithm, it is first necessary to investigate the serial performance of the rigid body framework. Fig. 8 shows a serial scaling experiment with up to 27 000 spheres in the scenario described above. Apparently the rigid body simulation exhibits a scaling behavior nonlinear with the number of bodies. One of the main reasons for this behavior is the lack of perfectly linear algorithms for some phases of the algorithm, as for instance the contact detection phase. In our simulation, we are using the sweep-and-prune algorithm as described in [ESH05] for the coarse collision detection. Although this is one of the fastest available algorithms, it doesn't offer a perfectly linear scaling behavior.

Table 2 contains the results for a strong scaling experiment with 262 144 spheres initially arranged in a $128 \times 128 \times 16$ grid. In this experiment, the spheres use approximately 1% of the total volume of the domain. Two results can be observed from the strong scaling experiment. The first observation is that doubling the number of CPU cores improves the performance of the simulation beyond a factor of two. This can easily be explained by the serial scaling results: dividing the number of rigid bodies per MPI process by two yields more than a factor of two in performance. The second observation is that the reduction of communication overhead by a smaller cut area improves the performance considerably.

Weak scaling in a rigid body simulation proves to be more complicated than strong scaling. In contrast to, for instance, grid-based simulations, where increasing the number of unknowns usually just means an increase of the resolution of the problem, increasing the number of rigid bodies inevitably changes the simulated scenario due to additional interactions. Weak scaling of a rigid body simulation also raises the question of where to create the new bodies. Therefore we chose to use weak scaling to analyze the performance influence of the number of communication neighbors and to demonstrate simulations with over a billion rigid bodies.

# Cores	# Spheres per core	Partitioning	Runtime [s]
16	16 386	16 × 1	838.2
		8 × 2	604.2
		4 × 4	543.7
32	8 192	32 × 1	320.2
		16 × 2	221.8
		8 × 4	193.8
64	4 096	64 × 1	154.8
		32 × 2	102.3
		16 × 4	77.1
		8 × 8	72.0
128	2 048	128 × 1	82.15
		64 × 2	48.09
		32 × 4	32.71
		16 × 8	25.94
256	1 024	128 × 2	24.01
		64 × 4	14.89
		32 × 8	10.89
		16 × 16	9.82
512	512	128 × 4	8.41
		64 × 8	6.18
		32 × 16	5.25

Table 2: Strong scaling experiment of 1 000 time steps with 262 144 spheres initially arranged in a 128 × 128 × 16 grid. The total volume of the spheres occupies approximately 1% of the domain volume.

In our weak scaling experiments, each process owns a cubic subdomain of initially 25 × 25 × 25 or 50 × 50 × 50 spheres, respectively. These blocks of spheres are now put together depending on the arrangement of the processes. Note that, due to this setup, the weak scaling experiments cannot be directly compared to the strong scaling experiments: whereas in both strong scaling experiments the shape of the domain was fixed and the shape of the subdomains was adapted according to the arrangement of the processes, in the weak scaling experiments the shape of a subdomain is fixed and the shape of the domain results from the arrangement of the processes.

The weak scaling experiments of our framework are shown in Table 3 and 4. The first observation is that the scaling behavior clearly depends on the number of communication neighbors: when choosing a particular number of processes, a one-dimensional partitioning is clearly the fastest, whereas a two-dimensional partitioning increases the maximum number of communication neighbors from two to eight. In case a three-dimensional partitioning is chosen, the maximum number of communication neighbors even increases to 26. Note that this results primarily from the choice of cubic blocks of spheres per process, which directly leads to an increased communication overhead in case of more neighbors, since for every neighboring process, messages have to be individually encoded and decoded. The second observation is that different partitionings with a similar communication overhead result in approximately the same runtime. Considering the fact that every single measurement represents a different scenario, this is a result that demonstrates the generally good scaling behavior of the MPI parallelization of the *pe* rigid body engine.

Our largest simulation with 9 120 cores deserves some special attention. Although this simulation does not yet show the limits of the *pe* framework, it demonstrates that it is possible to simulate more than one billion interacting rigid bodies in reasonable runtimes.

7 Conclusion

In this paper we have presented a novel approach to large-scale rigid body simulations. With the algorithm presented, we are able to simulate more than one billion rigid bodies on an arbitrary number of processes. This increase in the number of rigid bodies in a single simulation by several magnitudes in comparison to other algorithms enables for the first time the simulation of realistic rigid body scenarios such as granular media, fluidization processes or sedimentation processes. The obvious advantage in comparison to other

large-scale, particle-based simulation methods, for example molecular dynamics, the discrete element method, or smooth particle hydrodynamics, is that the geometry of the rigid bodies is fully resolved. Therefore it is possible to simulate these scenarios with arbitrarily shaped rigid bodies.

We have demonstrated the good scaling behavior of our algorithm. However, in this paper we have only used spheres, and we have not used any load balancing extension. Instead, we have focused on the analysis of the performance with fixed process boundaries. Further performance experiments should consider load balancing and investigate the influence of different geometries on the performance. Future work will include experiments on a larger number of cores and rigid body simulations of physically relevant scenarios.

References

[Ani06] M. Anitescu, *Optimization-based simulation of nonsmooth rigid multibody dynamics*, Math. Program. **105** (2006), no. 1, 113–143.

[CPS92] R.W. Cottle, J.S. Pang, and R.E. Stone, *The linear complementarity problem*, Academic Press, Inc., 1992.

[Ebe03] D. Eberly, *Game physics*, Series in Interactive 3D Technology, Morgan Kaufmann, 2003.

[ESH05] K. Erleben, J. Sporring, and K. Henriksen, *Physics-based animation*, Delmar, 2005.

[FE07] F. Fleissner and P. Eberhard, *Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection*, International Journal for Numerical Methods in Engineering **74** (2007), 531–553.

[GKZ08] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical simulation in molecular dynamics*, Springer, 2008.

[GSL99] W. Gropp, A. Skjellum, and E. Lusk, *Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1999.

[Homa] Homepage of the Leibnitz Computing Center Munich: , *http://www.lrz-muenchen.de/services/compute/hlrb/hardware/hardware.html*.

[Homb] Homepage of the Open Dynamics Engine (ODE):, *http://www.ode.org/*.

[Homc] Homepage of the OpenTissue simulation framework:, *http://www.opentissue.org*.

[ITR08] K. Iglberger, N. Thürey, and U. Rüde, *Simulation of Moving Particles in 3D with the Lattice Boltzmann Method*, Computers & Mathematics with Applications **55** (2008), no. 7, 1461–1468.

[Jea99] M. Jean, *The non-smooth contact dynamics method*, Computer Methods in Applied Mechanics and Engineering **177** (1999), no. 3–4, 235–257.

[KEP05] D. M. Kaufman, T. Edmunds, and D. K. Pai, *Fast frictional dynamics for rigid bodies*, ACM Transactions on Graphics (SIGGRAPH 2005) **24** (2005), 946–956.

[Mil07] I. Millington, *Game physics engine development*, Series in Interactive 3D Technology, Morgan Kaufmann, 2007.

[Pre08] T. Preclik, *Iterative rigid multibody dynamics*, Diploma thesis, Friedrich-Alexander University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2008, Computer Science Department 10 (System Simulation), University of Erlangen-Nuermberg.

[RA05] M. Renouf and P. Alart, *Conjugate gradient type algorithms for frictional multi-contact problems: applications to granular materials*, Computer Methods in Applied Mechanics Engineering **194** (2005), 2019–2041.

[TNA08] A. Tasora, D. Negrut, and M. Anitescu, *Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit*, Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics **222 (4)** (2008), 315–326.

[Wen07] H. Wengenroth, *Rigid body collisions*, Master's thesis, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2007, Computer Science Department 10 (System Simulation), University of Erlangen-Nuermberg.

# Cores	# Spheres	Partitioning	Runtime [s]
128	2 000 000	128 × 1 × 1	266.1
		64 × 2 × 1	361.0
		32 × 4 × 1	390.2
		16 × 8 × 1	392.0
		8 × 4 × 4	545.3
256	4 000 000	256 × 1 × 1	269.5
		128 × 2 × 1	366.4
		64 × 4 × 1	394.4
		32 × 8 × 1	392.7
		16 × 16 × 1	393.0
		8 × 8 × 4	547.5
448	7 000 000	448 × 1 × 1	265.3
		224 × 2 × 1	362.7
		112 × 4 × 1	396.4
		56 × 8 × 1	397.3
		28 × 16 × 1	391.9
		8 × 8 × 7	565.8
960	15 000 000	960 × 1 × 1	273.5
		480 × 2 × 1	369.0
		240 × 4 × 1	397.7
		120 × 8 × 1	402.5
		60 × 16 × 1	406.5
		32 × 30 × 1	402.1
		15 × 8 × 8	560.4
1920	30 000 000	1920 × 1 × 1	268.9
		960 × 2 × 1	367.4
		480 × 4 × 1	395.9
		240 × 8 × 1	406.4
		120 × 16 × 1	403.3
		60 × 32 × 1	415.8
		16 × 15 × 8	585.5
3840	60 000 000	3840 × 1 × 1	267.8
		1920 × 2 × 1	366.7
		960 × 4 × 1	399.5
		480 × 8 × 1	410.7
		240 × 16 × 1	408.9
		120 × 32 × 1	434.8
		64 × 60 × 1	432.1
		16 × 16 × 15	611.5
7680	120 000 000	7680 × 1 × 1	263.8
		3840 × 2 × 1	368.0
		1920 × 4 × 1	418.5
		960 × 8 × 1	425.8
		480 × 16 × 1	434.3
		240 × 32 × 1	432.4
		120 × 64 × 1	432.7
		32 × 16 × 15	599.0
9120	142 500 000	9120 × 1 × 1	264.9
		4560 × 2 × 1	363.8
		2280 × 4 × 1	410.4
		1140 × 8 × 1	422.5
		570 × 16 × 1	429.7
		285 × 32 × 1	430.5
		96 × 95 × 1	432.7
		24 × 20 × 19	600.3

Table 3: Weak scaling experiment of 1 000 time steps with up to 142 500 000 spheres. Each process initially owns a block of 25 × 25 × 25 spheres. The total volume of the spheres occupies approximately 1% of the domain volume.

# Cores	# Spheres	Partitioning	Runtime [s]
128	16 000 000	128 × 1 × 1	1059.73
		64 × 2 × 1	1101.53
		32 × 4 × 1	1148.56
		16 × 8 × 1	1139.79
		8 × 4 × 4	1400.87
256	32 000 000	256 × 1 × 1	1056.23
		128 × 2 × 1	1090.08
		64 × 4 × 1	1146.74
		32 × 8 × 1	1147.49
		16 × 16 × 1	1145.34
		8 × 8 × 4	1406.18
448	56 000 000	448 × 1 × 1	1049.47
		224 × 2 × 1	1094.19
		112 × 4 × 1	1142.32
		56 × 8 × 1	1142.36
		28 × 16 × 1	1141.81
		8 × 8 × 7	1405.81
960	120 000 000	960 × 1 × 1	1051.76
		480 × 2 × 1	1094.79
		240 × 4 × 1	1142.54
		120 × 8 × 1	1158.53
		60 × 16 × 1	1156.95
		32 × 30 × 1	1148.84
		15 × 8 × 8	1408.58
1920	240 000 000	1920 × 1 × 1	1065.06
		960 × 2 × 1	1111.4
		480 × 4 × 1	1149.23
		240 × 8 × 1	1149.17
		120 × 16 × 1	1152.86
		60 × 32 × 1	1157.27
		16 × 15 × 8	1414.46
3840	480 000 000	3840 × 1 × 1	1055.39
		1920 × 2 × 1	1098.88
		960 × 4 × 1	1154.26
		480 × 8 × 1	1167.33
		240 × 16 × 1	1153.03
		120 × 32 × 1	1154.49
		64 × 60 × 1	1157.28
		16 × 16 × 15	1418.86
9120	1 140 000 000	9120 × 1 × 1	1066.73
		4560 × 2 × 1	1121.13
		2280 × 4 × 1	1255.01
		1140 × 8 × 1	1304.21
		570 × 16 × 1	1326.71
		285 × 32 × 1	1333.48
		96 × 95 × 1	1359.29
		24 × 20 × 19	1613.73

Table 4: Weak scaling e×periment of 1 000 time steps with up to 1 140 000 000 spheres. Each process initially owns a block of 50×50×50 spheres. The total volume of the spheres occupies appro×imately 1% of the domain volume.