Original published: M. Hofmann and G. Rünger. Efficient data redistribution methods for coupled parallel particle codes. In *Proceedings of the 42nd International Conference on Parallel Processing (ICPP-2013)*, pages 40–49. IEEE, 2013. Online available at http://dx.doi.org/10.1109/ICPP.2013.13.

Efficient Data Redistribution Methods for Coupled Parallel Particle Codes*

Michael Hofmann and Gudula Rünger
Department of Computer Science
Chemnitz University of Technology
Email: {mhofma,ruenger}@cs.tu-chemnitz.de

***Abstract*—This article investigates two particle data redistribution methods for the coupling of application-independent solvers for long range interactions with a particle dynamics simulation. The solvers rely on their own particle data processing techniques and domain decomposition schemes, but are implemented within a single parallel software library with a unique interface. Thus, efficient data reordering and redistribution methods are required for the data transfer between an application and the library interface as well as between the library interface and a solver. The first particle data redistribution method hides all data reordering and redistribution inside the library and restores the original particle order and distribution. The second method uses the solver-specific particle order and distribution for the particle dynamics simulation to reduce the communication amount of particle redistribution steps. Both methods and their effects on the different program components of a parallel particle code are described. Performance results on the parallel computing cluster JuRoPA and the IBM Blue Gene/Q system Juqueen are presented to demonstrate the performance improvements achieved within a particle dynamics simulation application[1].**

***Keywords*-particle simulation; data redistribution; performance optimization;**

I. Introduction

Particle simulations are popular approaches for solving complex physical problems in areas, such as solid-state physics, fluid dynamics, biochemistry, or astrophysics [1]. Particle dynamics simulations according to classical mechanics are usually performed by a numerical integration of the equations of motion. Within these simulations, the force on each particle is calculated at discrete time steps in order to obtain the changes of the particle velocities. Major computational costs are often caused by force calculations resulting from long range interactions, such as Coulomb or gravitational interactions. These calculations need to compute the pairwise interactions between all particles, thus leading to a computational complexity of $O(n^2)$ for n particles.

To cope with the large computational costs for large particle systems, methods with lower computational complexity have been developed. This includes tree-based methods, such as the Fast Multipole Method (FMM) [2] or the Barnes-Hut algorithm [3], as well as grid-based methods, such as Particle-Particle-Particle Mesh (P3M) [4] or fast summations based on nonequispaced fast Fourier transforms (P2NFFT) [5]. The ScaFaCoS project [6] provides a parallel software library with efficient implementations of fast solver methods for long range interactions with an emphasis on high scaling parallel platforms. The solver methods are implemented independently from the specific application areas of particle simulations and can be used by various particle simulation codes. This model enables particle simulation codes that combine the efficient calculation of long range interactions with further individual program components, such as specific numerical integration methods, additional short range interactions, or specialized pre- or post-processing of the particle data.

The ScaFaCoS library integrates different solver methods and provides a unique programming interface. Newly developed implementations of solver methods as well as existing codes are included. All solver methods rely on their own particle data processing techniques and domain decomposition schemes on distributed memory parallel computers. Thus, coupling the different program components usually requires dedicated data transformation and redistribution steps. These data handling steps mainly concern the data transfer between a specific particle simulation code and the library interface as well as between the library interface and a specific solver method. Especially on high scaling parallel platforms, the necessary data redistribution methods have to be designed carefully and optimized to minimize performance overheads.

This article investigates the data reordering and redistribution methods required for coupling independent program components into a particle simulation code for distributed memory parallel computers. Two solver methods for long range interactions from the ScaFaCoS library are used: The tree-based method FMM utilizes a domain decomposition scheme based on space filling curves and uses a parallel sorting method to redistribute the particles among parallel processes. The grid-based method P2NFFT utilizes a uniform division of the particle system into subdomains for a Cartesian grid of processes. Both solver methods are coupled through the library interface with a particle application code that performs a particle dynamics simulation with several time steps. Two methods are presented to cope with the varying particle order and distribution caused by the solver methods:

A) The original (application-specific) order and distribution of the particles is retained during the simulation. The solver methods work with copies of the input data (i.e.,

*This work is funded by the Deutsche Forschungsgemeinschaft (DFG) within the Cluster of Excellence MERGE.

[1]The measurements were performed at the John von Neumann Institute for Computing, Forschungszentrum Jülich, Germany.

position and charge values of particles) and the result data of the solver methods (i.e., calculated potential and field values resulting from the long range interactions) is restored to the original particle order and distribution.

B) The changed (solver-specific) order and distribution of the particles is used during the simulation. Each time a solver method is executed within a time step, the input data (i.e., position and charge values) and the result data (i.e., potential and field values) are returned to the application with the changed particle order and distribution.

The method A represents the original approach supported by the ScaFaCoS library. This method eases the coupling of a solver method with existing particle codes, but can lead to expensive data redistribution steps in each time step. The method B represents a newly developed approach that tries to avoid these expensive data redistribution steps for particle dynamics simulations. However, particle simulation codes often contain additional application-specific particle data (e.g., velocity and field values from previous time steps) that is not used by the solver. In each time step, this additional particle data has to be reordered and redistributed to adapt to the changed particle order and distribution of the solver.

The domain decomposition schemes used by the solver methods distribute the particles according to their positions among parallel processes executed on a distributed memory parallel computer. Within a particle dynamics simulations, the particle positions usually change only slightly from one time step to the next. The method B exploits this fact for an efficient particle redistribution by retaining the solver-specific distribution of the particles during the simulation. Thus, the slightly changing particle positions lead to data redistribution steps that involve only a few particles. Furthermore, the limited movement of the particles is exploited to reduce the number of processes communicating with each other.

The rest of this article is organized as follows: Section II introduces the ScaFaCoS library together with the solver methods FMM and P2NFFT as well as their coupling into a parallel particle code. Section III describes methods to cope with the changed particle order and distribution during the simulation as well as to exploit the limited particle movement. Section IV shows performance results and Sect. V concludes the article.

II. Parallel Particle Simulations with the ScaFaCoS Library

The ScaFaCoS project aims to provide a parallel software library assembling efficient implementations of fast solver methods for long range interactions. The library is intended to be used in various applications that require the efficient computation of long range interactions. In this section, the interface and usage of the ScaFaCoS library is explained. Two solver methods (i.e., FMM and P2NFFT) implemented in the library are described, focusing on the particle data reordering and redistribution required for distributed memory parallel computers. Finally, an example application for performing a particle dynamics simulation is presented.

A. Interface and Usage of the ScaFaCoS Library

The ScaFaCoS library and the solvers contained in the library are intended for distributed memory parallel computers and are based on the Message Passing Interface (MPI) [7]. The software library uses prefix "fcs" (originally standing for Fast Coulomb Solvers) for all its public identifiers (e.g., function names, data types, constants, ...) and provides bindings for C/C++ and Fortran codes. The following descriptions refer to the C/C++ interface of the library. All solvers and library functions use the floating-point data type `fcs_float` and the integer data type `fcs_int`. Both data types can be configured during the installation of the library. A generic handle of type `FCS` is used to represent an instance of a specific solver within a particle code. The usage of the library proceeds as follows:

- The function `fcs_init` initializes a new solver instance. The specific solver method is chosen by a string parameter (e.g., "fmm" or "p2nfft"). An MPI communicator is used to specify the group of parallel processes that execute the solver.
- The properties of the particle system are set with the function `fcs_set_common`. The parameters include the periodicity of the system as well as the shape of the three-dimensional system box specified by an offset vector and three base vectors. Further general or solver-specific parameters can be set with dedicated setter functions.
- An optional tuning step can be performed with the function `fcs_tune` to determine solver-specific parameters before the actual computation of the interactions. The tuning step requires the specification of the current particle positions and charges, but the tuning results are expected to remain valid as long as the positions of the particles do not change "too much".
- The computation of the long range interactions is performed with the function `fcs_run`. The function parameters include two arrays for providing the current particle positions and charges as well as two arrays for storing the calculated potential and field values. Executing the function `fcs_run` is intended to be performed repeatedly, for example, in the simulation loop of a particle dynamics simulation.
- The function `fcs_destroy` is used to release a solver instance and its allocated resources.

Figure 1 shows an illustration for the usage of the ScaFaCoS library within a particle dynamics simulation application. The initialization of the solver and the setup of the particle system properties have to be performed identically by all parallel processes. The particles can be provided in any kind of distribution among the parallel processes. For executing `fcs_tune` and `fcs_run`, each process has to specify only the position and charge values of its local particles, the number of the local particles, and the maximum number of particles that can be stored in the local particle data arrays.

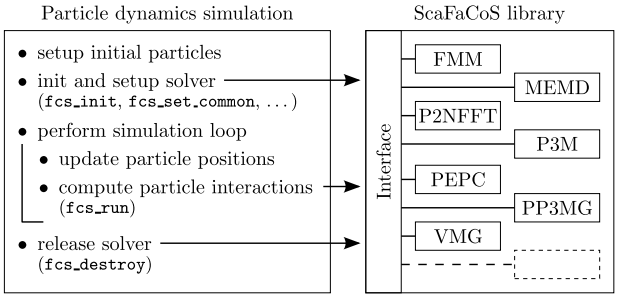

Figure 1. Illustration of the usage of the ScaFaCoS library within a particle dynamics simulation application.

B. FMM Solver

The FMM solver represents a parallel implementation of the Fast Multipole Method [2]. This tree-based algorithm uses a recursive subdivision of the system box of particles into smaller boxes. Contributions from interactions between particles inside each box and between neighboring boxes belong to the near field and are calculated by considering all pairs of particles. All other contributions belong to the far field and are approximated with multipole expansions. Using these expansions allows for a hierarchical grouping of the contributions and enables efficient operations for calculating interactions between entire boxes of particles at once. The level up to which the subdivision into boxes is performed controls the computational costs of the near and far field calculations. The FMM solver optimizes the subdivision into boxes and the expansion length in order to achieve a given accuracy for the results with minimum runtime [8].

The boxes resulting from the subdivision are numbered according to a Z-morton ordering [9]. Each particle is assigned a box number according to the box it is located in. The particles are placed into the boxes by sorting all particles (i.e., the position and charge values) according to their box numbers. The parallel implementation of the FMM solver uses a parallel sorting method to place all particles into their corresponding boxes. This approach leads to a domain decomposition of the particle system where each process is responsible for a set of particles that corresponds to a segment of a Z-order curve. Figure 2 (left) shows an example of this domain decomposition scheme for a two-dimensional particle system and four processes. The parallel sorting of the particles causes a reordering and redistribution of the particles. Thus, the potential and field values calculated by the FMM solver do not correspond to the original order and distribution.

C. P2NFFT Solver

The P2NFFT method is similar to Ewald-like particle-mesh algorithms [10], that split the calculation of the interactions into near field contributions from the real space and far field contributions from the Fourier space. Nonequispaced fast Fourier transforms are used to perform the fast computations of the Fourier space part. The calculations of the real space part require to consider all pairs of particles that are located within a given cutoff radius to each other. These computations are performed with a linked cell algorithm [11] that sorts all particles into boxes of size of the cutoff radius. A more detailed description of the P2NFFT algorithm is given in [5].

The parallel implementation of the P2NFFT method uses a domain decomposition scheme that distributes the particle system uniformly among a Cartesian process grid. Thus, the target process for each particle is calculated from its position. For the real space computations with the linked cell algorithm, each process needs access to particles located in boxes of neighboring processes within the process grid. Access to these particles is provided in the form of ghost particles. These duplicates of particles are created automatically during the particle data redistribution step for all particles that are close to the subdomain boundaries. Figure 2 (right) shows an example of this domain decomposition scheme for a two-dimensional particle system and four processes. The distribution of the particles among the process grid and the sorting of particles into boxes causes a reordering and redistribution of the particles. Thus, the potential and field values calculated by the P2NFFT solver do not correspond to the original order and distribution.

D. Integration Method for a Particle Dynamics Simulation

The example application for performing a particle dynamics simulation uses a second-order integration method based on the leapfrog approach [1]. Given the positions and velocities of all particles at time step i by vectors $\vec{x}_i$ and $\vec{v}_i$, the positions and velocities at time step $i+1$ are calculated as follows:

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_i \Delta t + \frac{1}{2}\vec{a}_i \Delta t^2 \quad (1)$$

$$\vec{v}_{i+1} = \vec{v}_i + \frac{1}{2}(\vec{a}_i + \vec{a}_{i+1})\Delta t \quad (2)$$

The initial positions $\vec{x}_0$ and velocities $\vec{v}_0$ are given by the particle system to be simulated. The accelerations $\vec{a}_i$ of all particles at time step i are determined from the field values calculated with the FMM or P2NFFT solver using particle positions $\vec{x}_i$. Parameter Δt specifies the size of each time step.

The simulation application reads the particle system from an input file and creates an initial distribution of the particles among the parallel processes. In Sect. IV-B, three different initial distributions are compared: all particles on one single process, uniformly random distribution of particles among processes, and a domain decomposition that distributes particles uniformly among a Cartesian process grid.

Figure 3 shows the pseudocode of the simulation application. To start the simulation, an instance of a solver from the ScaFaCoS library is created and its tuning step is executed (lines 2–4). The initial interactions of the particles are computed with the solver to determine the initial accelerations $\vec{a}_0$ of the particles from the calculated field values (lines 5-6). The application simulates T time steps by executing a simulation loop (line 8). In each loop iteration, the new particle positions $\vec{x}_{i+1}$ are calculated and the new interactions are computed with the solver (lines 9–10). The calculated field values are used to determine the new accelerations $\vec{a}_{i+1}$ such that they can

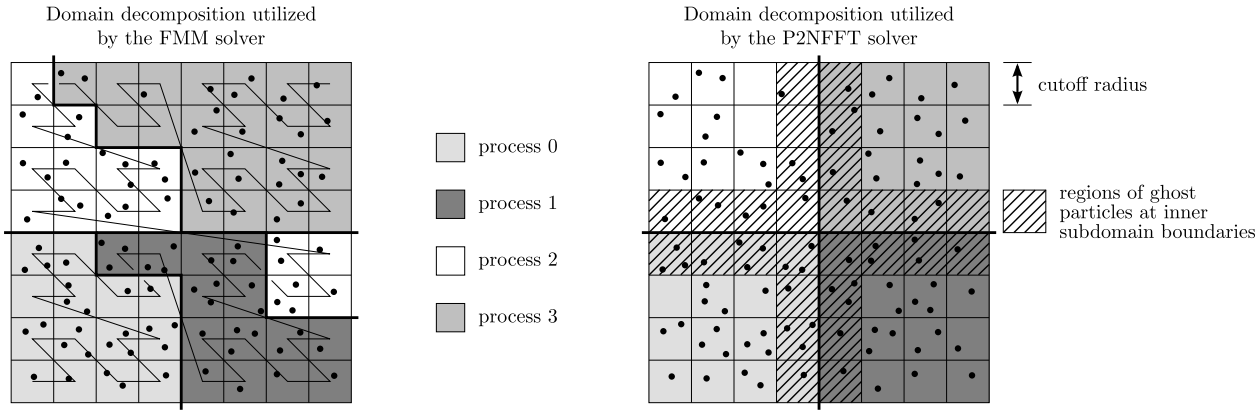

Figure 2. Domain decomposition of a two-dimensional particles system based on the Z-order curve utilized by the FMM solver (left) and on a Cartesian process grid utilized by the P2NFFT solver (right).

```
1  input: Initial positions x⃗_0 and velocities v⃗_0
2  fcs_init // init ScaFaCoS solver
3  fcs_set_common // setup particle system properties
4  fcs_tune(x⃗_0) // tune solver
5  fcs_run(x⃗_0) // compute initial interactions
6  Determine initial accel. a⃗_0 from calculated field values
7  // Execute the simulation loop
8  for i = 1 to T do
9      Calculate positions x⃗_{i+1} with Eq. (1)
10     fcs_run(x⃗_{i+1}) // compute interactions
11     Determine accel. a⃗_{i+1} from calculated field values
12     Calculate velocities v⃗_{i+1} with Eq. (2)
13 fcs_destroy // release solver
```

Figure 3. Pseudocode of the integration method used by the particle dynamics simulation application.

be used to calculate the new velocities $\vec{v}_{i+1}$ (lines 11–12). After the simulation is finished, the solver and its allocated resources are released (line 13). Including the computation of the initial interactions, the solver is executed $T + 1$ times for the simulation of T time steps.

III. Particle Data Reordering and Redistribution for Particle Dynamics Simulations

The coupling of independent program components into a particle code as described in the previous section involves the exchange of large amounts of particle data through the interfaces of the ScaFaCoS library. For each execution of a solver, the particle positions and charges have to be submitted and the calculated potential and field values have to be returned. Since program components, such as the solver methods and the calling applications, can process the particle data differently, the order and distribution of the particles has to be adjusted between these components. The required data transformation operations have to be performed efficiently to minimize their effects on the overall performance. In this section, two methods for the exchange and transformation of the particle data are presented.

A. Restoring the Original Particle Order and Distribution

The first method hides all reordering and redistribution of particle data performed by a solver method inside the ScaFaCoS library. A particle application can submit the particles with an arbitrary distribution among the parallel processes to the library and gets the calculated results back with the same distribution. This method reduces the effects on the particle data handling of the application, because its original particle order and distribution is restored. However, major drawbacks can occur for a solver method that relies on a domain decomposition scheme differing from the scheme of the application. In the following, the effects on the different program components from Sect. II (i.e., library interface, FMM and P2NFFT solver, and integration method) are described.

Library Interface: The library interface described in II-A does not require any modifications or extensions. The particle positions and charges given to the function `fcs_run` remain unchanged after the computations are finished. The order and distribution of the calculated potential and field values corresponds to the unchanged particle order and distribution. Implementing this behavior is left to the specific solvers.

FMM Solver: The FMM solver sorts the particles according to a Z-morton ordering of the boxes the particles are located in. Thus, the given particle positions and charges are changed and the calculated potential and field values correspond to the changed particle order. A consecutive numbering of the initial particles is used to preserve the information about their original order during the sorting step. After the computations of the FMM solver, the original order of the particles is restored by sorting all particles according their initial numbering. In the parallel case, a global numbering of the particles on all processes is used such that the particles of each single process are consecutively numbered. Sorting the particles into boxes is performed with a partition-based parallel sorting algorithm as described in [12]. Restoring the original order and distribution of the particles is performed by sending each particle back to

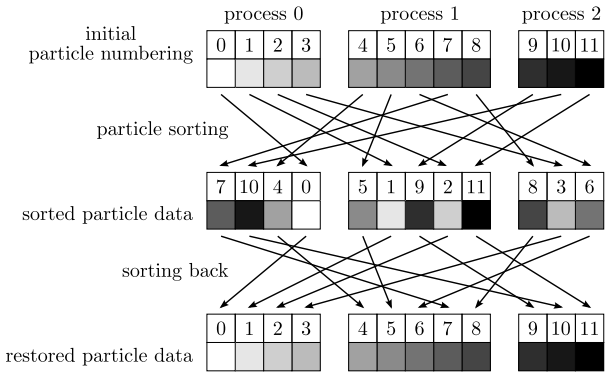

Figure 4. Example for restoring the original order and distribution of particles within the FMM solver.

its initial process and storing the particle at its initial position. Both the initial process and the initial position of each particle are given by the initial numbering. Figure 4 shows an example for restoring the original order and distribution of particles within the FMM solver.

P2NFFT Solver: The P2NFFT solver uses a domain decomposition scheme that distributes the particle system uniformly among a Cartesian process grid. The corresponding particle data redistribution step creates a copy of all particles and adds ghost particles at subdomain boundaries. Each of these particle copies contains an index value as reference to its original particle. The index values is an 64-bit integer using 32 bit to store the rank of the source process of the particle and 32 bit to store the source position of the particle on the source process. Ghost particles have an invalid index value to indicate that they are duplicates. After the computations of the P2NFFT solver, the index values are used to restore the original order and distribution for the calculated potential and field values.

Distributing the particles among the process grid as well as restoring the original distribution is performed using a so-called fine-grained data redistribution operation [13]. This operation performs an all-to-all communication operation with MPI where each particle is sent to an individual target process. The implementation of the P2NFFT solver utilizes a generalized version of the operation that uses a user-defined distribution function to specify the target processes and that supports the duplication of particles during the data redistribution [14]. Distributing the particles among the process grid uses a distribution function that calculates the target process rank and the required duplicates for ghost particles from the particle position. After this redistribution, a reordering of the particles is performed on each process for calculating the near field and far field contributions. Restoring the original particle distribution uses a distribution function that extracts the target process rank from the index value of each particle. After this redistribution, a permutation is performed using the source positions of the particles that are contained in the index values.

Integration Method: The integration method for the particle dynamics simulation described in Sect. II-D does not require modifications or extensions, because their chosen order and distribution of the particles is retained during the simulation.

B. Using the Changed Particle Order and Distribution

The positions of particles change only slightly from one time step to the next in particle dynamics simulations. In these cases, usually only a small reordering and redistribution of the particles is required. To benefit from these small changes of the particles, the changed particle order and distribution of the solver has to be returned to the application. Thus, the application has to be able to cope with these changes for their own processing of the particle data. Methods are required to adapt additional application-specific particle data, such as the current particle velocities or accelerations, to the changed particle order and distribution. Furthermore, an application can determine the maximum movement of the particles, for example, from the physical properties of the particle system or during the update of the particle positions. This information about the limited movement of the particles can be used for optimizing the particle data sorting or redistribution steps within a solver. In the following, the effects on the different program components from Sect. II (i.e., library interface, FMM and P2NFFT solver, and integration method) are described.

Library Interface: The particle positions and charges given to the function `fcs_run` can now be changed during the execution of this function. However, the redistributed particles of a solver can only be returned to the calling application if the given local particle data arrays are large enough. The information about the maximum number of particles that can be stored on each process is given to `fcs_run`. After executing `fcs_run`, a query function can be used to determine whether the particle order and distribution was changed. If the particles have changed, then the returned particle positions and charges as well as the calculated potential and field values corresponded to the changed particle order and distribution of the solver. Otherwise, the local particle data arrays of at least one process were too small and the original order and distribution had to be restored.

Usually, an application contains additional particle data that is not reordered and redistributed by the solver (e.g., particle velocities or accelerations). Thus, it is necessary to reorder and redistribute this additional particle data explicitly. To provide this functionality, each solver creates so-called resort indices that enable the subsequent reordering and redistribution of additional particle data. The resort indices are 64-bit integers that specify the target process and the target position for each original particle. The functions `fcs_resort_[floats,ints]` are provided to perform the subsequent reordering and redistribution of floating-point and integer, respectively. The implementation uses the fine-grained data redistribution operation (as described above for the P2NFFT solver) followed by a permutation according to the target positions contained in the resort indices. Creating the resort indices is left to the specific solvers.

FMM Solver: To return the changed particle order and distribution, the FMM solver omits its final step for restoring the original particle order and distribution. The resort indices

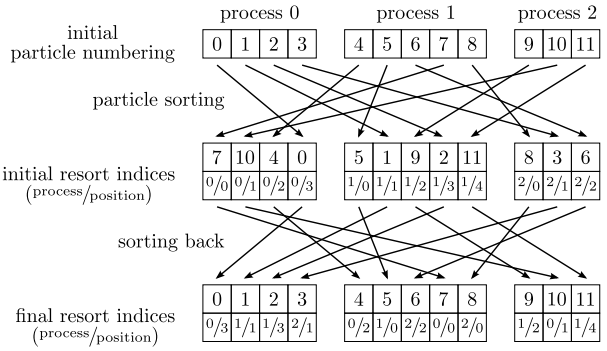

Figure 5. Example for the creation of resort indices within the FMM solver.

for the subsequent reordering and redistribution of additional particle data are created by inverting the permutation give by the initial numbering of the particles. While this numbering references the original position of each changed particle, the inverse numbering references the changed position of each original particle. The resort indices representing this inverse permutation are determined by initializing new index values consecutively for the changed particles and sorting these index values back according to the particle numbering. This sorting step is the same as the restoring of the original particle order and distribution for the FMM solver in Sect. III-A, but only performed with the resort indices. Figure 5 shows an example of the creation of the resort indices within the FMM solver.

The limited movement of the particles for the repeated execution of the FMM solver in a particle dynamics simulation leads to particles that are already almost sorted. Moreover, sorting the particles in this case causes that a majority of the particles stays on its current process or is only sent to a neighboring process. However, due to the shape of the Z-order curve, a small amount of particles may need to be sent to distant processes. To exploit these kind of almost sorted particles, a merge-based parallel sorting algorithm is used [15]. First all processes sort their local particles. After that, all processes perform pair-wise merging steps according to Batcher's Merge-Exchange sorting network [16] until all particles are sorted. This merge-based parallel sorting method uses only point-to-point communication operations while the partition-based parallel sorting method of Sect. III-A uses collective all-to-all communication operations. Switching between the parallel sorting methods for unsorted and almost sorted particles is done with a heuristic: The total volume of the particle system is divided by the number of parallel processes and it is assumed that the resulting volume per process represents a cube shaped subdomain of the particle system. If the maximum movement of the particles is less than the side length of such a cube, then the merge-based parallel sorting method is used.

P2NFFT Solver: To return the changed particle order and distribution, the P2NFFT solver omits its final step for restoring the original particle order and distribution. Instead, the created ghost particles are removed and the reordered and redistributed particle data (i.e., position, charge, potential, and field values) are copied to the given particle data arrays. The resort indices for the subsequent reordering and redistribution of additional particle data are created in the same way as described for the FMM solver: The existing index values representing a permutation that restores the original particle order and redistribution are inverted to create index values that perform the reordering and redistribution of original particles.

The limited movement of the particles for the repeated execution of the P2NFFT solver in a particle dynamics simulation usually limits the redistribution of particles to neighboring processes. In this case, the all-to-all communication used in Sect. III-A is replaced by neighborhood communication that is performed with non-blocking point-to-point communication operations. Switching between all-to-all communication and neighborhood communication is done as follows: If the maximum movement of the particles restricts their redistribution to direct neighbors within the process grid, then the neighborhood communication is used.

Integration Method: In each time step, the integration method of the example application submits its particle data to the solver and retrieves them back with a changed order and distribution. However, since the current particle velocities and accelerations are not changed by the solver, this additional particle data has to be reordered and redistributed subsequently. After executing `fcs_run`, the function `fcs_resort_floats` is used to adapt the particle velocities and accelerations to the changed particle order. This additional particle data is larger than the calculated potential and field values. Thus, in the first time step, it can be expected that there is no performance benefit in comparison to the method described in Sect. III-A. After the first time step, a significant performance benefit can be expected, because the solver-specific particle order and distribution is used, thus reducing the amount of communication.

IV. Performance Results

The particle data redistribution methods presented in Sect. III have been integrated into the ScaFaCoS library. The integration method for the particle dynamics simulation is implemented within a generic benchmark application that is part of the library. In this section, performance results are shown to demonstrate the influence of the particle data redistribution methods on the overall performance and the improvements achieved for particle dynamics simulations.

A. Experimental Setup

Two supercomputer systems called JuRoPA and Juqueen have been used to obtain the performance results. The JuRoPA system consists of 2208 compute nodes that are connected with a QDR InfiniBand network. Each compute node contains two quad-core Intel Xeon processors running at 2.93 GHz and 24 GiB main memory. The MPICH2-based ParaStation MPI library from ParTec is used. Results for the JuRoPA system were obtained executing 8 processes on each compute node.

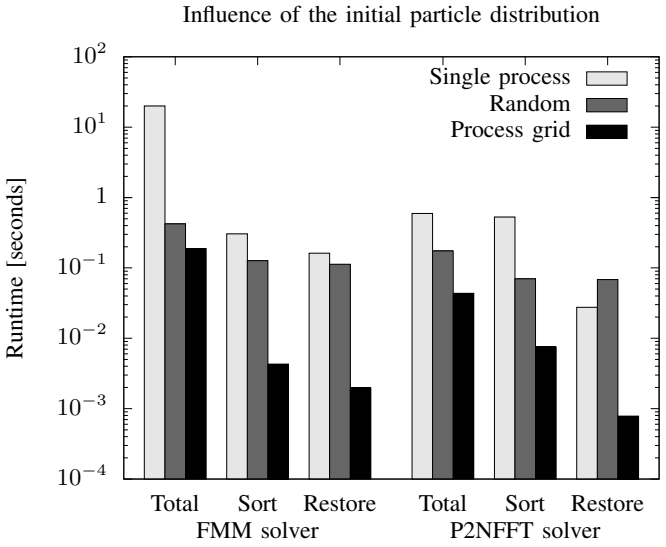

Figure 6. Total runtimes and runtimes for sorting and restoring the particles for the computation of particle interactions with the FMM solver and the P2NFFT solver using three different initial particle distributions: all particles on one single process (single process), uniformly random distribution of particles among processes (random), and a domain decomposition that distributes particles uniformly among a Cartesian process grid (process grid).

The Juqueen system is an IBM Blue Gene/Q system with 28 672 compute nodes. Each compute node consists of a PowerPC A2 1.6 GHz processor with 16 compute cores and 16 GiB main memory. An MPICH-based MPI implementation specially adapted to the interconnection network of the Blue Gene/Q platform [17] is used. Results for the Juqueen system were obtained executing 16 processes on each compute node.

A cubic particle system of size 248x248x248 with 829 440 particles and periodic boundary conditions is used for the benchmark runs. The particle system was generated by a simulation of a melting silica crystal and consists of positive and negative charged ions which are sufficiently homogeneously distributed. For the particle dynamics simulation, 1000 time steps with $\Delta t = 0.01$ and initial particle velocities $\vec{v}_0 = 0$ have been performed. The P2NFFT solver uses a fixed cutoff radius of 4.8 for the near field contributions and the FMM solver chooses its algorithmic parameters automatically in the tuning step. Both solver methods compute the long range interactions with an accuracy that leads to a relative error of less than 10^{-3} for the total energy of the system.

B. Influence of the Initial Particle Distribution

The influence of the initial particle distribution on the performance of the parallel solvers is investigated with the method A that restores the original particle order and distribution (Sect. III-A). Figure 6 shows total runtimes of the FMM solver and the P2NFFT solver as well as runtimes for sorting the particles and for restoring the original particle order and distribution. Results are obtained with 256 processes on the JuRoPA system using three different initial particle distributions: all particles on one single process (single process), uniformly random distribution of particles among processes (random), and a domain decomposition that distributes particles uniformly among a Cartesian process grid (process grid).

Storing all particles initially on a single processes leads to the largest runtimes, because this single process becomes the bottleneck for the communication. Since the FMM solver performs no further load balancing, its total runtime is also very large and corresponds to a sequential execution. The random distribution leads to a more balanced communication for sorting and restoring the particles, thus leading to smaller runtimes with a significant effect on the total runtimes. The process grid distribution decreases the runtimes for sorting and restoring the particles at least by an order of magnitude in comparison the random distribution. Since the P2NFFT solver utilizes the same distribution, the runtime is mainly caused by the creation of the ghost particles. Due to the homogeneously distributed particles in the melting silica particle system, the domain decomposition based on the Z-order curve utilized by the FMM solver differs only slightly from the process grid distribution. The results show the strong influence of the initial particle distribution on the overall performance.

C. Original vs. Changed Particle Order and Distribution

The method A that restores the initial particle order and distribution (Sect. III-A) is compared with the method B that uses the changed particle order and distribution of a solver (Sect. III-B). Exploiting the limited movement of the particles with the method B (i.e., switching to the merge-based parallel sorting for the FMM solver and to point-to-point communication with neighboring processes for the P2NFFT solver) is not used (see Sect. IV-D for these results). Results are obtained with 256 processes on the JuRoPA system.

Figure 7 shows runtimes with the FMM solver (left) and the P2NFFT solver (right) for the initial particles (line 5 in Fig. 3) and for the first eight time steps of the simulation (line 10 in Fig. 3). The initial particles are distributed uniformly random among the parallel processes. Results are shown for sorting the particles and restoring their original order and distribution (method A), for sorting the particles and resorting their velocity and acceleration values (method B), and for the total runtimes of the solvers with both methods.

For the FMM solver with the initial particles, both methods perform the same parallel sorting of the particles, thus leading to about the same results. However, resorting the particle velocities and accelerations with the method B is slightly faster than restoring the original particle order and distribution with the method A. For all following time steps, the method A achieves about the same runtimes as for the initial particles, because the initial random distribution of the particles is restored in each time step. The runtimes for sorting the particles and for resorting their velocities and accelerations with the method B decrease by about two orders of magnitude in the first time step. Thus, using the changed particle order and distribution of the FMM solver leads to a performance increase. These improvements result in significant differences in the total runtime: While the total runtime of the method A is

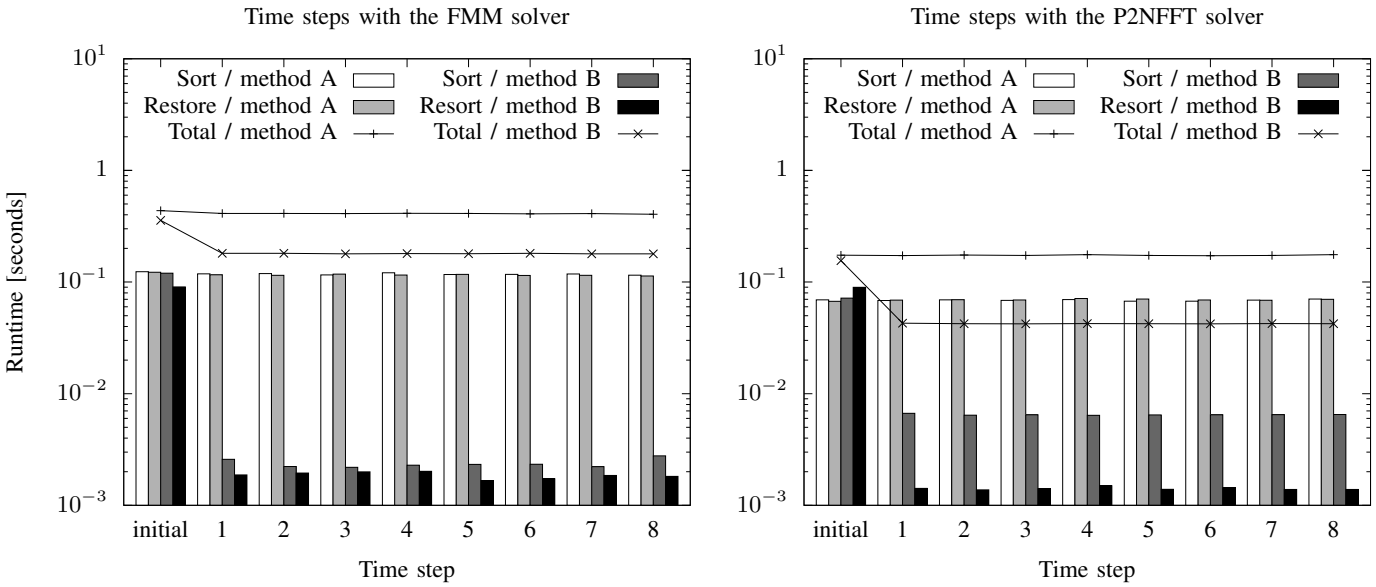

Figure 7. Runtimes from time steps of particle dynamics simulations using the FMM solver (left) and the P2NFFT solver (right) with 256 processes on the JuRoPA system. The initial particles are distributed uniformly random among the parallel processes. Total runtimes and runtimes for the particle data redistribution are shown for the method A that restores the initial particle order and distribution and the method B that uses the changed particle order and distribution of a solver.

constant over all time steps, the total runtime of the method B decreases to about 45 % in the first time step.

For the P2NFFT solver with the initial particles, sorting the particles and restoring their original order and distribution with the method A is slightly faster than resorting the particle velocities and accelerations with the method B. Apart from that, the behavior of the P2NFFT solver is similar to the FMM solver: The runtimes of the method A remain constant over all time steps and the runtimes for sorting the particles and resorting their velocities and accelerations with the method B decrease at least by an order of magnitude. These performance benefits overcompensate the slightly increased runtime with the initial particles. The total runtime of the method B decreases to about 20 % in the first time step. This improvement is larger than the improvement with the FMM solver, because the data handling of the P2NFFT solver represents a larger part of its total runtime. Nevertheless, using the solver-specific particle order and distribution for the particle dynamics simulation leads to significant performance benefits for both solvers.

Figure 8 shows runtimes for computing the particle interactions with the FMM solver (left) and the P2NFFT solver (right) for each time step of the simulation (line 10 in Fig. 3). The initial particles are distributed uniformly among a Cartesian process grid. Results are shown for sorting the particles and restoring their original order and distribution (method A), for sorting the particles and resorting their velocity and acceleration values (method B), and for the total runtimes of the solvers with both methods.

For the FMM solver until about 200 time steps, methods A and B achieve both very small runtimes for redistributing the particle data. This is the expected behavior, because the differences between the domain decomposition based on the Z-order curve and the initial process grid distribution of the particles are only small. After more than 200 time steps, the runtimes for sorting and restoring the particle order with method A start to increase. This behavior is caused by the particle movement which leads to particle distributions within the FMM solver that differ increasingly from the initial process grid distribution. Thus, sorting the particles and restoring their original distribution leads to an increasing overhead in each time step. The resulting increase of the runtime has a significant effect on the total runtime of the FMM solver. After 1000 time steps, about 50 % of the total runtime of the FMM solver in each time step is caused by the particle data redistribution. In comparison to that, method B leads to a significant performance benefit. Both, sorting the particles and resorting their velocities and accelerations leads to almost the same small runtimes for all time steps. The average runtime for the particle data redistribution causes only about 3 % of the total runtime of the FMM solver in each time step.

The results for the P2NFFT solver confirm the performance benefits achieved by redistributing the particle data according to method B. The P2NFFT solver uses the same particle distribution as the initial process grid distribution. Thus, the runtime for redistributing the particle data is only caused by the particle movement and the creation of ghost particles. However, after 300 time steps, the increasing differences between the initial process grid distribution and the process grid distribution of the moved particles lead to a significant increase of the runtimes for method A. After 1000 time steps, up to about 75 % of the total runtime of the P2NFFT solver in each time step is caused by the particle data redistribution. In comparison to that, method B prevents the increase of the runtimes during the simulation and leads to an average runtime

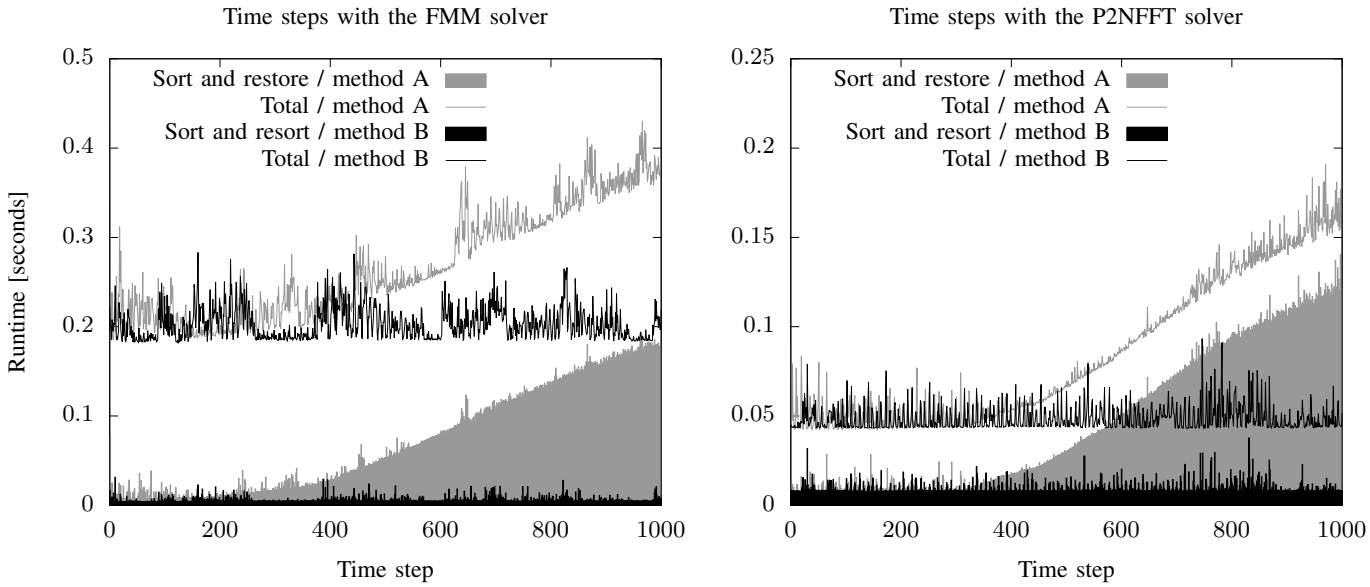

Figure 8. Runtimes from time steps of particle dynamics simulations using the FMM solver (left) and the P2NFFT solver (right) with 256 processes on the JuRoPA system. The initial particles are distributed uniformly among a Cartesian process grid. Total runtimes and runtimes for the particle data redistribution are shown for the method A that restores the initial particle order and distribution and the method B that uses the changed particle order and distribution of a solver.

for the particle data redistribution of about 2 % of the total runtime of the P2NFFT solver in each time step.

D. Parallel Runtimes and Limited Particle Movement

Parallel runtimes of the solvers with the method A (Sect. III-A) and the method B (Sect. III-B) are shown. Additionally, in each time step, the maximum distance between the new position of a particle (line 9 in Fig. 3) and its old position is determined. This maximum movement of the particles is used by the FMM solver to switch to the merge-based parallel sorting method and by the P2NFFT solver to perform point-to-point communication with neighboring processes instead of collective all-to-all communication. The initial particles are distributed uniformly among a Cartesian process grid.

Figure 9 shows runtimes of the particle dynamics simulation with the FMM solver on the JuRoPA system (left) and with the P2NFFT solver on the Juqueen system (right) depending on the number of parallel processes. Results are shown for sorting the particles and restoring their original order and distribution (method A), for sorting the particles and resorting their velocities and accelerations (method B), and for exploiting the limited particle movement with the method B.

For the FMM solver, a significant difference exists between the runtimes for redistributing the particle data with the method A and the method B. However, until 32 processes, the data handling represents only a small part of the total runtime, thus the differences between both methods are very small. With more than 32 processes, the differences between both methods are larger: The total runtime with the method A decreases until 1024 processes, but is always higher than with the method B. The total runtime with the method B achieves a minimum with 512 processes. The biggest difference of about 33 % between the method A and the method B occurs with 256 processes. Exploiting the limited particle movement shows a small increase of the total runtime of the FMM solver that is mainly caused by the merge-based parallel sorting.

The behavior of the P2NFFT solver on the JuRoPA system is not shown because it is very similar to the FMM solver: The achieved minimum total runtime with the method B is about 39 % smaller and exploiting the limited particle movement results in a small increase of the runtimes. In these cases, the collective all-to-all communication operations perform better than using several point-to-point communication operations.

For both solvers, the missing improvements for exploiting the limited particle movement can be attributed to the JuRoPA system: The total number of parallel processes usable for the benchmark runs remains small and the switched communication network does not provide performance benefits for communication between neighboring processes. Advantages for using point-to-point communication operations may arise only with higher numbers of parallel processes and with communication network topologies, such as grid or torus networks, that provide a more efficient communication between neighboring processes of the process grid.

The behavior of the P2NFFT solver on the Juqueen system in Fig. 9 (right) is different than the behavior seen on the JuRoPA system: With more than 1024 processes, significant differences exist between methods A and B, but with the method B being slower. This performance decrease is caused by the additional communication step required for resorting the particle velocities and accelerations. With both methods, the minimum total runtime is achieved with 4096 processes. Further increasing the number of processes leads to increasing total runtimes that are caused by the particle data redistribution

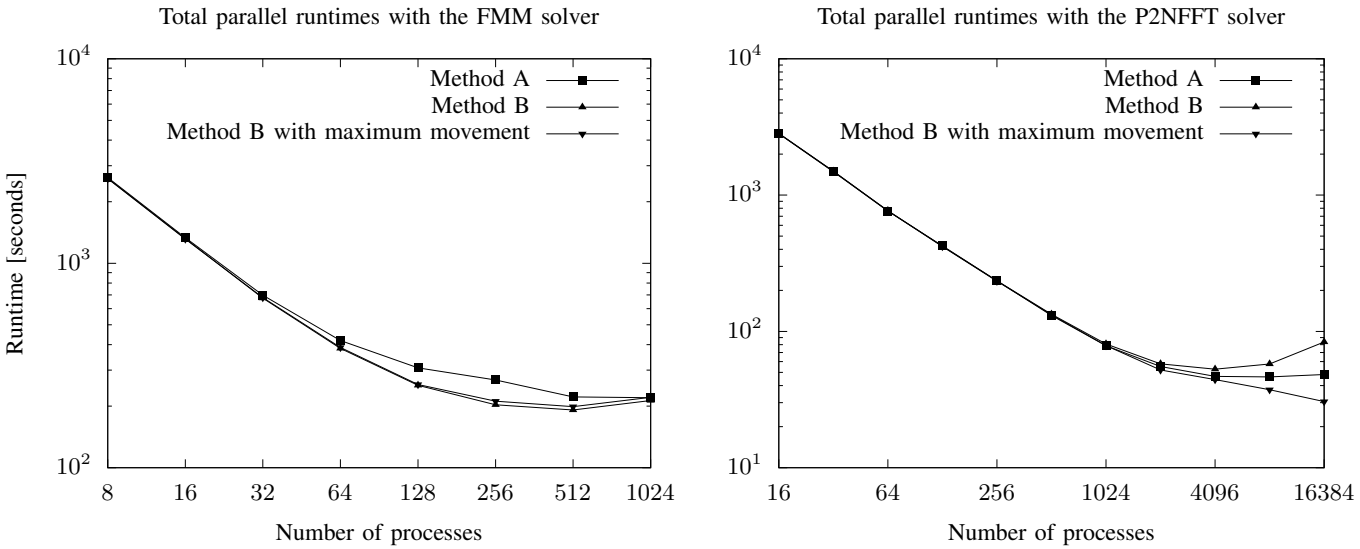

Figure 9. Parallel runtimes of particle dynamics simulations using the FMM solver on the JuRoPA system (left) and the P2NFFT solver on the Juqueen system (right). Total runtimes are shown for the method A that restores the initial particle order and distribution and the method B that uses the changed particle order and distribution of a solver. Additionally, results are shown for using the maximum movement of the particles for the method B.

of both methods. Exploiting the limited particle movement prevents the increasing runtimes observed with the method B. In these cases, the neighborhood communication with point-to-point communication operations on the torus network of the Blue Gene/Q platform is significantly faster than the collective all-to-all communication operations. The total runtime with the method B using the maximum movement of the particles continues to decrease until 16384 processes and is about 40 % smaller than the total runtime with the method A in this case.

V. Summary

In this article, we have described two methods for performing the particle data redistribution within a coupled parallel particle code that implements a particle dynamics simulations. The computation of particle interactions was performed using two fast solvers for long range interactions provided by the ScaFaCoS library. The solver methods utilize different particle data processing techniques and domain decomposition schemes, thus requiring efficient particle data reordering and redistribution methods for coupling with an independent simulation application. The first particle data redistribution method restores the original application-specific particle order and distribution after each execution of a solver method. The second particle data redistribution method uses the solver-specific particle order and distribution during the entire simulation. Performance results have shown that the second method leads to a significant reduction of the runtimes. The parallel scalability of the particle dynamics simulations was improved by reducing the amount of communication required for the particle data redistribution. Exploiting the limited movement of particles has lead to significant improvements for the high scaling Blue Gene/Q platform.

References

[1] R. Hockney and J. Eastwood, *Computer simulation using particles*. Taylor & Francis, 1988.

[2] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, pp. 325–348, 1987.

[3] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[4] A. Arnold, "Fourier transformed-based methods for long-range interactions: Ewald, P^3M and more," in *Fast Methods for Long-Range Interactions in Complex Systems*, ser. IAS Series. Forschungszentrum Jülich GmbH Zentralbibliothek, 2011, vol. 6, pp. 39–64.

[5] M. Pippig and D. Potts, "Parallel three-dimensional nonequispaced fast fourier transforms and their application to particle simulation," *SIAM Journal on Scientific Computing*, 2013, accepted.

[6] *ScaFaCoS – Scalable Fast Coloumb Solvers*, http://www.scafacos.de.

[7] *MPI: A Message-Passing Interface Standard Version 3.0*, MPI Forum, 2012, http://www.mpi-forum.org/.

[8] H. Dachsel, "An error-controlled fast multipole method," *Journal of Chemical Physics*, vol. 132, no. 119901, 2010.

[9] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[10] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical Simulation in Molecular Dynamics*. Springer, 2007.

[11] D. Frenkel and B. Smit, *Understanding Molecular Simulation*, 2nd ed. Academic Press, 2002.

[12] M. Hofmann and G. Rünger, "A partitioning algorithm for parallel sorting on distributed memory systems," in *Proc. of the 13th IEEE International Conference on High Performance Computing and Computations (HPCC'11)*. IEEE, 2011, pp. 402–411.

[13] M. Hofmann and G. Rünger, "Fine-grained data distribution operations for particle codes," in *Proc. of the 16th European PVM/MPI Users' Group Meeting Conference (EuroPVM/MPI'09)*. Springer, 2009, pp. 54–63.

[14] *ZMPI-ATASP – ZMPI All-to-all Specific Library Version 1.0.2*, http://www.tu-chemnitz.de/cs/PI/dl/software/.

[15] H. Dachsel, M. Hofmann, and G. Rünger, "Library support for parallel sorting in scientific computations," in *Proc. of the 13th International Euro-Par Conference (Euro-Par'07)*. Springer, 2007, pp. 695–704.

[16] D. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[17] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The IBM Blue Gene/Q interconnection fabric," *Micro, IEEE*, vol. 32, no. 1, pp. 32–43, 2012.