Proven Distributed Memory Parallelization of Particle Methods

JOHANNES PAHLKE, Faculty of Computer Science, Dresden University of Technology, Dresden, Germany and Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany

IVO F. SBALZARINI, Faculty of Computer Science, Dresden University of Technology, Dresden, Germany and Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany

We provide a mathematically proven parallelization scheme for particle methods on distributed-memory computer systems. Particle methods are a versatile and widely used class of algorithms for computer simulations and numerical predictions in various applications, ranging from continuum fluid dynamics to discrete granular flows and molecular dynamics simulations. Particle methods naturally lend themselves to implementation on parallel computing systems. So far, however, a mathematical proof of correctness and equivalence to sequential implementations has only been available for shared-memory parallelism. Here, we leverage a formal definition of the algorithmic class of particle methods to provide a proven parallelization scheme for distributed-memory computers. We prove that thus parallelized particle methods on distributed-memory computers are formally equivalent to their sequential counterpart for a well-defined class of particle methods, and we provide analytical expressions for the speedup and scalability bounds of this class of algorithms in function of their parameters. The parallelization scheme analyzed here is the basis of many real-world software designs for parallel particle methods. The present analysis is, therefore, of direct relevance to existing and new parallel implementations of particle methods and places them on solid theoretical grounds, rationalizing best practices and providing useful scalability and speedup bounds for benchmarking.

CCS Concepts: • **Theory of computation → Parallel algorithms**; **Distributed algorithms**; • **Computing methodologies → Distributed algorithms**; **Parallel algorithms;**

Additional Key Words and Phrases: Particle methods, simulation algorithms, formal definition, algorithmics, software engineering, parallelization, distributed memory, meshfree methods, scalability bounds

ACM Reference Format:
Johannes Pahlke and Ivo F. Sbalzarini. 2024. Proven Distributed Memory Parallelization of Particle Methods. *ACM Trans. Parallel Comput.* 11, 4, Article 17 (November 2024), 45 pages. https://doi.org/10.1145/3696189

This work was funded in parts by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907).
Authors' Contact Information: Johannes Pahlke, Faculty of Computer Science, Dresden University of Technology, Dresden, Germany and Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany; e-mail: pahlke@mpi-cbg.de; Ivo F. Sbalzarini, Faculty of Computer Science, Dresden University of Technology, Dresden, Germany and Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany; e-mail: ivos@mpi-cbg.de.

ACM 2329-4949/2024/11-ART17
https://doi.org/10.1145/3696189

1 Introduction

High-performance computing (HPC) is becoming increasingly important in research, especially in the life sciences [19, 32], where many physical experiments are not feasible due to ethical reasons or technical limitations in control and observation. At the same time, the power of computer hardware is increasing, mainly due to parallelism. This computing power enables the simulation of increasingly complex models but demands elaborate code, which typically incurs long development times. Therefore, generic software frameworks are actively developed in order to bridge the gap between accessible programming and heterogeneous hardware parallelization [27]. At the foundation of these frameworks are generic and parallelizable numerical algorithms.

Prominent examples of parallelizable numerical algorithms belong to the algorithmic class of particle methods, which has widespread use in scientific computing. Applications range from computational fluid dynamics [7] over computational chemistry [23] and plasma physics [15] to particle-based image processing [1, 6], encompassing many well-known numerical methods, such as **Discrete-Element Methods (DEMs)** [33], **Molecular Dynamics (MD)** [2], **Particle-In-Cell (PIC)** methods [15], **Reproducing Kernel Particle Methods (RKPMs)** [20], **Particle Strength Exchange (PSE)** [8, 9], **Discretization-Corrected PSE (DC-PSE)** [5, 30], and **Smoothed Particle Hydrodynamics (SPH)** [10, 22]. Particle methods have been efficiently parallelized on both shared and distributed-memory systems [16–18, 26, 29].

Recently, particle methods were mathematically defined [24], enabling their formal study across applications. Leveraging this formal definition, particle methods have been proven to be parallelizable on shared-memory systems under certain conditions [24]. However, no such results have been available for the distributed-memory parallelism prevalent in HPC. It was, therefore, not known under which conditions a distributed-memory implementation of a particle method is formally equivalent to its sequential counterpart, i.e., is correct in the sense that it computes the same results for any possible input.

Here, we provide a proof of equivalence for a basic distributed-memory parallelization scheme for particle methods and analyze the conditions under which it is valid. Our analysis is independent of a specific application or a specific numerical method. The proof covers a broad and well-defined class of particle-based algorithms. Moreover, the parallelization scheme we analyze is the basis of a wide range of practically used distributed-memory implementations. Hence, it provides the theoretical foundation for more specialized parallelization schemes [21] that can be reduced to or composited from the base case considered here.

The present base case builds on the classic cell-list algorithm [15] and a checkerboard-like domain decomposition. We formalize this scheme in mathematical equations as well as a Nassi–Shneiderman diagram. We then provide an exhaustive list of conditions a particle method must fulfill for the scheme to be correct. Under these conditions, we prove the equivalence of the presented parallelization scheme to the underlying sequential particle method. Furthermore, we use the presented formal analysis to infer the scheme's time complexity and parallel scalability bounds.

We hope this work provides a starting point for proving complexity bounds and correctness of distributed parallel codes in scientific HPC. In the long run, such efforts could lead to the development of provably correct software frameworks for scientific computing and enable formal analysis of distributed numerical simulation codes.

2 Prerequisites

Before analyzing shared-memory parallelization of particle methods, we introduce the notation used in this article and recall the classic cell-list data structure, the concept of domain

decomposition, and the formal definition of the algorithmic class of particle methods for which the statements are going to be derived.

2.1 Terminology and Notation

We start by introducing the notation and terminology used and defining the underlying mathematical concepts.

Definition 1. The *Kleene star* A^* is the set of all finite tuples of elements of a set A, including the empty tuple $()$. It is defined using the Cartesian product as

$$A^0 := \{()\},\ A^1 := A,\ A^{n+1} := A^n \times A \text{ for } n \in \mathbb{N}_{>0} \tag{1}$$

$$A^* := \bigcup_{j \in \mathbb{N}_0} A^j. \tag{2}$$

In this article, we use the Kleene star to describe the set of all particle tuples. A particle tuple is the mathematical representation of an array of particles in the computer memory.

Notation 1. We use *bold symbols* for tuples of arbitrary length, e.g.,

$$\mathbf{p} \in P^*. \tag{3}$$

This is used to denote a specific particle tuple out of the set of all possible ones.

Notation 2. We use *regular symbols with subscript indices* to denote the elements of a tuple, e.g.,

$$\mathbf{p} = (p_1, \ldots, p_n). \tag{4}$$

The elements of a particle tuple are the individual particles of a particle method.

Notation 3. We use *regular symbols* for tuples of determined length with specific elements, e.g.,

$$p = (a, b, c) \in A \times B \times C. \tag{5}$$

This is used, for example, to describe the properties of a particle p.

Notation 4. We use the same *indices* for tuples of determined length and their named elements, e.g.,

$$p_j = (a_j, b_j, c_j). \tag{6}$$

Notation 5. We use *underlined symbols* for vectors, e.g.,

$$\underline{v} \in A^n. \tag{7}$$

Vectors are a third kind of tuple whose length is fixed and all elements are from the same space. We use this to describe vectorial particle properties and cell lists.

Definition 2. Let $a \in \mathbb{R}$, $a = z + r$ with $z \in \mathbb{Z}$, $r \in \mathbb{R}$, and $0 \leq r < 1$. Then *rounding down* of a *real number* $a \in \mathbb{R}$ is defined as

$$\lfloor a \rfloor := z. \tag{8}$$

Definition 3. Rounding down of a real-valued vector $\underline{v} \in \mathbb{R}^d$ is defined element-wise:

$$\lfloor \underline{v} \rfloor := \begin{pmatrix} \lfloor v_1 \rfloor \\ \vdots \\ \lfloor v_d \rfloor \end{pmatrix}. \tag{9}$$

Definition 4. The *number of elements in a tuple* $\mathbf{p} = (p_1, \ldots, p_n) \in P^*$ is defined as

$$|\mathbf{p}| := n\,. \tag{10}$$

Definition 5. The *Euclidean length of a vector* $\underline{v} \in \mathbb{R}^d$ is defined by the ℓ^2-norm

$$\left|\underline{v}\right| = \left| \begin{matrix} v_1 \\ \vdots \\ v_d \end{matrix} \right| := \sqrt{v_1^2 + \cdots + v_d^2}\,. \tag{11}$$

Definition 6. The *composition operator* [24] $*_h$ of a binary function $h : A \times B \longrightarrow A$ is recursively defined as

$$*_h : A \times B^* \longrightarrow A \tag{12}$$

$$a *_h () := a \tag{13}$$

$$a *_h (b_1, b_2, \ldots, b_n) := h(a, b_1) *_h (b_2, \ldots, b_n)\,. \tag{14}$$

We use the composition operator to shorten and simplify equations, e.g., for the interaction of a particle with all of its neighbors.

Definition 7. The *concatenation* $\circ : A^* \times A^* \longrightarrow A^*$ of tuples $(a_1, \ldots, a_n), (b_1, \ldots, b_m) \in A^*$ is defined as

$$(a_1, \ldots, a_n) \circ (b_1, \ldots, b_m) := (a_1, \ldots, a_n, b_1, \ldots, b_m)\,. \tag{15}$$

The *big concatenation* $\bigcirc$ of tuples $\mathbf{a}_1, \ldots, \mathbf{a}_n \in A^*$ is defined as

$$\bigcirc_{j=1}^{n} \mathbf{a}_j := \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n\,. \tag{16}$$

We use the concatenation operator, for instance, to add particles from one tuple to another tuple.

Definition 8. We define the *construction of a subtuple* $\mathbf{b} \in A^*$ of $\mathbf{a} \in A^*$. Let $f : A^* \times \mathbb{N} \longrightarrow \{\top, \bot\}$ ($\top = true$, $\bot = false$) be the condition for an element a_j of the tuple $\mathbf{a}$ to be in $\mathbf{b}$. Then, $\mathbf{b} = (a_j \in \mathbf{a} : f(\mathbf{a}, j))$ defines a subtuple of $\mathbf{a}$ as

$$\begin{aligned} \mathbf{b} = (a_j \in \mathbf{a} : f(\mathbf{a}, j)) := (a_{j_1}, \ldots, a_{j_n}) \\ \longleftrightarrow \quad & \mathbf{a} = (a_1, \ldots, a_{j_1}, \ldots, a_{j_2}, \ldots, a_{j_n}, \ldots, a_m) \\ \wedge \quad & \forall k \in \{1, \ldots, n\} : \; f(\mathbf{a}, j_k) = \top \\ \wedge \quad & \forall l \in \{1, \ldots, m\} \backslash \{j_1, \ldots, j_n\} : \; f(\mathbf{a}, l) = \bot. \end{aligned} \tag{17}$$

Subtuples are used in the following to describe the neighborhood (i.e., the interaction partners) of a particle.

Definition 9. A *permutation* σ is a bijective function mapping the finite set A ($|A| < \infty$) to itself:

$$\sigma : A \longrightarrow A\,. \tag{18}$$

Definition 10. Let $\mathbf{a} = (a_1, \ldots, a_n) \in A^n$ a tuple and $\sigma : \{a_1, \ldots, a_n\} \longrightarrow \{a_1, \ldots, a_n\}$ a permutation. Then, the *permutation of a tuple* is defined as

$$\sigma(\mathbf{a}) := (\sigma(a_1), \ldots, \sigma(a_n))\,. \tag{19}$$

The permutation of a tuple is useful to keep track of potential reordering of particles throughout the algorithm.

Definition 11. Let $\alpha = (a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n$ a tuple. Then, an *element* $\langle \alpha \rangle_j$ of a tuple is defined as

$$\langle \alpha \rangle_j := a_j, \tag{20}$$

and a *collection tuple* $\langle \alpha \rangle_{(j_1,\ldots,j_m)}$ of a tuple with $j_1, \ldots, j_m \in \{1, \ldots, n\}$ is defined as

$$\langle \alpha \rangle_{(j_1,\ldots,j_m)} := (a_{j_1}, \ldots, a_{j_m}). \tag{21}$$

The collection tuple is used to select specific elements of a tuple, e.g., the neighbor particles in a particle tuple or a specific cell from a cell list.

Definition 12. A *subresult* ${}_k f$ of a function

$$f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m \tag{22}$$

is defined as

$${}_k f(a_1, \ldots, a_n) := \langle f(a_1, \ldots, a_n) \rangle_k \quad \text{with} \quad k \in \{1, \ldots, m\}. \tag{23}$$

If a function has a tuple as a result, the subresult is used to obtain a specific element of the result tuple.

Notation 6. We follow the notational convention of [24], according to which functions have uppercase superscripts to specify the level of abstraction, e.g., $copy^{ALL}$, and subscripts to specify fixed parameters, e.g., $copy^{ALL}_{\vec{g}}$.

Notation 7. We use a *bold number with over- and underbars* to denote a vector containing the same number for all entries, e.g.,

$$\underline{\overline{\mathbf{1}}} := (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{N}^d. \tag{24}$$

This notation is used to shift vectorial indices, especially in cases where the dimension is a variable.

Definition 13. Let d be the dimension of a discrete space with *vectorial index space*

$$\mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \qquad \text{with} \quad \underline{\overline{\mathbf{I}}} = (I_1, \ldots, I_d)^{\mathrm{T}} \in N^d_{>0} \tag{25}$$

and corresponding *scalar index space*

$$\left\{1, \ldots, \prod_{t=1}^{d} I_t\right\}. \tag{26}$$

Then, the *translation of a scalar index to a vectorial index* is

$$\underline{\overline{\mathbf{I}}}_\iota : \left\{1, \ldots, \prod_{t=1}^{d} I_t\right\} \to \mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \tag{27}$$

and defined as

$$\bar{\mathbf{I}}_\iota(j) := \begin{pmatrix} (j-1) - \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1 + 1 \\ \left\lfloor \frac{j-1}{I_1} \right\rfloor - \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_2 + 1 \\ \vdots \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor I_l + 1 \\ \vdots \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{d-2} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor I_{d-1} + 1 \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor + 1 \end{pmatrix}. \tag{28}$$

The *backward translation of a vectorial index to a scalar index* is

$$\bar{\mathbf{I}}_\iota^{-1} : \mathbb{N}^d \cap \left[\underline{\bar{\mathbf{1}}}, \underline{\bar{\mathbf{I}}}\right] \to \left\{1, \ldots, \prod_{t=1}^{d} I_t\right\} \tag{29}$$

and defined as

$$\bar{\mathbf{I}}_\iota^{-1}(\underline{j}) := \; 1 + (j_1 - 1) + (j_2 - 1) I_1 + (j_3 - 1) I_1 I_2 + \cdots + (j_l - 1) \prod_{t=1}^{l-1} I_t + \cdots + (j_d - 1) \prod_{t=1}^{d-1} I_t \,. \tag{30}$$

$[\underline{\bar{\mathbf{1}}}, \underline{\bar{\mathbf{I}}}]$ is the d-dimensional interval from $\underline{\bar{\mathbf{1}}}$ to $\underline{\bar{\mathbf{I}}}$. The vectorial index space is used, for instance, to index the cells of a cell list along each spatial dimension. This is possible since they form a regular Cartesian grid. The scalar index space is the linearized index, which can be used to store the cells from the d-dimensional grid in memory or iterate over them.

Notation 8. We use *square brackets* $[\cdot]$ in two situations: first, to describe closed intervals with defined borders that are included in the interval, and second, to combine multiple variables into a *single* variable, e.g., $[g, \mathbf{p}]$. The meaning is always clear from the data type of the arguments.

2.2 Cell Lists

In particle simulations, cell lists [15] are a classic data structure for neighbor access. Since particles can change their location during a simulation, cell lists are used to efficiently find the neighboring particles around any given particle in order to, e.g., compute particle interactions. The size of the neighborhood is defined by a parameter r_c, called the *cutoff radius*. Cell lists are then constructed by partitioning the simulation domain into uniform and equilateral Cartesian cells, where each edge length of each cell is equal to r_c (see Figure 1). Each particle is then, based on its position $\underline{x}_p$, assigned to the one cell with vectorial index $\underline{j}_{\text{cell}}$:

$$\underline{j}_{\text{cell}} = \left\lfloor \frac{\underline{x}_p}{r_c} \right\rfloor + \underline{\bar{\mathbf{1}}} \,. \tag{31}$$

This is done by adding the index p to the end of a linked list of particle indices that is stored in each cell. This construction ensures that particles within the same cell and its neighboring cells are the only possible interaction partners. Using the cell list as a particle-index lookup table when evaluating all pairwise particle interactions within an interaction radius of r_c reduces the time complexity of the interaction evaluation from $O(N^2)$ to $\Theta(N)$ on average for uniformly distributed particles, where N is the total number of particles. The worst-case complexity remains $O(N^2)$ when

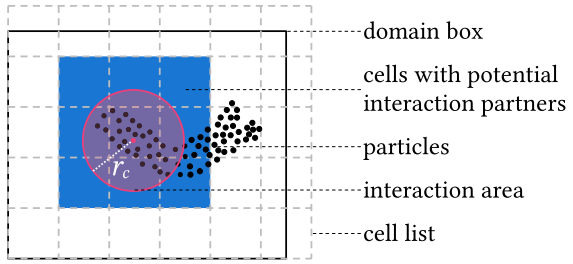


Fig. 1. Illustration of a cell-list grid with some particles. If the red particle needs to interact with all other particles within the red circle, searching over the blue cells is sufficient to find all interaction partners.

all particles cluster in a single cell. That case, however, is not of practical relevance. Moreover, we use cell lists here as they also simplify the data distribution and inter-processor communication in a distributed-memory parallel particle method.

2.3 Domain Decomposition

In order to parallelize a particle method onto a distributed-memory parallel computer, the simulation domain needs to be partitioned among the processes. In that way, each processor, or more precisely its local memory address space, only stores a part of the data of the overall particle method. This ensures that the algorithms also scale in the weak sense when using more processors. It fundamentally assumes that the simulation domain, i.e., the space in which the particles are distributed, possesses a bounding box. This assumption is not limiting, since a bounding box can always be chosen as large as needed to leave room for the free movement of the particles within the finite time of a simulation. Moreover, on a digital computer, positions cannot go to infinity, so the domain is trivially bounded by the limits of the internal number representation. Assuming the existence of a bounding box for domain decomposition, however, does not restrict how the particles can be distributed within the domain. Particle distributions are routinely irregular in order to, e.g., represent complex or moving and deforming geometries [4, 28].

The most common way of distributing data across memory address spaces is to compute a domain decomposition, assigning distinct segments to individual processes. A *process* is an autonomous and self-contained unit of computation with its own memory address space [14]. If a process needs to access data that is stored in the memory of another process, that data needs to be explicitly communicated, usually over a network interconnect. The collective amount of all communication performed by a distributed-memory parallel application, i.e., its *communication overhead*, limits the performance scalability of the application.

In particle methods, particles interact with their neighbors within a given cutoff radius. Therefore, physical-space domain decompositions (over particle locations) are used, as they lead to lower communication overhead than index-space decompositions (over particle indices). A physical-space domain decomposition partitions the simulation domain into subdomains as illustrated in Figure 2 on the left. While arbitrary partitions of unity are possible, software implementations are often restricted to aligned cuboidal subdomains, which is also what we consider here.

For all particle interactions on a given process to be computed independent of and in parallel with those on other processes, each process must possess information about particles from neighboring (in the domain decomposition) processes within the cutoff radius from its domain boundary. Most practical implementations of parallel particle methods exchange this data between processes

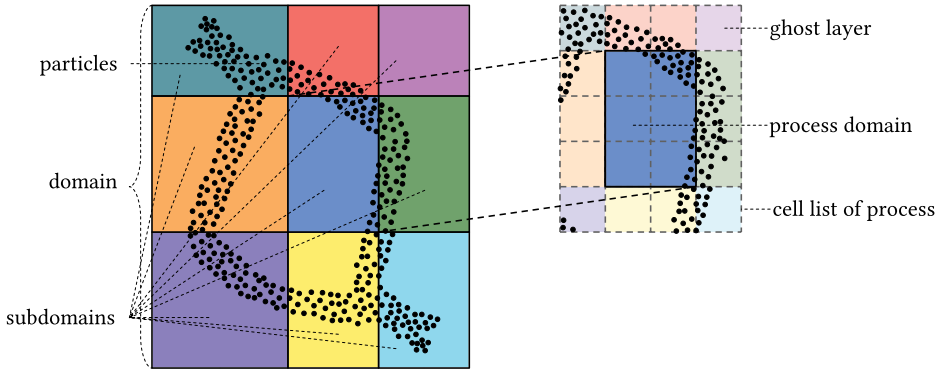


Fig. 2. Illustration of domain decomposition for distributed-memory particle methods. The simulation domain on the left is partitioned into cuboidal subdomains. Each subdomain (color) is assigned to one process. Each individual process (right inset for the example of the blue subdomain) extends its subdomain by a ghost layer of one cell-list cell thickness all around. This layer is filled with copies of the particle data from the respective neighboring processes (indicated by color). This defines the communication overhead of the parallel implementation and allows subsequent computation of pairwise particle interactions on each process without further communication.

before any interactions are computed. The boundary particles from neighboring processes are then stored in a *ghost layer*, also known as a padding or halo layer, which is an additional layer of cell-list cells outside the subdomain of the given process (Figure 2, right). Therefore, when using cell lists, domain decomposition typically operates on units of complete cells to minimize the communication overhead.

2.4 Mathematical Definition of Particle Methods

We briefly recall the formal definition of the algorithmic class of particle methods as presented in more detail elsewhere [24]. This mainly serves to remind of the core concepts for the derivations that follow. In any particle method, particles are collections of properties, such as position, velocity, mass, color, and so forth. Each particle interacts pairwise with other particles in order to change its properties and independently evolves. For convenience, and in order to better reflect practical implementations of particle methods, the definition also includes a global variable, which usually is a collection of aggregate properties, such as simulation time, total energy, total number of particles, and so forth.

In scientific simulation applications, particles can be used either to represent discrete model entities or to mathematically discretize continuous fields. A prominent example of the discrete case is molecular dynamics simulations, where particles represent atoms or molecules that interact with each other according to physical force potentials and evolve according to Newton's second law. Other discrete examples include traffic simulations with particles representing traffic participants (cars, pedestrians, bicycles, etc.) and granular flow simulations where particles represent individual grains (e.g., of sand, salt, or any other granular material). When numerically simulating models involving continuous mathematical fields, such as the electric fields in Maxwell's equations or the fluid pressure and velocity fields in the Navier–Stokes equations, particles represent mathematical discretization points of some numerical scheme. Since these discretization points are not required to lie on a grid or mesh, and they can move during the simulation (e.g., with the flow velocity in Lagrangian fluid simulations or to locally adapt the numerical resolution in adaptive solvers),

particle methods possess certain beneficial numerical properties, such as improved linear stability. Examples of widely used numerical particle schemes for continuous models include SPH for fluid mechanics and PIC for plasma physics. While these examples serve to illustrate what particle methods are, we note that equivalent algorithms are also used in fields other than scientific computing. This includes point-based graphics in computer graphics, particle-based computer vision, particle filters in statistics, and so forth, which are all subsumed in our formal definition of particle methods [24].

Formally, the definition of any particle method consists of three parts: the algorithm, the instance, and the state transition.

Particle Method Algorithm. The definition of a particle method algorithm encapsulates the structural elements of its implementation in a small set of data structures and functions that need to be specified at the onset.

In detail, the components are defined as follows:

Definition 14. A *particle method algorithm* is a 7-tuple $(P, G, u, f, i, e, \mathring{e})$, consisting of the two data structures

$$P := A_1 \times A_2 \times \cdots \times A_n \qquad \text{the particle space,} \tag{32}$$

$$G := B_1 \times B_2 \times \cdots \times B_m \qquad \text{the global variable space,} \tag{33}$$

such that $[G \times P^*]$ is the *state space* of the particle method, and five functions:

$$u : [G \times P^*] \times \mathbb{N} \to \mathbb{N}^{*\,0} \qquad \text{the neighborhood function,} \tag{34}$$

$$f : G \to \{\top, \bot\} \qquad \text{the stopping condition,} \tag{35}$$

$$i : G \times P \times P \to P \times P \qquad \text{the interact function,} \tag{36}$$

$$e : G \times P \to G \times P^{*\,0} \qquad \text{the evolve function,} \tag{37}$$

$$\mathring{e} : G \to G \qquad \text{the evolve function of the global variable.} \tag{38}$$

Particle Method Instance. A particle method instance describes the initial state of the particle method, instantiating the data structures of a particle method algorithm.

Definition 15. An initial state defines a *particle method instance* for a given particle method algorithm $(P, G, u, f, i, e, \mathring{e})$:

$$[g^1, \mathbf{p}^1] \in [G \times P^*]. \tag{39}$$

The instance consists of an initial value for the global variable $g^1 \in G$ and an initial tuple of particles $\mathbf{p}^1 \in P^*$.

Particle State Transition Function. The particle method state transition function describes how a particle method proceeds from the instance to the final state by using the particle method algorithm. The state transition function consists of a series of state transition steps. This series ends when the stopping condition f returns *true* ($\top$). The state transition steps use the functions $u, i, e, \mathring{e}$ in a way that is the same for all particle methods, regardless of their specific implementation.

Definition 16. The *state transition function* $S : [G \times P^*] \to [G \times P^*]$ is defined with the following Nassi–Shneiderman diagram:

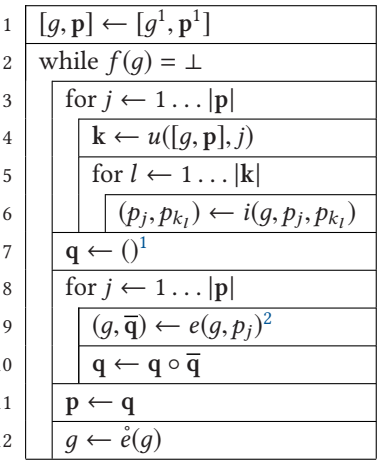


Fig. 3. Nassi–Shneiderman diagram of the state transition function S.

The Nassi–Shneiderman diagram corresponds to equations that mathematically define the state transition function S. It is divided into sub-functions here for better readability. We explicitly list these equations hereafter, indicating for each one of them which line (1–12) in the above the Nassi–Shneiderman diagram it corresponds to.

Notation 9. In the following, we use superscripts to indicate over how many particles or pairs of particles the superscribed function iterates. These are only names/labels and not variables. Specifically, I indicates that only one particle or pair is involved, whereas N indicates iterations over all particles. $\mathrm{I} \times \mathrm{U}$ indicates iterations over all pairs of one particle with each of its neighbors, and $\mathrm{N} \times \mathrm{U}$ indicates iterations over all particles and for each particle with all of its respective neighbors.

Notation 10. An *overbar* indicates a variable that stores the change to the value of the overbared variable. It therefore denotes a different variable and is not a unary operator.

All interaction sub-functions return an altered particle tuple $\mathbf{p}$. The first interaction sub-function ι^{I} calculates one single interaction and formalizes line 6:

$$\iota^{\mathrm{I}}([g, \mathbf{p}], j, k) := (p_1, \ldots, p_{j-1}, \overline{p}_j, p_{j+1}, \ldots, p_{k-1}, \overline{p}_k p_{k+1}, \ldots, p_{|\mathbf{p}|}),$$
$$\text{with} \quad (p_1, \ldots, p_{|\mathbf{p}|}) = \mathbf{p}, \quad \left(\overline{p}_j, \overline{p}_k\right) := i\left(g, p_j, p_k\right). \tag{40}$$

The second interaction sub-function $\iota^{\mathrm{I}\times\mathrm{U}}$ calculates the interaction of one particle with *all* its interaction partners and formalizes lines 4 to 6:

$$\iota^{\mathrm{I}\times\mathrm{U}}([g, \mathbf{p}], j) := \mathbf{p} \ *_{\iota^{\mathrm{I}}_{(g,j)}} \ u([g, \mathbf{p}], j) \qquad \text{with} \quad \iota^{\mathrm{I}}_{(g,j)}(\mathbf{p}, k) := \iota^{\mathrm{I}}([g, \mathbf{p}], j, k). \tag{41}$$

The third interaction sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$ calculates the interactions of *all* particles with *all* of their respective interaction partners and formalizes lines 3 to 6:

$$\iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}]) := \mathbf{p} \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_g} (1, \ldots, |\mathbf{p}|) \qquad \text{with} \quad \iota^{\mathrm{I}\times\mathrm{U}}_g(\mathbf{p}, j) := \iota^{\mathrm{I}\times\mathrm{U}}([g, \mathbf{p}], j). \tag{42}$$

[1] $\mathbf{q}$ is an intermediate result.
[2] $\overline{\mathbf{q}}$ is an intermediate result.

The first evolution sub-function ϵ^{I} calculates the evolution of one particle and stores the result in an intermediate particle tuple $\mathbf{q}$ and in the global variable. It formalizes line 9 and 10:

$$\epsilon^{\mathrm{I}}([g,\mathbf{p}],\mathbf{q},j) := \left[\overline{g},\quad \mathbf{q}\circ\overline{\mathbf{q}}\right] \qquad \text{with} \quad (\overline{g},\overline{\mathbf{q}}) := e(g,p_j). \tag{43}$$

The second evolution sub-function ϵ^{N} calculates the evolution of *all* particles and returns the result in a new particle tuple $\mathbf{q}$ and the global variable. It formalizes lines 7 to 10:

$$\epsilon^{\mathrm{N}}([g,\mathbf{p}]) := \left[g,()\right] \ *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} \ (1,\ldots,|\mathbf{p}|) \qquad \text{with} \quad \epsilon^{\mathrm{I}}_{\mathbf{p}}([g,\mathbf{q}],j) := \epsilon^{\mathrm{I}}([g,\mathbf{p}],\mathbf{q},j). \tag{44}$$

The state transition step s combines all sub-functions, formalizing lines 3 to 12:

$$s\,([g,\mathbf{p}]) := \left[\mathring{e}(\overline{g}),\ \overline{\mathbf{p}}\right] \qquad \text{with} \quad [\overline{g},\overline{\mathbf{p}}] := \epsilon^{\mathrm{N}}\left([g,\ \iota^{\mathrm{N}\times\mathrm{U}}([g,\mathbf{p}])]\right). \tag{45}$$

Finally, the state transition function S advances the instance to the final state, formalizing the complete diagram:

$$\begin{aligned} S([g^1,\mathbf{p}^1]) = [g^T,\mathbf{p}^T] \qquad &\longleftrightarrow \\ f(g^T) = \top \quad \wedge \quad \forall t \in \{2,\ldots,T\} :\ &[g^t,\mathbf{p}^t] = s\left([g^{t-1},\mathbf{p}^{t-1}]\right) \ \wedge \ f(g^{t-1}) = \bot. \end{aligned} \tag{46}$$

3 Distributed Pull Particle Methods without Global Operations

While most parallelization strategies tend to be algorithm specific, the formal definition of particle methods [24] enables us to formulate, prove, and analyze a parallelization scheme for an entire class of algorithms. We consider the parallelization of particle methods on distributed-memory computers. Shared-memory parallelism of particle methods has previously been considered [24].

The present parallelization scheme is valid for particle methods with pull interactions and without global operations. Pull interactions denote the case where only the interacting particle is changed and there is no change to any of its neighbor particles and interaction partners. This is also sometimes referred to as *asymmetric particle interactions*. The parallelization scheme analyzed here is based on cell lists and domain decomposition, as described in the previous sections and often used in practical code implementations.

Contrary to practical implementations, however, we assume a checkerboard inter-process communication schedule in order to avoid conflicting or colliding communications. This is not usually implemented in practice but rather left to the scheduler of the network communication library used. Nevertheless, we assume it here for simplicity of the mathematical notation. In addition, we assume that the particle method contains no distributed operations on the global variable, as that would require additional inter-process communication (reduction followed by broadcast).

In the resulting parallelization scheme, each process concurrently executes the steps illustrated in Figure 4 from top to bottom for each iteration (time step) of the particle method. This is the minimal parallelization scheme for particle methods, which we study here in order to prove its correctness and to derive bounds on its complexity and scalability. Any more sophisticated or elaborate scheme would have to be within the resulting bounds and be reducible to this minimal scheme for correctness.

Formally, cell lists require that each particle p has a position $\underline{x}$:

$$p = (\ldots,\underline{x},\ldots) \in P, \tag{47}$$

and that the neighborhood function u is based on a cutoff radius r_c and not directly on particle indices. This makes it, in some sense, order independent. Hence, it is restricted to the form

$$u\,([g,\mathbf{p}],j) := \left(k \in (1,\ldots,|\mathbf{p}|) : p_k,p_j \in \mathbf{p} \ \wedge |\underline{x}_k - \underline{x}_j| \le r_c \wedge \Omega(g,j,k,p_j,p_k)\right)$$
$$\text{with } \Omega : G\times\mathbb{N}\times\mathbb{N}\times P\times P \to \{\top,\bot\}, \tag{48}$$

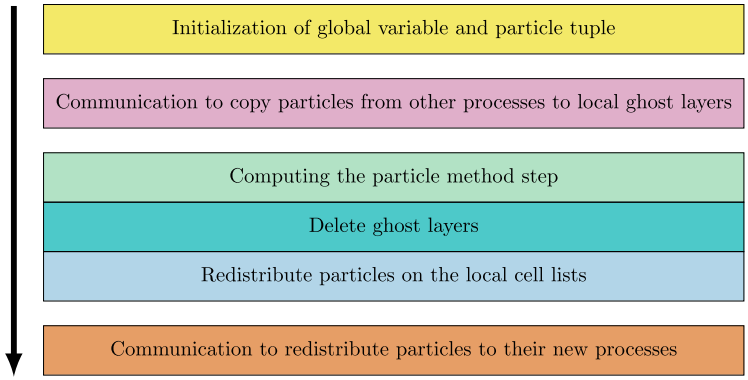


Fig. 4. Overview of the minimal parallelization scheme for distributed-memory particle methods with pull interactions and without global operations. In each iteration of the particle method, each process concurrently executes the steps from top to bottom.

where the function Ω is an additional constraint that can be used to encode more complex interaction neighborhoods, e.g., to exclude self-interactions ($\Omega(g, j, k, p_j, p_k) := (j \neq k)$) or to express anisotropic or adaptive interaction neighborhoods.

The position $\underline{x}$ of a particle is limited to a cuboidal spatial domain of dimension d for all states/times t:

$$\forall t \in \{1, \ldots, T\}\ \forall p_j^t \in \mathbf{p}^t : \underline{x}_j^t \in [\underline{D}_{\min}, \underline{D}_{\max}) \subset \mathbb{R}^d. \tag{49}$$

This is usually achieved with some boundary conditions, e.g., periodic boundary conditions, at the edges of the domain. In addition, the position is not allowed to change by more than a cutoff radius r_c in a single state transition step:

$$\forall t \in \{1, \ldots, T-1\}\ \forall p_j^t \in \mathbf{p}^t : |\underline{x}_j^t - \underline{x}_j^{t+1}| \leq r_c. \tag{50}$$

This latter condition is, in most cases, not practically limiting, since the particle displacements are usually even more strongly restricted by the consistency and stability requirements of the numerical method, e.g., MD, PSE, DEM. Here, we require this restriction because we are going to consider the limit of the finest possible domain decomposition, where each subdomain is of smallest possible edge length r_c as well as the ghost layer. In practice, unless the numerical scheme requires otherwise, the condition can be relaxed but must be accounted for.

Further, the interaction is limited to a pull interaction:

$$\forall p_j, p_k \in P, g \in G :\ i(g, p_j, p_k) = (\overline{p}_j, p_k), \tag{51}$$

independent of previous interactions of the interaction partner to avoid chain dependencies of interaction results:

$$\forall p_j, p_k, p_{k'} \in P, g \in G :\ {}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) = {}_1i_g(p_j, p_k) \quad \text{with} \quad {}_1i_g(p_j, p_k) := \langle i(g, p_j, p_k)\rangle_1, \tag{52}$$

and be order independent:

$$\forall p_j, p_k, p_{k'} \in P, g \in G :\ {}_1i_g({}_1i_g(p_j, p_k), p_{k'}) = {}_1i_g({}_1i_g(p_j, p_{k'}), p_k). \tag{53}$$

Order independence is not strictly required, but it is not limiting in practice and avoids re-sorting the particles. The neighborhood function u needs to be independent of previous interactions so the cell list can be prepared before the interactions happen. Here, $*_{\mathrm{I}}$ is the composition operator

defined in Definition 6 and used on ι^{I}, whereas ι^{I} is the first transition sub-function defined in Equation (40):

$$\forall j, k', k'' \in \{1, \ldots, |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*] : \; u([g, \mathbf{p}], j) = u\left(\left[g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')\right], j\right). \tag{54}$$

Finally, we assume that the evolve function e does not change the global variable g:

$$\forall g \in G, p \in P : e(g, p) = (g, \mathbf{q}), \tag{55}$$

which avoids global operations and their $O(\log(\mathit{number\ of\ processes}))$ communication complexity [11].

Under these constraints on a particle method, we formulate the parallelization of any particle method onto multiple distributed-memory processes. The parallelization scheme is based on the cell-list algorithm [15]. Cell lists require the definition of a cutoff radius r_c to divide the domain into equal-sized Cartesian cells. The number of cells along each dimension of the computational domain is represented by the vector

$$\underline{\overline{\mathbf{I}}} = \begin{pmatrix} I_1 \\ \vdots \\ I_d \end{pmatrix} := \left\lfloor \tfrac{1}{r_c}(\underline{D}_{\max} - \underline{D}_{\min}) + \underline{\overline{\mathbf{1}}} \right\rfloor. \tag{56}$$

The total number of cells is

$$N_{\mathrm{cell}} := \prod_{l=1}^{d} I_l. \tag{57}$$

To prove correctness of the parallel scheme, we consider the case with the highest degree of parallelism for this scheme. The highest degree of parallelism is reached when assigning each cell-list cell to a separate process. This domain decomposition of finest granularity also avoids load balancing. Therefore, it results in a simpler overall scheme and, hence, proof. Each process then has to exchange information with its direct (face-, edge-, and corner-connected) neighbors in the cell-list grid. We assume that each process can only communicate with one other process at a time, and that the network behaves as a fully connected graph. In order to guarantee that communications never overlap, we assume that any two communicating processes have at least two inactive processes between them. This results in a checkerboard-like inter-process communication pattern, as illustrated in Figure 5, which is the Cartesian special case of the graph-coloring communication scheduling often implemented in practice [29].

The number of active cells in each dimension is given by the vector

$$\underline{{}^{k}\overline{\mathbf{I}}^*} = \begin{pmatrix} {}^{k}I_1^* \\ \vdots \\ {}^{k}I_d^* \end{pmatrix} := \left\lfloor \tfrac{1}{\overline{3}} \left(\underline{\overline{\mathbf{I}}} - \underline{\overline{3}}\iota(k) + \underline{\overline{3}} \right) \right\rfloor, \tag{58}$$

where k represents one of the 3^d different communicating process configurations $k \in \{1, \ldots, 3^d\}$. In total, the number of communicating processes for each k is

$$^{k}N_{\mathrm{cell}}^* := \prod_{l=1}^{d} {}^{k}I_l^*. \tag{59}$$

We address the communicating processes for each k by

$$\gamma(k, j) := \overline{\mathbf{I}}\iota^{-1}\left(\underline{{}^{k}\overline{\mathbf{I}}^*}\iota(j) \cdot 3 + \underline{\overline{3}}\iota(k) - \underline{\overline{3}} \right). \tag{60}$$

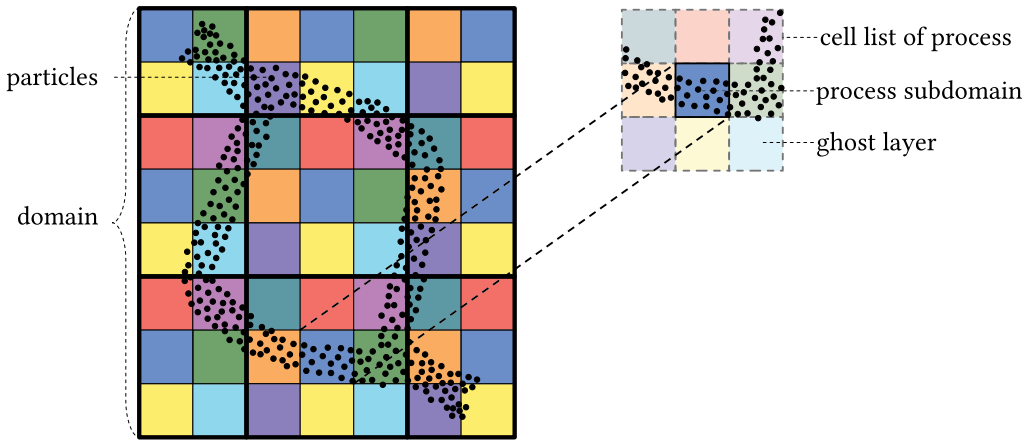


Fig. 5. Illustration of the checkerboard-like inter-process communication strategy. Each colored square represents a different process. Processes of the same color communicate concurrently, while the others wait. Once all pairwise communication has happened, each process (right inset for an example) has its ghost layers populated with data from all neighboring processes.

The lth neighbor cell of the tth process is

$$\beta(t,l) := \bar{\underline{\mathrm{I}}}\iota^{-1}\left(\bar{\underline{\mathrm{I}}}\iota(t) + \bar{\underline{3}}\iota(l) - \underline{\bar{2}}\right). \tag{61}$$

The result of β is undefined unless the result of $\bar{\underline{\mathrm{I}}}\iota^{-1}$, Definition 13 and Equation (56), is defined. We note that this checkerboard-like pattern is not how codes are usually implemented in practice. There, communications would all be launched at once, leaving their scheduling to the networking sub-system. For the sake of theoretical analysis, however, the two are interchangeable, and it separates the consideration of the algorithm from the properties of the interconnect.

We formulate the standard procedure of distributing particles into a cell list according to Figure 1 as a condition for the initial particle distribution. Hence, each cell k initially contains a tuple of particles $\mathbf{p}^1_k$. To ensure no particle is lost, we require that the concatenation of these particle tuples is a permutation π of the initial particle tuple $\mathbf{p}^1$:

$$\mathbf{p}^1_1 \circ \ldots \circ \mathbf{p}^1_{N_\mathrm{cell}} = \pi\left(\mathbf{p}^1\right), \tag{62}$$

and that particles are distributed according to their position:

$$\forall p^1_j \in \mathbf{p}^1 : p^1_j \in \mathbf{p}^1_w \quad \text{with} \quad w = \bar{\underline{\mathrm{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}^1_j - \underline{D}_\mathrm{min})\right\rfloor + \underline{\bar{1}}\right). \tag{63}$$

Each process has its own memory address space, containing its global variable storage g and its particle storage $\underline{\mathbf{p}}$. The latter stores the particles within the cell-list cell assigned to that process and copies of the particles from neighboring cells, as illustrated in Figure 5. Therefore, the particle storage of each process is compartmentalized cell-wise:

$$\underline{\mathbf{p}} = \begin{pmatrix} \left\langle \underline{\mathbf{p}} \right\rangle_1 \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \end{pmatrix}^\top \in (P^*)^{3^d}, \tag{64}$$

where the center entry (i.e., location $\frac{3^d+1}{2}$) contains the "real" particles of the cell assigned to that process. The other entries contain the copies of the cells from the neighboring processes in the order corresponding to their position in the cell list. Two processes are neighbors if the corresponding cells are direct (face-, edge-, and corner-connected) neighbors in the cell-list grid.

The **particle storages of all processes (PROC)** together is

$$\mathcal{P} = \begin{pmatrix} {}^{\text{PROC}_1}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{N_{\text{cell}}}}\underline{\mathbf{p}} \end{pmatrix}^{\top} \in \left((P^*)^{3^d} \right)^{N_{\text{cell}}}. \tag{65}$$

We use "PROC" with an index to emphasize that the variable lives on a specific process. The initial particle tuples in each process's particle storage are

$${}^{\text{PROC}_n}\underline{\mathbf{p}}^1 = \begin{pmatrix} (), \ldots, (), \underbrace{\mathbf{p}_n^1}_{\frac{3^d+1}{2}\text{-th entry}}, (), \ldots, () \end{pmatrix}. \tag{66}$$

Then, the initial particle storage of all processes is

$$\mathcal{P}^1 := \begin{pmatrix} {}^{\text{PROC}_1}\underline{\mathbf{p}}^1 \\ \vdots \\ {}^{\text{PROC}_{N_{\text{cell}}}}\underline{\mathbf{p}}^1 \end{pmatrix}^{\top}. \tag{67}$$

The global variable storage of all processes is

$$\mathcal{G}^1 := (g^1, \ldots, g^1) \in G^{N_{\text{cell}}}. \tag{68}$$

We introduce a second global variable $\tilde{g}$ to store the cell-list-specific parameters from Equations (49) and (56). This is different from the global variable g of the particle method. We therefore use a tilde to distinguish this second global variable from the standard one:

$$\tilde{g} = \left(\underline{D}_{\text{min}}, \underline{D}_{\text{max}}, d, \bar{\mathbf{I}} \right). \tag{69}$$

The function $copy_{(\tilde{g},\mathcal{P})}$ copies the center particle storage compartment of a process $\text{PROC}_{\beta(w,l)}$ to the specified compartment l in the storage $\underline{\mathbf{p}}$ of another process:

$$copy_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l) := \left(\left\langle \underline{\mathbf{p}} \right\rangle_1, \ldots, \left\langle \underline{\mathbf{p}} \right\rangle_{l-1}, \left\langle {}^{\text{PROC}_{\beta(w,l)}}\underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}}, \left\langle \underline{\mathbf{p}} \right\rangle_{l+1}, \ldots, \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \right). \tag{70}$$

This coping is done for every kth cell in the checkerboard-like configuration by

$$copy_{\tilde{g}}^{ACTIVE}(\mathcal{P},k) := \begin{pmatrix} {}^{\text{PROC}_1}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma(k,1)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma(k,1)}}\underline{\mathbf{p}} *_{copy_{(\tilde{g},\mathcal{P},\gamma(k,1))}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma(k,1)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma(k,l)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma(k,l)}}\underline{\mathbf{p}} *_{copy_{(\tilde{g},\mathcal{P},\gamma(k,l))}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma(k,l)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma\left(k,{}^{k}N^*_{\text{cell}}\right)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma\left(k,{}^{k}N^*_{\text{cell}}\right)}}\underline{\mathbf{p}} *_{copy_{\left(\tilde{g},\mathcal{P},\gamma\left(k,{}^{k}N^*_{\text{cell}}\right)\right)}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma\left(k,{}^{k}N^*_{\text{cell}}\right)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{N_{\text{cell}}}}\underline{\mathbf{p}} \end{pmatrix}^{\top}. \tag{71}$$

Doing so for all distinct checkerboard-like configurations results in

$$copy_{\tilde{g}}^{ALL}(\mathcal{P}) := \mathcal{P} *_{copy_{\tilde{g}}^{ACTIVE}} \left(1,\ldots,3^d\right). \tag{72}$$

After the function $copy_{\tilde{g}}^{ALL}$, each process has (copies of) all particles required to calculate the interactions and evolutions of the particles in its cell without any further inter-process communication. We define for this scheme the interaction of all particles with their respective interaction partners by the function

$$interaction_g(\underline{\mathbf{p}}) := \begin{pmatrix} \langle \mathbf{q} \rangle_1 \;\; *_1 i_g \left\langle \bigcirc_{w=1}^{3^d} \langle \underline{\mathbf{p}} \rangle_w \right\rangle_{u\left(\left[g, \bigcirc_{w=1}^{3^d} \langle \underline{\mathbf{p}} \rangle_w\right], z+1\right)} \\ \vdots \\ \langle \mathbf{q} \rangle_{|\mathbf{q}|} \;\; *_1 i_g \left\langle \bigcirc_{w=1}^{3^d} \langle \underline{\mathbf{p}} \rangle_w \right\rangle_{u\left(\left[g, \bigcirc_{w=1}^{3^d} \langle \underline{\mathbf{p}} \rangle_w\right], z+|\mathbf{q}|\right)} \end{pmatrix}^{\top}, \tag{73}$$

where $z = \left| \bigcirc_{w=1}^{\frac{3^d+1}{2}-1} \langle \underline{\mathbf{p}} \rangle_w \right|$ and $\mathbf{q} =< \underline{\mathbf{p}} >_{\frac{3^d+1}{2}}$. The step function $step_{(\tilde{g},g)}$ uses the function $interaction_g$ to compute the state transition step (mostly simulation time step) including the

evolutions of the particle properties and positions:

$$step_{(\tilde{g},g)}(\underline{\mathbf{p}}) := \begin{pmatrix} \left\langle \underline{\mathbf{p}} \right\rangle_1 \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}-1} \\ {}_2\epsilon^{\mathrm{N}}\left(\left[g, interaction_g\left(\underline{\mathbf{p}}\right)\right]\right) \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}+1} \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \end{pmatrix}^{\mathsf{T}}. \tag{74}$$

Calculating the state transition step on all processes results in

$$step^{ALL}_{(\tilde{g},\tilde{\mathcal{G}})}(\mathcal{P}) := \left(step_{(\tilde{g},\ {}^{[\mathrm{PROC1}]}g)}\left({}^{[\mathrm{PROC1}]}\underline{\mathbf{p}}\right),\ \ldots,\ step_{(\tilde{g},\ {}^{[\mathrm{PROC}|\mathcal{P}|]}g)}\left({}^{[\mathrm{PROC}|\mathcal{P}|]}\underline{\mathbf{p}}\right)\right), \tag{75}$$

where ${}^{[\mathrm{PROC}k]}g := \langle \mathcal{G} \rangle_k$ and ${}^{[\mathrm{PROC}k]}\underline{\mathbf{p}} := \langle \mathcal{P} \rangle_k$.

After the function $step^{ALL}_{(\tilde{g},\tilde{\mathcal{G}})}$, all particles in the center storage compartments of all processes have the correct positions and properties. However, they may be in the wrong cell/storage compartment after moving. Therefore, the particles in the center storage compartments must be reassigned into the compartments and communicated to the new process if they have moved to another cell/compartment. Cell/compartment assignment of each particle q is done by

$$dist_{(\tilde{g},g,j)}(\underline{\mathbf{p}}, q) := \begin{pmatrix} \left\langle \underline{\mathbf{p}} \right\rangle_1 \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\alpha-1} \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\alpha} \circ (q) \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\alpha+1} \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \end{pmatrix}^{\mathsf{T}}, \tag{76}$$

where

$$q = (\ldots, \underline{x}, \ldots) \in P, \quad \alpha := {}^{\overline{3}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{\min}) \right\rfloor - {}^{\overline{1}}\iota(j) + \overline{\mathbf{3}}\right). \tag{77}$$

For all particles in a center storage compartment $\mathbf{q}$, redistribution is done by the function

$$dist^N_{(\tilde{g},g,j)}(\mathbf{q}) := ((), \ldots, ()) *_{dist_{(\tilde{g},g,j)}} \mathbf{q}, \tag{78}$$

where $((), \ldots, ())$ is the tuple of 3^d empty tuples, representing the empty particle storage of a process. For all processes, the redistribution procedure is

$$dist^{ALL}_{(\tilde{g},\tilde{\mathcal{G}})}(\mathcal{P}) := \begin{pmatrix} dist^N_{(\tilde{g},\ {}^{\mathrm{PROC}_1}g,1)}\left(\left\langle {}^{\mathrm{PROC}_1}\underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}}\right) \\ \vdots \\ dist^N_{\left(\tilde{g},\ {}^{\mathrm{PROC}_{|\mathcal{P}|}}g, |\mathcal{P}|\right)}\left(\left\langle {}^{\mathrm{PROC}_{|\mathcal{P}|}}\underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}}\right) \end{pmatrix}^{\mathsf{T}}, \tag{79}$$

where ${}^{\text{PROC}_k}g := \langle \mathcal{G} \rangle_k$ and ${}^{\text{PROC}_k}\underline{\mathbf{p}} := \langle \mathcal{P} \rangle_k$. After the function $dist^{ALL}_{(\tilde{g},\tilde{\mathcal{G}})}$ the particles on all processes are correctly assigned to their respective storage compartments/cell-list cells, but those particles may belong to another process now, except for the particles in the center storage compartment. Therefore, communication between processes is required for the particles that have moved to a different storage compartment/cell. Collecting the particles that moved from the $\beta(w,l)$-th process to the wth process is done by the function

$$collect_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l) := \begin{pmatrix} \left\langle \underline{\mathbf{p}} \right\rangle_1 \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}-1} \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}} \circ \left\langle {}^{\text{PROC}_{\beta(w,l)}}\underline{\mathbf{p}} \right\rangle_{\bar{\bar{3}}_l^{-1}\left(\bar{\underline{4}}-\bar{\bar{3}}_l(l)\right)} \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}+1} \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \end{pmatrix}. \tag{80}$$

All processes collecting particles from the other processes simultaneously could lead to overlapping communications. As discussed above, we aim to prevent this. Therefore, the collection procedure again follows the checkerboard-like pattern. For the kth checkerboard-like pattern, the collection function is

$$collect^N_{\tilde{g}}(\mathcal{P}, k) := \begin{pmatrix} {}^{\text{PROC}_1}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma(k,1)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma(k,1)}}\underline{\mathbf{p}} *_{collect_{(\tilde{g},\mathcal{P},\gamma(k,1))}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma(k,1)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma(k,l)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma(k,l)}}\underline{\mathbf{p}} *_{collect_{(\tilde{g},\mathcal{P},\gamma(k,l))}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma(k,l)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{\gamma\left(k,{}^kN^*_{\text{cell}}\right)-1}}\underline{\mathbf{p}} \\ {}^{\text{PROC}_{\gamma\left(k,{}^kN^*_{\text{cell}}\right)}}\underline{\mathbf{p}} *_{collect_{\left(\tilde{g},\mathcal{P},\gamma\left(k,{}^kN^*_{\text{cell}}\right)\right)}} \left(1,\ldots,3^d\right) \\ {}^{\text{PROC}_{\gamma\left(k,{}^kN^*_{\text{cell}}\right)+1}}\underline{\mathbf{p}} \\ \vdots \\ {}^{\text{PROC}_{|\mathcal{P}|}}\underline{\mathbf{p}} \end{pmatrix}^{\top}, \tag{81}$$

where ${}^{\text{PROC}_k}\underline{\mathbf{p}} := \langle \mathcal{P} \rangle_k$. The complete collection procedure is sequential over the 3^d checkerboard-like patterns. Hence, it takes 3^d steps to finish. The complete collection is

$$collect_{\tilde{g}}^{ALL}(\mathcal{P}) := \mathcal{P} *_{collect_{\tilde{g}}^{N}} \left(1, \ldots, 3^d\right), \tag{82}$$

which results in each central particle storage compartment of all processes containing the correct particles. The copies from the neighboring processes will then be updated in the subsequent iteration, again by the function $copy_{\tilde{g}}^{ALL}$.

Taking all functions together, the parallelized state transition step for a distributed-memory particle method can be formally expressed as

$$\tilde{s}_{\tilde{g}}\left([\mathcal{G}, \mathcal{P}]\right) = \left[\begin{pmatrix} \overset{\circ}{e}\left(\langle \mathcal{G} \rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\langle \mathcal{G} \rangle_{N_{\text{cell}}}\right) \end{pmatrix}^{\mathrm{T}}, collect_{\tilde{g}}^{ALL}\left(dist_{(\tilde{g},\tilde{G})}^{ALL}\left(step_{\tilde{G}}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P})\right)\right)\right) \right]. \tag{83}$$

We use this parallel state transition step $\tilde{s}_{\tilde{g}}$ to define the parallel state transition function similar to Equation (46):

$$\begin{aligned} &\tilde{S}\left(\left[\mathcal{G}^1, \mathcal{P}^1\right]\right) := \left[\mathcal{G}^T, \mathcal{P}^T\right] \quad \longleftrightarrow \\ &f\left(\left\langle \mathcal{G}^T \right\rangle_1\right) = \top \quad \wedge \quad \forall t \in \{2, \ldots, T\} : \left[\mathcal{G}^t, \mathcal{P}^t\right] = \tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^{t-1}, \mathcal{P}^{t-1}\right]\right) \wedge f\left(\left\langle \mathcal{G}^{t-1} \right\rangle_1\right) = \bot. \end{aligned} \tag{84}$$

The complete parallelization scheme is summarized by the Nassi–Shneiderman diagram in Figure 6, where the blocks are colored according to the steps from Figure 4.

The algorithm starts by initializing the global variable storage (line 1) and the particle storage (line 3). The global variable storage is filled with the initial global variable, and the center particle storage compartment is filled with the particles from the corresponding cell-list cell. The remaining particle storage compartments are filled with copies of the particles from directly adjacent neighboring cells by copying them from the respective process (lines 5–8), which is repeated for each state (line 4). Then, each process evaluates the state transition step of the particle method (lines 10 and 11), which is only guaranteed to return the correct result for the particles of the center particle storage compartment. The results of the remaining particles may be corrupted by missing interactions with particles outside the process storage. Therefore, the algorithm proceeds to store the particles of the center storage compartment in $\mathbf{q}$ (line 12) and deletes all particles in the particle storage $\underline{\mathbf{p}}$ (line 13). The particles of the center storage compartment, now in $\mathbf{q}$, are redistributed to the process's particle storage compartments according to their new positions (lines 14–16). Finally, the algorithm collects for each process all particles that newly belong to it and which are in the corresponding particle storage compartments on other processes (lines 17–21).

3.1 Lemmata

We aim to formally prove that the above distributed-memory parallelization of a particle method is equivalent to the original sequential algorithm. To prepare the proof, we first derive a couple of useful lemmata. The proofs for all lemmata can be found in Appendix A.

Lemma 1. *$\overline{\mathrm{I}}_\iota$ and $\overline{\mathrm{I}}_\iota{}^{-1}$ are bijections and mutual functional inverses, i.e., $(\overline{\mathrm{I}}_\iota{}^{-1})^{-1} = \overline{\mathrm{I}}_\iota$.*

Lemma 2. *Order independence of a series of interactions follows from the order independence of the interact function:*

$$ {}_1i_g({}_1i_g(p_j, p_k), p_{k'}) =_1 {}_1i_g({}_1i_g(p_j, p_{k'}), p_{k''}) \tag{85}$$

$$\rightarrow \tilde{p} \; *_{{}_1i_g} \; \sigma(p_1, \ldots, p_n) = \tilde{p} \; *_{{}_1i_g} \; (p_1, \ldots, p_n). \tag{86}$$

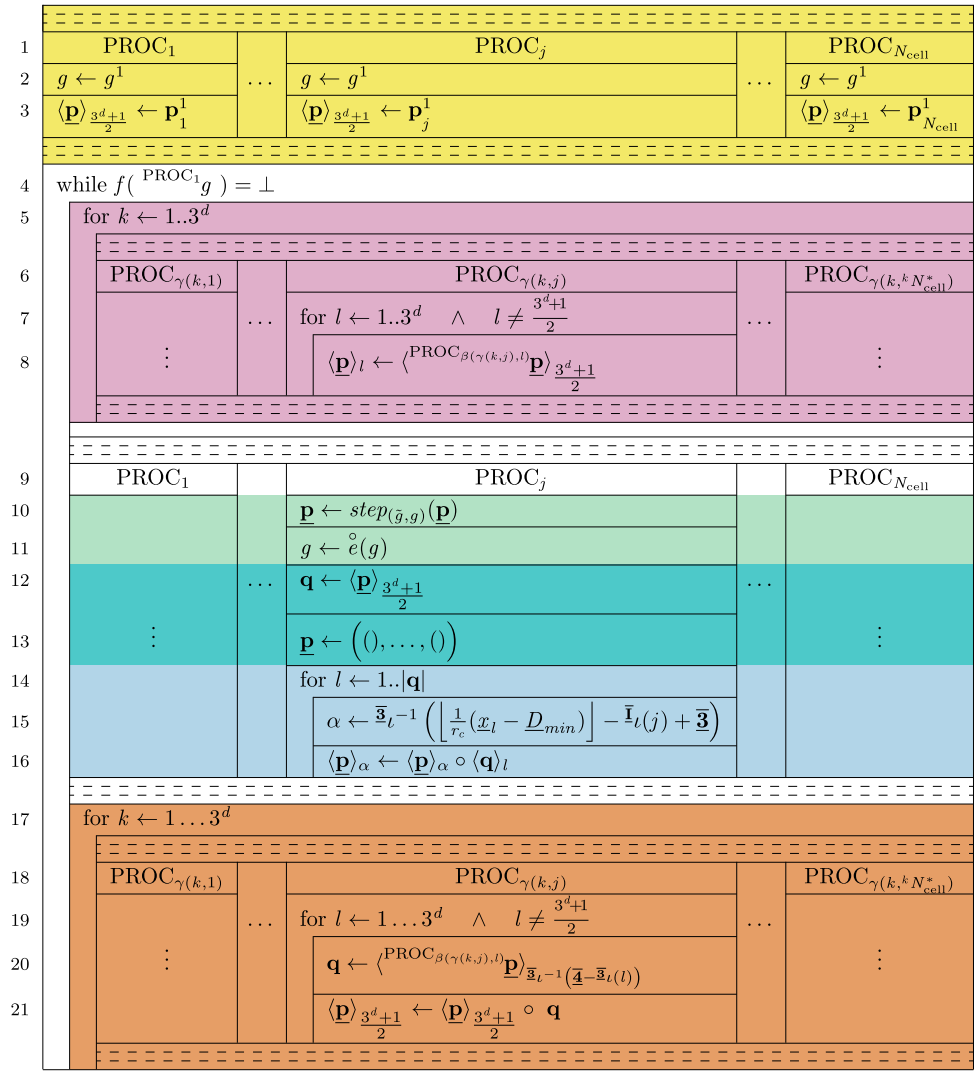


Fig. 6. Nassi–Shneiderman diagram of the distributed-memory parallelization of the outer loop of a particle method with pull interactions. The dashed double lines mark parallel sections of the algorithm. The diagram is colored according to the six basic steps of the algorithm as defined in Figure 4.

Lemma 3. *The function* $copy_{\tilde{g}}^{ALL}$ *does not cause overlapping communications. This implies that two processes can only communicate to the same third process or to each other.*

Lemma 4. *Each process "owns" one distinct cell-list cell. The* $copy_{\tilde{g}}^{ALL}$ *function copies the particles from all neighbor cells/processes. After the copy procedure, the particle storage compartments of all processes contain the particles of the corresponding cell and copies of the particles from all neighboring cells. Therefore, after each process has executed the copy function, all interaction partners of all particles in the center cell are in process-local memory.*

Lemma 5. *The function* $dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ *redistributes particles from the center particle storage compartment to only the other storage compartments on the same process. Therefore, it does not try to place*

them into non-existing storage compartments. Hence,

$$\forall w \in \{1,\ldots,N_{cell}\} \forall p_j^1 \in \left\langle \left\langle \mathcal{P}^1 \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} : p_j^2 := \left\langle \left\langle \left\langle step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j$$
$$\longrightarrow \alpha = \overline{\overline{3}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min})\right\rfloor - \overline{\overline{\mathrm{I}}}\iota(w) + \underline{\overline{3}}\right) \in \{1,\ldots,3^d\}. \quad (87)$$

Lemma 6. *The function $dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ places particles only into storage compartments that represent cells inside the computational domain. Hence, processes at the domain border do not have particles in their "outer" storage compartments. Let*

$$\underline{w} = (w_1,\ldots,w_d)^{\mathrm{T}} := \overline{\mathrm{I}}\iota(w) \quad (88)$$

$$\underline{\alpha} = (\alpha_1,\ldots,\alpha_d)^{\mathrm{T}} := \left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min})\right\rfloor - \underline{w} + \underline{\overline{3}}, \quad (89)$$

then

$$\forall w \in \{1,\ldots,N_{\mathrm{cell}}\} \quad \forall p_j^2 := \left\langle \left\langle \left\langle step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j :$$
$$(k \in \{1,\ldots,d\} \wedge w_k = 1) \longrightarrow \alpha_k \in \{2,3\} \wedge (k \in \{1,\ldots,d\} \wedge w_k = I_k) \longrightarrow \alpha_k \in \{1,2\}. \quad (90)$$

Lemma 7. *$\overline{\mathcal{P}}^1 := step_{\mathcal{G}^1}^{ALL}(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1))$ does not necessarily fulfill the condition, Equation (63), to iterate $step_{\mathcal{G}^2}^{ALL}(copy_{\tilde{g}}^{ALL}(\overline{\mathcal{P}}^1))$. To avoid $interact_g$ missing interactions due to incomplete neighborhoods, the particles in $\overline{\mathcal{P}}^1$ need to be redistributed such that Equation (63) is fulfilled. The decomposition of the initial permuted particle tuple is stored in the center storage compartment of the processes. Hence, Equation (63) can be rewritten for all state transition steps as*

$$\forall p_j^t \in \bigcirc_{v=1}^{N_{\mathrm{cell}}} \left\langle \left\langle \mathcal{P}^t \right\rangle_v \right\rangle_{\frac{3^d+1}{2}} : p_j^t \in \left\langle \left\langle \mathcal{P}^t \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}, \quad (91)$$

where

$$w = \overline{\mathrm{I}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^t - \underline{D}_{\min})\right\rfloor + \underline{\overline{1}}\right). \quad (92)$$

This is achieved by the two functions $dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ and $collect_{\tilde{g}}^{ALL}$.

3.2 Proof of Correctness and Equivalence with Sequential Algorithm

Theorem 1. *On the particles stored in the center storage compartments of the processes, and under the assumptions stated at the beginning of this section, the presented parallelization scheme for particle methods on distributed-memory computers in Figure 6 computes the same results, except for particle ordering, as the underlying sequential particle method in Figure 3. Hence,*

$$S\left(\left[g^1, \mathbf{p}^1\right]\right) = \left[\left\langle \mathcal{G}^T \right\rangle_1, \sigma^*\left(\bigcirc_{w=1}^{N_{cell}} \left\langle \left\langle \mathcal{P}^T \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}\right)\right], \quad (93)$$

where

$$\left[\mathcal{G}^T, \mathcal{P}^T\right] = \tilde{S}\left(\left[\mathcal{G}^1, \mathcal{P}^1\right]\right). \quad (94)$$

We prove that each sequential state transition step s, Equation (45), is equivalent to each parallel state transition step $\tilde{s}$, Equation (83), and that both algorithms terminate after the same number of state transitions. The sequential state transition step was defined as follows:

$$s\left([g, \mathbf{p}]\right) := \left[\overset{\circ}{e}(\overline{g}), \overline{\mathbf{p}}\right] \quad \text{with } [\overline{g}, \overline{\mathbf{p}}] = \epsilon^{\mathrm{N}}\left(g, \iota^{\mathrm{N}\times\mathrm{U}}\left([g, \mathbf{p}]\right)\right). \quad \text{(Equation (45))}$$

Under the condition that the interact function, Equation (52), and the neighborhood function, Equation (54), are independent of previous interactions, we can rewrite the third interact subfunction $\iota^{\mathrm{N}\times\mathrm{U}}$ for pull interaction particle methods, Equation (51), to [24]

$$\iota^{\mathrm{N}\times\mathrm{U}}\left([g,\mathbf{p}]\right) = \begin{pmatrix} p_1 *_1 i_g \ \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)} \\ \vdots \\ p_{|\mathbf{p}|} *_1 i_g \ \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)} \end{pmatrix}^\top. \tag{95}$$

We also know [24] that if the evolve function does not change the global variable, Equation (55), we can write the state transition step as

$$s\left([g,\mathbf{p}]\right) := \left[\overset{\circ}{e}(g),\ {}_2\epsilon^{\mathrm{N}}\left(\begin{pmatrix} p_1 *_1 i_g \ \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)} \\ \vdots \\ p_{|\mathbf{p}|} *_1 i_g \ \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)} \end{pmatrix}^\top \right) \right]. \tag{96}$$

We use proof by induction to prove equivalence for the global variable and the particles separately. We start with the global variable. Following the construction of $\mathcal{G}^1$, we directly get

$$\begin{pmatrix} g^2 \\ \vdots \\ g^2 \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(g^1\right) \\ \vdots \\ \overset{\circ}{e}\left(g^1\right) \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(\langle \mathcal{G}^1 \rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\langle \mathcal{G}^1 \rangle_{N_{\mathrm{cell}}}\right) \end{pmatrix}^\top = \begin{pmatrix} \langle \mathcal{G}^2 \rangle_1 \\ \vdots \\ \langle \mathcal{G}^2 \rangle_{N_{\mathrm{cell}}} \end{pmatrix}^\top = \mathcal{G}^2. \tag{97}$$

This is the base case for the proof by induction. For the induction step, we start from

$$\begin{pmatrix} g^t \\ \vdots \\ g^t \end{pmatrix}^\top = \mathcal{G}^t \tag{98}$$

and get

$$\begin{pmatrix} g^{t+1} \\ \vdots \\ g^{t+1} \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(g^t\right) \\ \vdots \\ \overset{\circ}{e}\left(g^t\right) \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(\langle \mathcal{G}^t \rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\langle \mathcal{G}^t \rangle_{N_{\mathrm{cell}}}\right) \end{pmatrix}^\top = \begin{pmatrix} \langle \mathcal{G}^{t+1} \rangle_1 \\ \vdots \\ \langle \mathcal{G}^{t+1} \rangle_{N_{\mathrm{cell}}} \end{pmatrix}^\top = \mathcal{G}^{t+1}. \tag{99}$$

This completes the part for the global variable.

We now prove that the sequential state transition function S and the distributed-memory state transition function $\tilde{S}$ stop after the same number of states. For the state transition S, the stop function f only depends on g^t. For $\tilde{S}$, f depends only on $\langle \mathcal{G}^t \rangle_1$. From this and $g^t = \langle \mathcal{G}^t \rangle_1$ follows

$$\underline{{}_1S\left(\left[g^1,\mathbf{p}^1\right]\right) = g^T = \langle \mathcal{G}^T \rangle_1 = \langle {}_1\tilde{S}\left(\left[\mathcal{G}^1,\mathcal{P}^1\right]\right) \rangle_1} \tag{100}$$

$${}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\begin{pmatrix} p_1^1 *_1 i_{g^1} \ \langle \mathbf{p}^1 \rangle_{u(g^1,\mathbf{p}^1,1)} \\ \vdots \\ p^1_{|\mathbf{p}^1|} *_1 i_{g^1} \ \langle \mathbf{p}^1 \rangle_{u(g^1,\mathbf{p}^1,|\mathbf{p}^1|)} \end{pmatrix}^\top \right). \tag{101}$$

The interact function is order independent, Equation (53), and the permutation $\pi(\mathbf{p}^1)$, Equation (62), of the initial particle tuple sorts the particles into cells. The neighborhood function is restricted

to not using indices, Equation (48). Hence, for a permuted particle tuple, the neighborhood function returns the same result as for the unpermuted particle tuple, with the same permutation also applied to the result. This leads to

$$ {}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\left(\begin{array}{c} p^1_{\pi(1)} *_1 i_{g^1} \left\langle \pi(\mathbf{p}^1)\right\rangle_{u(g^1, \pi(\mathbf{p}^1), \pi(1))} \\ \vdots \\ p^1_{\pi(|\mathbf{p}^1|)} *_1 i_{g^1} \left\langle \pi(\mathbf{p}^1)\right\rangle_{u(g^1, \pi(\mathbf{p}^1), \pi(|\mathbf{p}^1|))} \end{array}\right)^{\top}\right)\right). \tag{102} $$

From the construction of the initial particle storages of all processes $\mathcal{P}^1$, Equations (63) to (67), as well as Lemmata 3 and 4, we derive

$$ {}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left(\begin{array}{c} {}^w p^1_1 *_1 i_{g^1} \ {}^w\mathbf{q}^1{}_{u(g^1, {}^w\mathbf{q}^1, {}^w z+1)} \\ \vdots \\ {}^w p^1_{{}^w n} *_1 i_{g^1} \ {}^w\mathbf{q}^1{}_{u(g^1, {}^w\mathbf{q}^1, {}^w z+{}^w n)} \end{array}\right)^{\top}\right)\right), \tag{103} $$

where the tuple of all particles in the storage of the wth process is

$$ {}^w\mathbf{q}^1 := \bigcirc_{l=1}^{3^d}\left\langle\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right\rangle_l, \tag{104} $$

the number of particles in ${}^w\mathbf{q}^1$ in front of the particles of the central storage compartment is

$$ {}^w z := \sum_{l=1}^{\frac{3^d+1}{2}-1}\left|\left\langle\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right\rangle_l\right|, \tag{105} $$

the number of particles in the central storage compartment is

$$ {}^w n = \left|\left\langle\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right|, \tag{106} $$

and the particles in the central storage compartment are

$$ \left({}^w p^1_1, \ldots, {}^w p^1_{{}^w n}\right) = \left\langle\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}. \tag{107} $$

The interactions of the particles with their neighbors in Equation (103) resemble the $interaction_g$ function (Equation (3)):

$$ {}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}} interaction_{g^1}\left(\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{108} $$

The evolution function cannot change the global variable (condition in Equation (55)). Hence, the second evolution subfunction ϵ^{N} cannot change the global variable either and is, therefore, independent of the ordering of the particles. Since the evolve function e can create or destroy particles, the permutation π^{-1} cannot rearrange the result of ϵ^{N} to the sequential state transition result. But the calculation on the particles is identical. Hence, there exists a permutation $\tilde{\pi}^{-1}$ that rearranges the result such that

$$ {}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left({}_2\epsilon^{\mathrm{N}}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}} interaction_{g^1}\left(\left\langle copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{109} $$

All processes calculate the $intercation_g$ function independently. Hence, the ϵ^{N} function can also be executed on each process independently. This means that

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} {}_2\epsilon^{\mathrm{N}}\left(interaction_{g^1}\left(\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{110}$$

The combination of ${}_2\epsilon^{\mathrm{N}}$ and $interaction_g$ is the same as the $step_{(\tilde{g},g)}$ function, Equation (74), at the center storage compartment:

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle step_{(\tilde{g},g^1)}\left(\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right\rangle_{\frac{3^d+1}{2}}\right). \tag{111}$$

The function $step_{\mathcal{G}^1}^{ALL}$, Equation (75), calculates the step for all processes, leading to

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{112}$$

$\overline{\mathcal{P}}^1 := step_{\mathcal{G}^1}^{ALL}(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1))$ does not necessarily fulfill the condition, Equation (63), to iterate $step_{\mathcal{G}^2}^{ALL}(copy_{\tilde{g}}^{ALL}(\overline{\mathcal{P}}^1))$. For $interact_g$ to not miss interactions due to incomplete neighborhoods, the particles in $\overline{\mathcal{P}}^1$ need to be redistributed such that the condition in Equation (63) is fulfilled. This is achieved by the functions $dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$, Equation (79), and $collect_{\tilde{g}}^{ALL}$, Equation (82), as proven in the Lemmata 5, 6, and 7. Hence,

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle collect_{\tilde{g}}^{ALL}\left(dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}\left(step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right)\right)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \tag{113}$$

where $\tilde{\pi}'^{-1}$ is a new permutation. We insert the definition of ${}_2\tilde{s}_{\tilde{g}}$ to get

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle \underbrace{{}_2\tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^1,\mathcal{P}^1\right]\right)}_{=:\mathcal{P}^2}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \tag{114}$$

$$ \left[g^2,\mathbf{p}^2\right] = \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle \mathcal{P}^2\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{115}$$

This is the base case for the proof by induction. For the induction step, we start from

$$ \left[g^t,\mathbf{p}^t\right] = \tilde{\pi}''^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle \mathcal{P}^t\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{116}$$

We can assume that $\mathcal{P}^t$ fulfills the same conditions as $\mathcal{P}^2$ and, hence, also as $\mathcal{P}^1$, especially the condition in Equation (63) (or Equations (103) and (104)). Then, we can define a new particle method where $\left[g^t,\mathbf{p}^t\right]$ is the instance and $\mathcal{P}^t$ its corresponding cell-list-based distribution onto processes. Using Lemmata 5, 6, and 7, we derive that

$$ {}_2s\left(\left[g^t,\mathbf{p}^t\right]\right) = \tilde{\pi}'''^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle \underbrace{{}_2\tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^t,\mathcal{P}^t\right]\right)}_{=:\mathcal{P}^{t+1}}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \tag{117}$$

and hence,

$$ \left[g^{t+1},\mathbf{p}^{t+1}\right] = \tilde{\pi}'''^{-1}\left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle\left\langle \mathcal{P}^{t+1}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{118}$$

We do this now for all t until $f(g^t) = \top$. Together with the derivation of the proof for the global variable, this leads to

$$S\left(\left[g^1, \mathbf{p}^1\right]\right) = \left[\left\langle \mathcal{G}^T \right\rangle_1 , \sigma^* \left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle \left\langle \mathcal{P}^T \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right)\right], \tag{119}$$

where $\left[\mathcal{G}^T, \mathcal{P}^T\right] = \tilde{S}\left(\left[\mathcal{G}^1, \mathcal{P}^1\right]\right)$.

Hence, the present parallelization scheme produces the same result up to a different ordering of the particles. The particles are permuted by an unknown permutation σ^*. This proves that the distributed-memory parallelization scheme is correct for order-independent particle methods.

4 Bounds on Time Complexity and Parallel Scalability

For the parallelization scheme considered in the previous section, we derive bounds on the time complexity and the parallel scalability in both the weak and strong scaling cases. Since these bounds are for a minimal parallelization scheme, they should provide useful limits for more elaborate parallelization schemes, and they indicate the relevance of the considered parallelization scheme. The detailed derivations and closed-form formulas can be found in Appendix B.

An algorithm's time complexity describes the runtime required by a machine to execute that algorithm. It depends on the input size of the algorithm. For a particle method, the input size is the length of the initial tuple $\mathbf{p}^1$. We assume that constants bound the sizes of the global variable and of each particle. We further assume that the algorithm terminates in finite time. Hence, upper bounds exist for all functions' time complexities.

Since particles can move, the neighborhood search function checks each pair of particles to see if they are neighbors. This has a complexity in $O(n^2)$, where n is the number of particles. Fast neighbor list algorithms like cell lists [15] reduce this to $O(n)$ under the assumption of even and constant particle density. We can confirm this by comparing the simplified time complexity of the cell-list-based scheme, Equation (267), with the time complexity of the sequential state transition, Equation (260), where we eliminated the dependency of the time complexity of the neighborhood function on the particle number. Our cell-list-based scheme scales linearly with the number of particles ($N_{\text{cell}} n_{\max} \approx N_{\mathbf{p}}^{\max}$) and not quadratically as it would without the assumption of even and constant particle density. Hence, the speedup is $O(N_{\mathbf{p}}^{\max})$ as derived in Equation (277) and visualized in Figure 7(a).

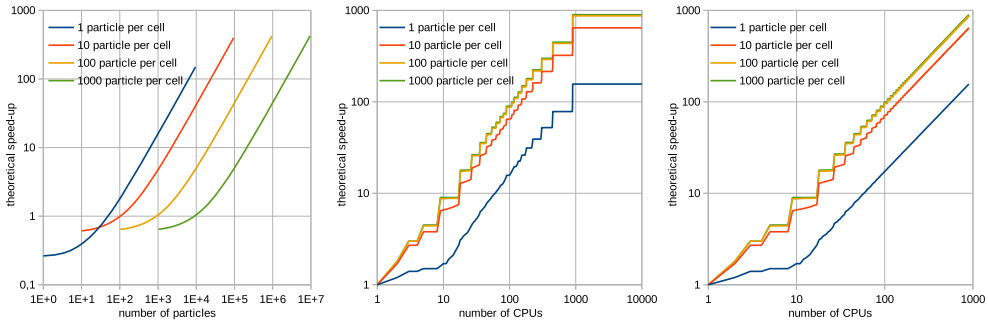
Second, Amdahl's law [3] provides an upper bound on the speedup of the cell-list scheme on multiple processors when the problem size is fixed for increasing processors n_{CPU} (strong scaling). In this case, we can increase n_{CPU} until we reach the number of cells N_{cell}. After that, there will be no further speedup. But also, until then, we find a step-like behavior with increasing n_{CPU} because cells cannot be split across CPUs, as visualized in Figure 7(b).

Third, Gustafson's law [12] provides an upper bound on the speedup of the cell-list scheme on multiple processors when the ratio of problem size to process number is constant while increasing the number of processors (weak scaling); $\frac{N_{\text{cell}}}{n_{\text{CPU}}} = \text{const}$. For a perfectly fitting processor interconnect network topology, we predict a linear speedup on average with steps as visualized in Figure 7(c).

Overall, the scheme behaves as expected for cell-list algorithms.

5 Conclusions and Discussion

Particle methods encompass a wide range of computer simulation algorithms, such as DEMs [33], MD [2], RKPMs [20], PSE [8, 9], DC-PSE [5, 30], and SPH [10, 22]. Although particle methods are widely used, mathematically defined, and practically parallelized in software frameworks such as the PPM Library [29], OpenFPM [16], POOMA [26], or FDPS [17], little work has been



(a) Speed-up of the cell list algorithm.

(b) Speed-up according to Amdahl's law.

(c) Speed-up according to Gustafson's law.

Fig. 7. Theoretical speedup bounds. The constants are chosen to be $d = 2$, $C_u = C_\alpha = C_\beta = C_\gamma = 1$, $\tau_i = \tau_e = 3$, and $\tau_f = \tau_{\tilde{e}} = 1$. For Amdahl's and Gustafson's laws, $N_{\text{cell}} = 900$.

done to formally investigate generic parallelization schemes for particle methods and prove their correctness, complexity, and scalability.

Here, we took a first step in this direction by providing a distributed-memory parallelization scheme for particle methods independent of specific applications. We proved the correctness of the presented cell-list parallelization scheme by showing equivalence to the sequential particle methods definition under certain assumptions, which we also defined here. Finally, we derived upper bounds on the time complexity of the proposed scheme executed on both sequential and parallel computers, and we discussed the parallel scalability limits in the weak and strong scaling case.

The presented parallelization scheme is not novel, and it is, in fact, a consensus of what is commonly implemented in software. Therefore, our analysis is of immediate practical relevance for general-purpose particle methods frameworks, even for critical applications, as the proof guarantees algorithmic correctness, leaving room only for errors induced by finite-precision arithmetics.

The present analysis was deliberately done for a minimal, simple parallelization scheme. Practically implemented schemes tend to be more elaborate and, e.g., deviate from our assumption of a uniform Cartesian domain decomposition, isotropic and uniform particle interaction radii, or the collision-free checkerboard communication pattern. They are, however, similar in structure, such that the bounds derived here can still provide useful limits. Also, many more elaborate schemes can be reduced to the present minimal scheme to show their correctness. This would not yield conceptually different or novel results, albeit the mathematical notation of the proofs would become a lot more involved.

We focused here on the distributed-memory parallelization of particle methods. The shared-memory case has already been proven earlier [24]. We also did not consider hardware-specific parallelization strategies, like streaming multi-threading on GPUs [13, 16], nor instruction-level parallelism in SIMD processor architectures [25, 31]. Another limitation of the present parallelization scheme is that it considers only pull interactions between the particles. This neglects the potential runtime benefits of symmetric interaction evaluations, where both interaction partners are changed simultaneously. However, pull interactions are suitable for more computer architectures, and they are more general, encompassing also particle methods that do not possess symmetric interaction functions (e.g., DC-PSE [30]). Pull interactions also have lower communication overhead, since no changes to ghost particles need to be sent back to the source processes [29]. In this sense, the scalability bounds derived here are valid also for push–pull schemes, which are expected to scale worse but potentially have lower overall runtime. We also restricted the neighborhood

function such that the cell-list strategy became applicable. This limits the expressiveness of the particle method, for example, neglecting spatially adaptive or anisotropic interaction neighborhoods. However, more complex neighborhoods could be modeled using the function Ω in the neighborhood step, maintaining the validity of the present correctness proof also in this case.

Further, the constraint that particles are not allowed to leave the domain keeps domain decomposition simple. Otherwise, subdomains would dynamically need to be added or removed as necessary, resulting in a more complex and dynamic mapping of subdomains to processes. In practice, however, this assumption is not limiting, since a bounding box can always be chosen as large as needed to leave room for the free movement of the particles within the finite time of a simulation. Moreover, it does not preclude irregular particle distributions within the domain to, e.g., represent complex or moving and deforming geometries [4, 28]. We also restricted particles to not move further than the cutoff radius in a single-state iteration or time step of the algorithm. Since the cutoff radius determines the smallest possible cell size and individual cells cannot be split across multiple processes, this guarantees that processes only need to communicate with their immediately adjacent neighbors. In practice, this constraint can be relaxed to require particles to never move further than one subdomain per state iteration. Our choice, however, enabled us to prove correctness in the limit case of the highest degree of parallelism, where each cell-list cell is assigned to a different process. Additionally, we restricted the global variable to only be changed by the evolve function of the global variable and not by the evolve function of any particle. Therefore, no global operations were allowed that would require additional communication. Still, global variable changes can be independently computed locally, keeping them synchronized without communication. Finally, the time complexity of the checkerboard-like communication scheme does not have optimal pre-factor, leaving many processes inactive. Nevertheless, it scales linearly with the number of processes and abstracts the internal scheduling of the network subsystem, providing a correct asymptotic bound on the scalability while permitting correctness proofs.

Notwithstanding these limitations, the present parallelization scheme, with its proof of equivalence to the sequential particle methods definition, stands in contrast with algorithm-specific or empirically tested parallelization schemes. We therefore hope that our analysis paves the way for future research on the theory of parallel scientific simulation algorithms and the engineering of provably correct parallel software implementations.

Future theoretical work could optimize the presented scheme for computer architectures with parallel or asynchronously clocked communication. Global operations could then also be allowed, and the checkerboard-like communication pattern could be relaxed, leading to an improved scalability pre-factor. Also, the network topology of the machine's interconnect could be explicitly incorporated into the parallelization scheme. Furthermore, proofs for push and symmetric interaction schemes could be beneficial for specific use cases, as well as combining parallelization schemes for shared and distributed memory to better match the heterogeneous architecture of modern supercomputers. On the software engineering side, future work could leverage the presented parallelization scheme and proofs to design a new generation of theoretically founded software frameworks. They would potentially be more predictable, suitable for security- and safety-critical applications, and more maintainable and understandable as they are based on a common formal framework [27].

Overall, the presented proven parallelization scheme provides a way to parallelize a class of particle methods on distributed-memory systems with full knowledge of its validity, performance, and assumptions. We proved that it computes the same result as the underlying sequential particle method. Therefore, using it in a general framework for particle methods is suitable, even for critical computations, since the proof guarantees that the parallelization does not change the results. We therefore hope that the present work will generate downstream investigation and studies in the theory of algorithms for scientific computing.

Appendices

A Proofs of Lemmata

We prove the helping lemmata for the proof that the presented parallelization scheme is equivalent to the original particle method.

Proof of Lemma 1. We need to prove that $\bar{\underline{\mathrm{I}}}_\iota$ and $\bar{\underline{\mathrm{I}}}_\iota^{-1}$ are bijective, and we need to prove that $\bar{\underline{\mathrm{I}}}_\iota$ and $\bar{\underline{\mathrm{I}}}_\iota^{-1}$ are the inverse functions of one another.

$$\forall j \in \left\{1, \ldots, \prod_{\delta=1}^{d} I_\delta\right\} : \bar{\underline{\mathrm{I}}}_\iota^{-1}\left(\bar{\underline{\mathrm{I}}}_\iota(j)\right) = j \tag{120}$$

$$\wedge \quad \forall \underline{j} \in \mathbb{N}^d \cap \left[\underline{\mathbf{1}}, \bar{\underline{\mathrm{I}}}\right] : \bar{\underline{\mathrm{I}}}_\iota\left(\bar{\underline{\mathrm{I}}}_\iota^{-1}(\underline{j})\right) = \underline{j}. \tag{121}$$

First, we prove Equation (120):

$$\bar{\underline{\mathrm{I}}}_\iota^{-1}\left(\bar{\underline{\mathrm{I}}}_\iota(j)\right) = 1 + \left(\left((j-1) - \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1 + 1\right) - 1\right) \quad + \left(\left(\left\lfloor \frac{j-1}{I_1} \right\rfloor - \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_2 + 1\right) - 1\right) I_1 \tag{122}$$

$$+ \left(\left(\left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor - \left\lfloor \frac{j-1}{I_1 I_2 I_3} \right\rfloor I_3 + 1\right) - 1\right) I_1 I_2 \quad + \cdots + \left(\left(\left\lfloor \frac{j-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor I_l + 1\right) - 1\right) \prod_{t=1}^{l-1} I_t \tag{123}$$

$$+ \left(\left(\left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{l+1} I_t} \right\rfloor I_{l+1} + 1\right) - 1\right) \prod_{t=1}^{l} I_t \tag{124}$$

$$+ \cdots + \left(\left(\left\lfloor \frac{j-1}{\prod_{t=1}^{d-2} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor I_{d-1} + 1\right) - 1\right) \prod_{t=1}^{d-2} I_t \quad + \left(\left(\left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor + 1\right) - 1\right) \prod_{t=0}^{d-1} I_t \tag{125}$$

$$= 1 + (j-1) \quad \underbrace{- \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1 + \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1}_{=0} \quad \underbrace{- \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_2 I_1 + \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_1 I_2}_{=0} \tag{126}$$

$$- \left\lfloor \frac{j-1}{I_1 I_2 I_3} \right\rfloor I_3 I_1 I_2 + \cdots + \left\lfloor \frac{j-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor \prod_{t=1}^{l-1} I_t \quad \underbrace{- \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor \prod_{t=1}^{l} I_t + \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor \prod_{t=1}^{l} I_t}_{=0} \tag{127}$$

$$- \left\lfloor \frac{j-1}{\prod_{t=1}^{l+1} I_t} \right\rfloor \prod_{t=1}^{l+1} I_t + \cdots + \left\lfloor \frac{j-1}{\prod_{t=1}^{d-2} I_t} \right\rfloor \prod_{t=1}^{d-2} I_t \quad \underbrace{- \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor I_{d-1} \prod_{t=1}^{d-2} I_t + \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor \prod_{t=1}^{d-1} I_t}_{=0} \tag{128}$$

$$= j. \tag{129}$$

Second, we prove Equation (121) by subdividing the resulting vector in its entries, starting with the first entry:

$$\left\langle \bar{\underline{\mathrm{I}}}_\iota\left(\bar{\underline{\mathrm{I}}}_\iota^{-1}(\underline{j})\right)\right\rangle_1 = \left(\left(1 + (j_1 - 1) + (j_2 - 1) I_1 + \cdots + (j_d - 1) \prod_{t=1}^{d-1} I_t\right) - 1\right) \tag{130}$$

$$- \left\lfloor \frac{\left(1 + (j_1 - 1) + \cdots + (j_d - 1) \prod_{t=1}^{d-1} I_t\right) - 1}{I_1} \right\rfloor I_1 + 1 \tag{131}$$

$$
(\textit{side calculations})\left[\begin{array}{rl}
 & \left\lfloor \frac{\left(1+(j_1-1)+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{I_1} \right\rfloor I_1 \\
= & \left\lfloor \frac{j_1-1}{I_1} + \frac{(j_2-1)I_1+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t}{I_1} \right\rfloor I_1 \\
= & \left\lfloor \underbrace{\frac{j_1-1}{I_1}}_{1\le j_1\le I_1} + \underbrace{(j_2-1)+\cdots+(j_d-1)\prod_{t=2}^{d-1} I_t}_{\in\mathbb{N}_0} \right\rfloor I_1 \\
= & \left((j_2-1)+\cdots+(j_d-1)\prod_{t=2}^{d-1} I_t\right) I_1 \\
= & (j_2-1)I_1+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t
\end{array}\right] \tag{132}
$$

$$
=(j_1-1)+(j_2-1)I_1+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t \;-\left((j_2-1)I_1+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t\right)+1 \tag{133}
$$

$$
=j_1. \tag{134}
$$

We proceed with all entries except the first and last entry:

$$
\forall l\in\{2,\ldots,d-1\}: \tag{135}
$$

$$
\left\langle \bar{\mathrm{I}}_l\left(\bar{\mathrm{I}}_l^{-1}(\underline{j})\right)\right\rangle_l = \left\lfloor \frac{\left(1+(j_1-1)+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor \tag{136}
$$

$$
-\left\lfloor \frac{\left(1+(j_1-1)+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l} I_t} \right\rfloor I_l + 1 \tag{137}
$$

$$
\textit{auxiliary calculations}: \tag{138}
$$

$$
\left[\begin{array}{rl}
 & \left\lfloor \frac{\left(1+(j_1-1)+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor \\
= & \left\lfloor \frac{(j_1-1)+\cdots+(j_{l-1}-1)\prod_{t=1}^{l-2} I_t}{\prod_{t=1}^{l-1} I_t} + \frac{(j_l-1)\prod_{t=1}^{l-1} I_t+\cdots+(j_d-1)\prod_{t=1}^{d-1} I_t}{\prod_{t=1}^{l-1} I_t} \right\rfloor \\
 & \textit{auxiliary calculations}: \\
 & \left[\begin{array}{rl}
0\le & \underbrace{(j_1-1)}_{1\le j_1\le I_1}+\cdots+\underbrace{(j_{l-1}-1)}_{1\le j_{l-1}\le I_{l-1}}\prod_{t=1}^{l-2} I_t \\
\le & I_1-1+(I_2-1)I_1+\cdots+(I_{l-1}-1)\prod_{t=1}^{l-2} I_t \\
= & I_1-1+I_2I_1-I_1+\cdots+\prod_{t=1}^{l-1} I_t-\prod_{t=1}^{l-2} I_t \\
= & \prod_{t=0}^{l-1} I_t-1 \\
\prod_{t=0}^{l-1} I_t-1< & \prod_{t=0}^{l-1} I_t
\end{array}\right] \\
= & \left\lfloor \underbrace{\frac{(j_1-1)+\cdots+(j_{l-1}-1)\prod_{t=1}^{l-2} I_t}{\prod_{t=1}^{l-1} I_t}}_{\ge 0,\ <1} + \underbrace{(j_l-1)+\cdots+(j_d-1)\prod_{t=l}^{d-1} I_t}_{\in\mathbb{N}_0} \right\rfloor \\
= & (j_l-1)+\cdots+(j_d-1)\prod_{t=l}^{d-1} I_t
\end{array}\right] \tag{139}
$$

$$
=\left((j_l-1)+\cdots+(j_d-1)\prod_{t=l}^{d-1} I_t\right)-\left((j_{l+1}-1)+\cdots+(j_d-1)\prod_{t=l+1}^{d-1} I_t\right)I_l+1 \tag{140}
$$

$$=(j_l-1)+(j_{l+1}-1)I_l+\cdots+(j_d-1)\prod_{t=l}^{d-1}I_t-(j_{l+1}-1)I_l-\cdots-(j_d-1)\prod_{t=l}^{d-1}I_t+1 \tag{141}$$

$$=j_l. \tag{142}$$

We finish with the last entry:

$$\left\langle \bar{\underline{\mathrm{I}}}_\iota\left(\bar{\underline{\mathrm{I}}}_\iota^{-1}(\underline{j})\right)\right\rangle_d = \left\lfloor \frac{\left(1+(j_1-1)+\cdots+(j_d-1)\prod_{t=1}^{d-1}I_t\right)-1}{\prod_{t=1}^{d-1}I_t}\right\rfloor+1 \tag{143}$$

$$=\left\lfloor \underbrace{\frac{1+(j_1-1)+\cdots+(j_{d-1}-1)\prod_{t=1}^{d-2}I_t}{\prod_{t=1}^{d-1}I_t}}_{<1}+\underbrace{(j_d-1)}_{\in\mathbb{N}_0}\right\rfloor+1 \tag{144}$$

$$=j_d. \tag{145}$$

Taking these three results together, we can conclude

$$\bar{\underline{\mathrm{I}}}_\iota\left(\bar{\underline{\mathrm{I}}}_\iota^{-1}(\underline{j})\right)=\begin{bmatrix} j_1\\ j_2\\ \vdots\\ j_d\end{bmatrix} \tag{146}$$

$$=\underline{j}. \tag{147}$$

We need to prove that $\bar{\underline{\mathrm{I}}}_\iota$ and $\bar{\underline{\mathrm{I}}}_\iota^{-1}$ are bijective, and we need to prove that $\bar{\underline{\mathrm{I}}}_\iota$ and $\bar{\underline{\mathrm{I}}}_\iota^{-1}$ are the inverse functions of one another. □

Proof of Lemma 2. We need to prove that the order independence of the interact function follows the order independence of a series of interactions:

$${}_1i_g({}_1i_g(p_j,p_k),p_{k'})={}_1i_g({}_1i_g(p_j,p_{k'}),p_{k''}) \tag{148}$$

$$\rightarrow \tilde{p}\ *_{{}_1i_g}\ \sigma(p_1,\ldots,p_n)=\tilde{p}\ *_{{}_1i_g}\ (p_1,\ldots,p_n). \tag{149}$$

The proof idea is to use the bubble sort strategy to go from an arbitrary permutation σ of the interacting particles to the original order. Therefore, we prove that the necessary swap of two consecutive particles does not change the result:

$${}_1i_g({}_1i_g(p_j,p_k),p_{k'})={}_1i_g({}_1i_g(p_j,p_{k'}),p_{k''}) \tag{150}$$

$$\rightarrow \tilde{p}\ *_{{}_1i_g}\ \sigma(p_1,\ldots,p_n)=\underbrace{\tilde{p}\ *_{{}_1i_g}\ (\sigma(p_1),\ldots,\sigma(p_{j-1}),}_{=:p'}\sigma(p_j),\sigma(p_{j+1}),\ldots,\sigma(p_n)) \tag{151}$$

$$=p'\ *_{{}_1i_g}\ (\sigma(p_j),\sigma(p_{j+1}),\ldots,\sigma(p_n)) \tag{152}$$

$$={}_1i_g({}_1i_g(p',\sigma(p_j)),\sigma(p_{j+1}))\ *_{{}_1i_g}\ (\sigma(p_{j+2}),\ldots,\sigma(p_n)) \tag{153}$$

$$={}_1i_g({}_1i_g(p',\sigma(p_{j+1})),\sigma(p_j))\ *_{{}_1i_g}\ (\sigma(p_{j+2}),\ldots,\sigma(p_n)) \tag{154}$$

$$=\tilde{p}\ *_{{}_1i_g}\ (\sigma(p_1),\ldots,\sigma(p_{j-1}),\sigma(p_{j+1}),\sigma(p_j),\sigma(p_{j+2}),\ldots,\sigma(p_n)) \tag{155}$$

$$\text{using bubble sort} \tag{156}$$

$$=\tilde{p}\ *_{{}_1i_g}\ (p_1,\ldots,p_n). \tag{157}$$

□

Proof of Lemma 3. We need to prove that the function $copy_{\tilde{g}}^{ALL}$ does not induce overlapping communications.

Overlapping means that two processes communicate either to the same other third process or to each other. The potential overlapping communications are avoided by letting only some processes communicate simultaneously. The communicating processes are distributed in a checkerboard-like structure. Between two communicating processes are always two passive processes. Since each process communicates only with its direct neighbor processes, there cannot be an overlapping communication. To prove that, we need to prove

$$\begin{aligned} &\forall k, l', l'' \in \left\{1, \ldots, 3^d\right\} \ \forall j', j'' \in \left\{1, \ldots, {}^{k}N_{cell}^{*}\right\} : \\ &j' \neq j'' \rightarrow \ \beta(\gamma(k, j'), l') \neq \beta(\gamma(k, j''), l'') \ \vee \ \beta(\gamma(k, j'), l') = undef. , \end{aligned} \tag{158}$$

where k is the number of the checkerboard-like pattern; $\gamma(k, j)$, Equation (60), is an index of a reading process; and $\beta(\gamma(k, j), l)$, Equation (61), its lth neighbor with which it communicates. Hence, two communicating processes do not have a common process with which they communicate. If β is undefined, there is no process.

Inserting the definition of γ into β results in

$$\beta(\gamma(k, j), l) = \bar{\mathrm{I}}_{\iota}^{-1}\left(\bar{\mathrm{I}}_{\iota}\left(\bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j) \cdot 3 + \bar{3}_{\iota}(k) - \bar{\underline{3}}\right)\right) + \bar{3}_{\iota}(l) - \bar{\underline{2}}\right) \tag{159}$$

$$= \bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j) \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l) - \bar{\underline{5}}\right). \tag{160}$$

From here on, we do a proof by contradiction. Therefore, we negate the statement we want to prove and derive absurdity.

Under the assumption that

$$j' \neq j'' \ \wedge \ \beta(\gamma(k, j'), l') = \beta(\gamma(k, j''), l'') \ \wedge \ \beta(\gamma(k, j'), l') \neq undef. \tag{161}$$

$\xleftarrow{(Equation\ (159))}$

$$\begin{aligned} &j' \neq j'' \\ \wedge\ &\bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}}\right) = \bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j'') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l'') - \bar{\underline{5}}\right) \\ \wedge\ &\bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}}\right) \neq undef. \end{aligned} \tag{162}$$

$\xleftarrow{Lemma\ 1}$

$$\begin{aligned} &j' \neq j'' \\ \wedge\ &\Bigg({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}} = {}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j'') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l'') - \bar{\underline{5}} \\ &\quad \vee\ \bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}}\right) = undef. \Bigg) \\ \wedge\ &\bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}}\right) \neq undef. \end{aligned} \tag{163}$$

$\longleftrightarrow$

$$\begin{aligned} &j' \neq j'' \\ \wedge\ &{}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}} = {}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j'') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l'') - \bar{\underline{5}} \\ \wedge\ &\bar{\mathrm{I}}_{\iota}^{-1}\left({}^{k}\bar{\mathrm{I}}_{\iota}^{*}(j') \cdot 3 + \bar{3}_{\iota}(k) + \bar{3}_{\iota}(l') - \bar{\underline{5}}\right) \neq undef. \end{aligned} \tag{164}$$

$\xleftarrow{Definition\ 13}$

$$j' \neq j'' \ \wedge \ \underbrace{{}^{k\bar{\mathbf{I}}^*}\iota(j')}_{\in\mathbb{N}_1^d}\cdot 3 + \underbrace{\bar{3}\iota(l')}_{\in[\bar{1},\bar{3}]} = \underbrace{{}^{k\bar{\mathbf{I}}^*}\iota(j'')}_{\in\mathbb{N}_1^d}\cdot 3 + \underbrace{\bar{3}\iota(l'')}_{\in[\bar{1},\bar{3}]} \ \wedge \ {}^{k\bar{\mathbf{I}}^*}\iota(j')\cdot 3 + \bar{3}\iota(k) + \bar{3}\iota(l') - \underline{\bar{5}} \in \mathbb{N}_1^d \cap \left[\underline{\bar{1}}, \underline{\bar{\mathbf{I}}}\right] \quad (165)$$

$\longleftrightarrow$

$$j' \neq j'' \ \wedge \ \underbrace{{}^{k\bar{\mathbf{I}}^*}\iota(j') - {}^{k\bar{\mathbf{I}}^*}\iota(j'')}_{\in\mathbb{Z}^d}\cdot 3 = \underbrace{\bar{3}\iota(l'') - \bar{3}\iota(l')}_{\in[-\bar{2},\bar{2}]} \ \wedge \ {}^{k\bar{\mathbf{I}}^*}\iota(j')\cdot 3 + \bar{3}\iota(k) + \bar{3}\iota(l') - \underline{\bar{5}} \in \mathbb{N}_1^d \cap \left[\underline{\bar{1}}, \underline{\bar{\mathbf{I}}}\right] \quad (166)$$

$\xleftarrow{Definition\ 13}$

$$j' \neq j'' \ \wedge \ \underbrace{{}^{k\bar{\mathbf{I}}^*}\iota(j') - {}^{k\bar{\mathbf{I}}^*}\iota(j'')}_{\in\mathbb{Z}^d}\cdot 3 = \underbrace{\bar{3}\iota(l'') - \bar{3}\iota(l')}_{\in[-\bar{2},\bar{2}]} = \underline{\bar{0}} \ \wedge \ {}^{k\bar{\mathbf{I}}^*}\iota(j')\cdot 3 + \bar{3}\iota(k) + \bar{3}\iota(l') - \underline{\bar{5}} \in \mathbb{N}_1^d \cap \left[\underline{\bar{1}}, \underline{\bar{\mathbf{I}}}\right] \quad (167)$$

$\xleftarrow{Lemma\ 1}$

$$j' \neq j'' \ \wedge \ j' = j'' \ \wedge \ l' = l'' \ \wedge \ {}^{k\bar{\mathbf{I}}^*}\iota(j')\cdot 3 + \bar{3}\iota(k) + \bar{3}\iota(l') - \underline{\bar{5}} \in \mathbb{N}_1^d \cap \left[\underline{\bar{1}}, \underline{\bar{\mathbf{I}}}\right] \quad (168)$$

$$\lightning\,(\text{contradiction}) \quad (169)$$

$\longrightarrow$

$$\begin{aligned} &\forall k, l', l'' \in \left\{1, \ldots, 3^d\right\} \ \forall j', j'' \in \left\{1, \ldots, {}^kN^*_{cell}\right\} : \\ &j' \neq j'' \rightarrow \ \beta(\gamma(k, j'), l') \neq \beta(\gamma(k, j''), l'') \ \vee \ \beta(\gamma(k, j'), l') = undef. \end{aligned} \quad (170)$$

□

Proof of Lemma 4. We need to prove, after the $copy^{ALL}_{\tilde{g}}$ function, that the storages of all processes contain the particles of the corresponding cells and all their neighbor particles. Hence, first, we need to prove that each process executes the $copy^{ALL}_{\tilde{g}}$ function and, second, that on a process after the $copy^{ALL}_{\tilde{g}}$ function all neighbor particles of all particles of the center storage compartment are in the storage.

First, the idea is to prove that $\gamma(k, j)$ is bijective and the codomain is the set of all indices of the processes $\{1, \ldots, N_{cell}\}$:

$$\gamma : \left\{1, \ldots, 3^d\right\} \times \left\{1, \ldots, {}^kN^*_{cell}\right\} \rightarrow \{1, \ldots, N_{cell}\} \text{ is bijective.} \quad (171)$$

We define a helper function $\underline{\gamma}^*$, prove it is bijective, and transform it into γ.
We declare

$$\underline{\gamma}^* : \{1, \ldots, 3\}^d \times \mathbb{N}^d \cap \left[\underline{\bar{1}}, \ {}^k\underline{\bar{\mathbf{I}}}^*\right] \rightarrow \mathbb{N}^d \cap \left[\underline{\bar{1}}, \underline{\bar{\mathbf{I}}}\right] \quad (172)$$

and define

$$\underline{\gamma}^*(\underline{k}, \underline{j}) := \underline{k} + (\underline{j} - \underline{\bar{1}}) \cdot 3. \quad (173)$$

One element of $\underline{\gamma}^*$ is $< \underline{\gamma}^*(\underline{k}, \underline{j}) >_w$ and we can rewrite it by

$$\left\langle \underline{\gamma}^*(\underline{k}, \underline{j}) \right\rangle_w = 1 + (k_w - 1) + (j_w - 1) \cdot 3 = \underbrace{{}^k\underline{\bar{\mathbf{I}}}_w \iota^{-1}\left(\begin{pmatrix} k_w \\ j_w \end{pmatrix}\right)}_{\text{is bijective } (\iota \text{ is bijective})}, \quad (174)$$

where ${}^{k}\underline{\bar{\mathbf{J}}}_w := \begin{pmatrix} 3 \\ {}^{k}I_w^* \end{pmatrix}$. The domain of the arguments j and k depend on each other. Hence, it is not obvious that $\underline{\gamma}^*$ is not leaving the codomain $\mathbb{N}^d \cap [\underline{\mathbf{1}}, \underline{\bar{\mathbf{I}}}]$. $\underline{\gamma}^*$ is monotone. Therefore, it is sufficient to prove that $\underline{\gamma}^*$ is in the codomain for the smallest and largest arguments. The minimal value of $\underline{\gamma}^*$ is

$$\min\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) = 1 + (1-1)\cdot 3 = \underline{1}. \tag{175}$$

This is in the codomain. The maximal value of $\underline{\gamma}^*$ is

$$\max\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) = k_w^{\max} + \left({}^{k^{\max}}I_w^* - 1\right)\cdot 3. \tag{176}$$

Using the definition of ${}^{k_{\max}}\underline{\bar{\mathbf{I}}}^*$, Equation (58), leads to

$$\max\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) = k_w^{\max} + \left(\left\lfloor \tfrac{1}{3}\left(I_w - k_w^{\max} + 3\right)\right\rfloor - 1\right)\cdot 3. \tag{177}$$

We substitute

$$I_w - k_w^{\max} + 3 =: \underbrace{T'}_{\in\mathbb{N}}\cdot 3 + \underbrace{T''}_{\in\{0,1,2\}} \tag{178}$$

and get

$$\max\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) = k_w^{\max} + \left(\left\lfloor \tfrac{3T'}{3} + \tfrac{T''}{3}\right\rfloor - 1\right)\cdot 3 \tag{179}$$

$$= k_w^{\max} + (T'-1)\cdot 3. \tag{180}$$

We rearrange the substitution to

$$3T' = I_w - k_w^{\max} + 3 - T'' \tag{181}$$

and plug it in:

$$\max\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) = k_w^{\max} + I_w - k_w^{\max} + 3 - T'' - 3 \tag{182}$$

$$= I_w - \underbrace{T''}_{\in\{0,1,2\}}. \tag{183}$$

This means

$$\max\left(\left\langle \underline{\gamma}^*(\underline{k}, \underline{j})\right\rangle_w\right) \le I_w. \tag{184}$$

It remains to be shown that I_w can be reached. This is only possible if $T'' = 0$. Hence, we need to prove

$$\exists k_w \in \{1,2,3\} : T'' = 0. \tag{185}$$

From $T'' = 0$ it follows that

$$\frac{3T' + T''}{3} = \frac{I_w + 3 - k_w^{\max}}{3} = n \in \mathbb{N} \tag{186}$$

$$\longrightarrow$$

$$n + \frac{k_w^{\max} - 1}{3} = \frac{I_w + 2}{3}. \tag{187}$$

Taking the $floor$ on both sides leads to

$$\left\lfloor n + \underbrace{\frac{k_w^{\max} - 1}{3}}_{\in\{0,1,2\}}\right\rfloor = \left\lfloor \frac{I_w + 2}{3}\right\rfloor \tag{188}$$

$$\longrightarrow$$

$$n = \left\lfloor \frac{I_w + 2}{3} \right\rfloor. \tag{189}$$

Inserting this into Equation (187) leads to

$$k_w = ((I_w + 3) - 1) - \left\lfloor \tfrac{(I_w+3)-1}{3} \right\rfloor \cdot 3 + 1 = \left\langle \overline{\underline{\mathbf{J}}} \iota(I_w + 3) \right\rangle_1 \in \{1, 2, 3\}, \tag{190}$$

where $\overline{\underline{\mathbf{J}}} = (3, \ldots)^{\mathrm{T}}$. From this, we can conclude

$$\forall w \in \{1, \ldots, d\} : \left\langle \underline{\gamma}^* \right\rangle_w \text{ is bijective} \tag{191}$$

$$\longrightarrow$$

$$\underline{\gamma}^* \text{ is bijective.} \tag{192}$$

Now we need to transform $\underline{\gamma}^*$ to γ. We substitute $\underline{k}$ and $\underline{j}$ by

$$\underline{k} = {}^{\overline{3}}\iota(k), \qquad \underline{j} = {}^{k\overline{\mathbf{I}}^*}\iota(j). \tag{193}$$

We know that ι is bijective (Lemma 1, Definition 13) and therefore,

$$k \in \{1, \ldots, 3^d\}, \qquad j \in N^d \cap \left[\overline{\underline{\mathbf{1}}}, {}^{k}\overline{\underline{\mathbf{I}}}^* \right]. \tag{194}$$

We also know ι^{-1} is bijective. Hence,

$$\begin{aligned} {}^{\overline{\mathbf{I}}}\iota^{-1}\left(\underline{\gamma}^* \left({}^{\overline{3}}\iota(k), {}^{k\overline{\mathbf{I}}^*}\iota(j) \right) \right) &= {}^{\overline{\mathbf{I}}}\iota^{-1}\left({}^{\overline{3}}\iota(k) + \left({}^{k\overline{\mathbf{I}}^*}\iota(j) - \overline{\underline{\mathbf{1}}} \right) \cdot 3 \right) && (195) \\ &= \gamma(k, j). && (196) \end{aligned}$$

This means γ is bijective, and the $copy^{ALL}_{\tilde{g}}$ function is executed for all processes.

Second, it remains to be proven that the neighbor particles of all particles of the center storage compartment are in the storage. The function $copy_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l)$, Equation (70), copies the central $(\frac{3^d+1}{2})$ storage compartment form the $\beta(w, l)$-th process to the lth storage compartment on the wth process. We start from the vectorial index view to find the corresponding cell/process where the particles are for the lth storage compartment on the wth process. The vectorial index of the wth process is ${}^{\overline{\mathbf{I}}}\iota(w)$ and the vectorial index of the lth storage compartment is ${}^{\overline{3}}\iota(l) \in \{1, 2, 3\}$. The storage center compartment $l = (\frac{3^d+1}{2})$ corresponds to the cell belonging to the process. The rest of the storage compartments should represent the surrounding cells. Therefore, we need to shift the storage compartment index by -2 in all dimensions to account for this. Hence, the vectorial index of the corresponding process for the lth storage compartment of the wth process is

$${}^{\overline{\mathbf{I}}}\iota(w) + {}^{\overline{3}}\iota(l) - \overline{\underline{\mathbf{2}}}. \tag{197}$$

Transforming the vectorial index to a scalar index results in

$${}^{\overline{\mathbf{I}}}\iota^{-1}\left({}^{\overline{\mathbf{I}}}\iota(w) + {}^{\overline{3}}\iota(l) - \overline{\underline{\mathbf{2}}} \right) = \beta(w, l). \tag{198}$$

Hence, the $\beta(w, l)$-th process has the corresponding particles in its center storage compartment. The storage gets the particles from the surrounding cells by the *copy* function from the other processes. With this, we can derive the domain area that the storage covers. The vectorial indices of the cells that belong to the domain are

$$\left\{ {}^{\overline{\mathbf{I}}}\iota(\beta(w, l)) : l \in \{1, \ldots, 3^d\} \right\} = \left[{}^{\overline{\mathbf{I}}}\iota(w) - \overline{\underline{\mathbf{1}}},\ {}^{\overline{\mathbf{I}}}\iota(w) + \overline{\underline{\mathbf{1}}} \right] \cap \left[\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}} \right] \cap \mathbb{N}^d. \tag{199}$$

Per Equation (63), the particles that belong to the wth cell are

$$p_j = (\ldots, x_j, \ldots) \in \mathbf{p}^1_w : \ \bar{\mathbb{I}}_\iota(w) = \left\lfloor \frac{1}{r_c}(\underline{x}_j - \underline{D}_{\min}) \right\rfloor + \overline{\mathbf{1}} \tag{200}$$

$$\longrightarrow$$

$$p_j \in \mathbf{p}^1_w : \ \bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} = \left\lfloor \frac{1}{r_c}(\underline{x}_j - \underline{D}_{\min}) \right\rfloor . \tag{201}$$

$$\longrightarrow$$

$$p_j \in \mathbf{p}^1_w : \ \frac{1}{r_c}(\underline{x}_j - \underline{D}_{\min}) \in \left[\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}}, \ \bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} + \overline{\mathbf{1}} \right) \tag{202}$$

$$\longrightarrow$$

$$p_j \in \mathbf{p}^1_w : \ \underline{x}_j \in \left[\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min}, \ \bar{\mathbb{I}}_\iota(w) \cdot r_c + \underline{D}_{\min} \right) . \tag{203}$$

Taking all cells/compartments, Equation (199), of the storage of the wth process leads to

$$p_j \in \bigcirc_{l=1}^{3^d} \langle {}^{\text{PROC}_w}\underline{\mathbf{p}}^1 \rangle_l : \ \underline{x}_j \in \left[\left(\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} \right) - \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min}, \ \left(\bar{\mathbb{I}}_\iota(w) + \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min} \right) \cap \left[\underline{D}_{\min}, \underline{D}_{\max} \right) , \tag{204}$$

where $[\overline{\mathbf{1}}, \overline{\mathbf{I}}]$ translates to $\left[\underline{D}_{\min}, \underline{D}_{\max} \right)$, Equation (56):

$$\longrightarrow$$

$$p_j \in \bigcirc_{l=1}^{3^d} \langle {}^{\text{PROC}_w}\underline{\mathbf{p}}^1 \rangle_l : \ \underline{x}_j \in \left[\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{2}} \right) \cdot r_c + \underline{D}_{\min}, \ \left(\bar{\mathbb{I}}_\iota(w) + \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min} \right) \cap \left[\underline{D}_{\min}, \underline{D}_{\max} \right) . \tag{205}$$

It remains to be proven that the neighbor particles of all particles of the central storage compartment of the wth process are in the storage. Hence, the domain area covered by the storage includes the area covered by the neighborhood function u. Let $p_k \in \mathbf{p}^1$ and $p_j \in \left\langle \mathbf{p}^1 \right\rangle_{u(g, \mathbf{p}^1, k)}$, Equation (48); then

$$|\underline{x}_k - \underline{x}_j| \leq r_c \tag{206}$$

$$\longrightarrow$$

$$\underline{x}_j \in \left[\underline{x}_k - \overline{\mathbf{1}} \cdot r_c, \ \underline{x}_k + \overline{\mathbf{1}} \cdot r_c \right] . \tag{207}$$

Let now $p_k \in \mathbf{p}^1_w$; then we know, Equation (203):

$$\underline{x}_k \in \left[\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min}, \ \bar{\mathbb{I}}_\iota(w) \cdot r_c + \underline{D}_{\min} \right) \tag{208}$$

$$\longrightarrow$$

$$\underline{x}_j \in \left[\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min} - \overline{\mathbf{1}} \cdot r_c, \ \bar{\mathbb{I}}_\iota(w) \cdot r_c + \underline{D}_{\min} + \overline{\mathbf{1}} \cdot r_c \right) \cap \left[\underline{D}_{\min}, \underline{D}_{\max} \right) \tag{209}$$

$$\longrightarrow$$

$$\underline{x}_j \in \left[\left(\bar{\mathbb{I}}_\iota(w) - \overline{\mathbf{2}} \right) \cdot r_c + \underline{D}_{\min}, \ \left(\bar{\mathbb{I}}_\iota(w) + \overline{\mathbf{1}} \right) \cdot r_c + \underline{D}_{\min} \right) \cap \left[\underline{D}_{\min}, \underline{D}_{\max} \right) . \tag{210}$$

This is the same as Equation (205), which means that the neighbor particles of all particles of the center storage compartment of the wth process are all in the storage of the wth process, and this is true for all processes. □

Proof of Lemma 5. We need to prove that the function $dist^{ALL}_{(\tilde{g}, \mathcal{G}^1)}$ places for each process the particles from the center storage compartments only to the other storage compartments of the same processes. Hence, it does not try to place them in a not-existing storage compartment. Hence,

$$\forall w \in \{1, \ldots, N_{cell}\} \forall p^1_j \in \left\langle \left\langle \mathcal{P}^1 \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} : p^2_j := \left\langle \left\langle \left\langle step^{ALL}_{\mathcal{G}^1} \left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1) \right) \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j$$

$$\longrightarrow \ \alpha = {}^{\overline{3}}\iota^{-1} \left(\left\lfloor \frac{1}{r_c}(\underline{x}^2_j - \underline{D}_{\min}) \right\rfloor - \bar{\mathbb{I}}_\iota(w) + \overline{\mathbf{3}} \right) \in \{1, \ldots, 3^d\}. \tag{211}$$

All particles have a position $\underline{x} \in \mathbb{R}^d$; hence, $p_j^1 = (\ldots, \underline{x}_j^1, \ldots)$ and $p_j^2 = (\ldots, \underline{x}_j^2, \ldots)$. The condition in Equation (50) restricts the movement of the particles to be smaller than r_c, meaning for $\underline{\Delta x} := \underline{x}_j^2 - \underline{x}_j^1$:

$$\left|\underline{\Delta x}\right| \leq r_c. \tag{212}$$

$$\xrightarrow{Definition\ 5}$$

$$\left|\underline{\Delta x}\right| = \left|\begin{pmatrix} \Delta x_1 \\ \vdots \\ \Delta x_d \end{pmatrix}\right| = \sqrt{\Delta x_1^2 + \cdots + \Delta x_d^2} \leq r_c \tag{213}$$

$$\longrightarrow$$

$$\forall k \in \{1, \ldots, d\} : r_c \leq \Delta x_k \leq r_c \tag{214}$$

$$\longrightarrow$$

$$\forall k \in \{1, \ldots, d\} : \Delta x_k \in [-r_c, r_c]. \tag{215}$$

Using this, we can rewrite α from

$$\alpha = \overline{\overline{3}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min})\right\rfloor - \overline{\overline{\mathbb{I}}}\iota(w) + \overline{\underline{3}}\right) \tag{216}$$

to

$$\alpha = \overline{\overline{3}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^1 + \underline{\Delta x} - \underline{D}_{\min})\right\rfloor - \overline{\overline{\mathbb{I}}}\iota(w) + \overline{\underline{3}}\right). \tag{217}$$

We put the index for the dimension as pre-subscript:

$$\alpha = \overline{\overline{3}}\iota^{-1}\left(\begin{pmatrix} \left\lfloor \frac{1}{r_c}({}_1x_j^1 + {}_1\Delta x - {}_1D_{\min})\right\rfloor - \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_1 + 3 \\ \vdots \\ \left\lfloor \frac{1}{r_c}({}_dx_j^1 + {}_d\Delta x - {}_dD_{\min})\right\rfloor - \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_d + 3 \end{pmatrix}\right). \tag{218}$$

We know from Equation (63) that

$$\forall p_j^1 \in \mathbf{p}^1 : p_j^1 \in \mathbf{p}_w^1 \text{ with } \quad w = \overline{\overline{\mathbb{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^1 - \underline{D}_{\min})\right\rfloor + \overline{\underline{1}}\right). \tag{219}$$

Taking this into α we get

$$\alpha = \overline{\overline{3}}\iota^{-1}\left(\underbrace{\begin{pmatrix} \underbrace{\left\lfloor \frac{{}_1x_j^1 - {}_1D_{\min}}{r_c} + \underbrace{\frac{{}_1\Delta x}{r_c}}_{\in[-1,1]}\right\rfloor}_{\in\left\{\left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_1 - 2,\ \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_1 - 1,\ \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_1\right\}} - \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_1 + 3 \\ \vdots \\ \underbrace{\left\lfloor \frac{{}_dx_j^1 - {}_dD_{\min}}{r_c} + \underbrace{\frac{{}_d\Delta x}{r_c}}_{\in[-1,1]}\right\rfloor}_{\in\left\{\left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_d - 2,\ \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_d - 1,\ \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_d\right\}} - \left\langle \overline{\overline{\mathbb{I}}}\iota(w)\right\rangle_d + 3 \end{pmatrix}}_{\in\mathbb{N}\cap\left[\overline{\underline{1}},\overline{\underline{3}}\right]}\right) \tag{220}$$

$$\longrightarrow$$

$$\underline{\alpha \in \{1, \ldots, 3^d\}}. \tag{221}$$

Hence, under the condition in Equation (63), no particle leaves the storage compartments of its process through the function $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$. □

Proof of Lemma 6. We need to prove that the function $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ places particles only inside storage compartments that represent cells inside the domain. Hence, processes at the domain's border do not have particles in their outer storage compartments. Let

$$\underline{w} = (w_1, \ldots, w_d)^{\mathrm{T}} := \bar{\underline{\mathrm{I}}}\iota(w) \tag{222}$$

$$\underline{\alpha} = (\alpha_1, \ldots, \alpha_d)^{\mathrm{T}} := \left\lfloor \tfrac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min}) \right\rfloor - \underline{w} + \underline{\mathbf{3}}. \tag{223}$$

Then

$$\forall w \in \{1, \ldots, N_{cell}\}\ \forall p_j^2 := \left\langle \left\langle \left\langle step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right)\right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j :$$

$$(k \in \{1, \ldots, d\} \wedge w_k = 1) \longrightarrow \alpha_k \in \{2, 3\} \wedge (k \in \{1, \ldots, d\} \wedge w_k = I_k) \longrightarrow \alpha_k \in \{1, 2\}. \tag{224}$$

We do a proof by contradiction. First, we prove the statement with $w_k = 1$. Assuming

$$\exists k \in \{1, \ldots, d\} : w_k = 1 \wedge \alpha_k = 1, \tag{225}$$

then

$$\alpha_k = 1 = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) \right\rfloor - 1 + 3. \tag{226}$$

$$\longrightarrow$$

$$-1 = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) \right\rfloor \tag{227}$$

$$\longrightarrow$$

$$\tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) = [-1, 0) \tag{228}$$

$$\longrightarrow$$

$${}_k x_j^2 = [-r_c + {}_k D_{\min}, 0) \quad \lightning. \tag{229}$$

This contradicts the condition that no particle leaves the domain, Equation (49). Hence, in this case $\alpha_k \in \{2, 3\}$. Now we prove the second statement with $w_k = I_k$. Assuming

$$\exists k \in \{1, \ldots, d\} : w_k = I_k \wedge \alpha_k = 3, \tag{230}$$

then

$$\alpha_k = 3 = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) \right\rfloor - I_k + 3 \tag{231}$$

$$\longrightarrow$$

$$I_k = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) \right\rfloor \tag{232}$$

$$\longrightarrow$$

$$\tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{\min}) = [I_k, I_k + 1) \tag{233}$$

$$\longrightarrow$$

$${}_k x_j^2 = [I_k r_c + {}_k D_{\min}, I_k r_c + {}_k D_{\min} + r_c) \quad \lightning \tag{234}$$

$$\xrightarrow{def.\ \bar{\underline{\mathrm{I}}}\ (Equation\ (56))}$$

$${}_k x_j^2 > {}_k D_{\max}) \quad \lightning. \tag{235}$$

This contradicts the condition that no particle leaves the domain, Equation (49). Hence, in this case $\alpha_k \in \{1, 2\}$. Taking both cases together, the function $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ places particles only inside storage compartments that represent cells inside the domain. □

Proof of Lemma 7. $\overline{\mathcal{P}}^1 := step^{ALL}_{\mathcal{G}^1}(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1))$ does not necessarily fulfill the condition, Equation (63), to reiterate $step^{ALL}_{\mathcal{G}^2}(copy^{ALL}_{\tilde{g}}(\overline{\mathcal{P}}^1))$. The function $interact_g$ could miss interaction due to incomplete neighborhoods. Therefore, the particles in $\overline{\mathcal{P}}^1$ need to be redistributed such that the requirement in Equation (63) is fulfilled. The decomposition of the initial permutated particle tuple is stored in the center storage compartment of the processes. Hence, the condition in Equation (63) can be rewritten for all state transition steps as

$$\forall p^t_j \in \bigcirc_{v=1}^{N_{cell}} \left\langle \left\langle \mathcal{P}^t \right\rangle_v \right\rangle_{\frac{3^d+1}{2}} : p^t_j \in \left\langle \left\langle \mathcal{P}^t \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}, \tag{236}$$

where

$$w = \underline{\overline{\mathrm{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}^t_j - \underline{D}_{\min})\right\rfloor + \underline{\overline{\mathbf{1}}}\right). \tag{237}$$

We need to prove that the two functions $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ and $collect^{ALL}_{\tilde{g}}$ achieve this.

We prove this by induction. Regarding the base case, we know that the condition in Equations (91) and (92) is true for $\mathcal{P}^1$ by the definition in Equation (63).

Regarding the induction step, we need to prove that if $\mathcal{P}^t$ fulfills the condition in Equations (91) and (92), then

$$\mathcal{P}^{t+1} = collect^{ALL}_{\tilde{g}}\left(dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}\left(step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^t)\right)\right)\right) \tag{238}$$

fulfills also the condition.

$dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$, Equation (79), empties all storage compartments except the center storage compartment and then distributes all particles of the center storage compartments according to their position to the other storage compartments of each process. α gives the index of the new storage compartment for each particle. It is trivial that it considers all particles and all processes since it simply iterates over them.

$collect^{ALL}_{\tilde{g}}$, Equation (82), copies for each process its particles from the other processes corresponding to storage compartments. It uses the same strategy as the function $copy^{ALL}_{\tilde{g}}$ to avoid overlapping conditions; hence, Lemma 3 is also valid for the $collect^{ALL}_{\tilde{g}}$ function. The $collect^{ALL}_{\tilde{g}}$ function takes the particles from the $\underline{\overline{3}}\iota^{-1}(\underline{\overline{4}}-\underline{\overline{3}}\iota(l))$-th storage compartment of the $\beta(w,l)$-th process and stores them in the center storage compartment of the wth process.

We need to prove that a particle from each storage compartment of all processes ends up in the suitable process's central storage compartment according to the condition in Equations (91) and (92). We choose without restricting generality a particle p from the $\underline{\overline{3}}\iota^{-1}(\underline{\overline{4}} - \underline{\overline{3}}\iota(l))$-th storage compartment of the $\beta(w,l)$-th process. Hence,

$$p = (\ldots, x, \ldots) \in \left\langle \left\langle step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^t)\right)\right\rangle_{\beta(w,l)}\right\rangle_{\underline{\overline{3}}\iota^{-1}\left(\underline{\overline{4}}-\underline{\overline{3}}\iota(l)\right)}. \tag{239}$$

Combining this with the distribution of the $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ function, we get

$$\underbrace{\alpha}_{\stackrel{!}{=}\underline{\overline{3}}\iota^{-1}\left(\underline{\overline{4}}-\underline{\overline{3}}\iota(l)\right)} := \underline{\overline{3}}\iota^{-1}\left(\left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor - \underline{\overline{\mathrm{I}}}\iota(\underbrace{j}_{\stackrel{!}{=}\beta(w,l)}) + \underline{\overline{3}}\right), \tag{240}$$

where α is set to $\underline{\overline{3}}\iota^{-1}(\underline{\overline{4}} - \underline{\overline{3}}\iota(l))$, the index of the storage compartment from which the particles are collected, and j is set to $\beta(w,l)$, the index of the process from which it is collected. Hence, our

particle is in that storage compartment of that process and gets distributed by the mechanism of the $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ function. We put the indices in and transform

$$ {}^{\overline{3}}\iota^{-1}\left(\underline{\overline{4}} - {}^{\overline{3}}\iota(l)\right) = {}^{\overline{3}}\iota^{-1}\left(\left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor - {}^{\overline{\mathbb{I}}}\iota(\beta(w,l)) + \underline{\overline{3}}\right) \tag{241} $$

$$ \xrightarrow{Lemma\ 1} $$

$$ \underline{\overline{4}} - {}^{\overline{3}}\iota(l) = \left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor - {}^{\overline{\mathbb{I}}}\iota(\beta(w,l)) + \underline{\overline{3}} \tag{242} $$

$$ \xrightarrow{def.\ of\ \beta\ (Equation\ (61))} $$

$$ \underline{\overline{4}} - {}^{\overline{3}}\iota(l) = \left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor - {}^{\overline{\mathbb{I}}}\iota\left({}^{\overline{\mathbb{I}}}\iota^{-1}\left({}^{\overline{\mathbb{I}}}\iota(w) + {}^{\overline{3}}\iota(l) - \underline{\overline{2}}\right)\right) + \underline{\overline{3}} \tag{243} $$

$$ \xrightarrow{Lemma\ 1} $$

$$ \underline{\overline{4}} - {}^{\overline{3}}\iota(l) = \left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor - {}^{\overline{\mathbb{I}}}\iota(w) - {}^{\overline{3}}\iota(l) + \underline{\overline{2}} + \underline{\overline{3}} \tag{244} $$

$$ \longrightarrow $$

$$ {}^{\overline{\mathbb{I}}}\iota(w) = \left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor + \underline{\overline{1}} \tag{245} $$

$$ \xrightarrow{Lemma\ 1} $$

$$ w = {}^{\overline{\mathbb{I}}}\iota^{-1}\left(\left\lfloor \tfrac{1}{r_c}(\underline{x} - \underline{D}_{\min})\right\rfloor + \underline{\overline{1}}\right). \tag{246} $$

The functions $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ and $collect^{ALL}_{\tilde{g}}$ put our particle p in the central storage compartment of the wth process:

$$ p \in \left\langle\left\langle collect^{ALL}_{\tilde{g}}\left(dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}\left(step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^t)\right)\right)\right)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}. \tag{247} $$

Hence,

$$ p \in \left\langle\left\langle \mathcal{P}^{t+1}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}} \tag{248} $$

fulfills the condition in Equations (91) and (92). □

B Derivation of the Bounds on Time Complexity and Parallel Scalability

Following [24], an upper bound on the time complexity of the interact function $\tau_{i(g,p',p'')}$ is

$$ \forall g \in \{g^1,\ldots,g^T\}, p', p'' \in \bigcirc_{w=1}^{T} \mathbf{p}^w\ \exists \tilde{g} \in \{g^1,\ldots,g^T\}, \tilde{p}', \tilde{p}'' \in \bigcirc_{w=1}^{T} \mathbf{p}^w :\ \tau_{i(g,p',p'')} \leq \tau_{i(\tilde{g},\tilde{p}',\tilde{p}'')} =: \tau_i\,. \tag{249} $$

Similarly, an upper bound on the time complexity of the evolve function $\tau_{e(g,p)}$ is

$$ \forall g \in \{g^1,\ldots,g^T\}, p \in \bigcirc_{w=1}^{T} \mathbf{p}^w\ \exists \tilde{g} \in \{g^1,\ldots,g^T\}, \tilde{p} \in \bigcirc_{w=1}^{T} \mathbf{p}^w :\ \tau_{e(g,p)} \leq \tau_{e(\tilde{g},\tilde{p})} =: \tau_e\,. \tag{250} $$

An upper bound on the time complexity of the evolve function of the global variable $\tau^{\circ}_{e(g)}$ is

$$ \forall g \in \{g^1,\ldots,g^T\}\ \exists \tilde{g} \in \{g^1,\ldots,g^T\} :\quad \tau^{\circ}_{e(g)} \leq \tau^{\circ}_{e(\tilde{g})} =: \tau^{\circ}_{e}\,. \tag{251} $$

An upper bound on the time complexity of the stopping condition function $\tau_{f(g)}$ is

$$ \forall g \in \{g^1,\ldots,g^T\}\ \exists \tilde{g} \in \{g^1,\ldots,g^T\} :\quad \tau_{f(g)} \leq \tau_{f(\tilde{g})} =: \tau_f\,. \tag{252} $$

An upper bound on the time complexity of the neighborhood function $\tau_{u([g,\mathbf{p}],j)}$ is

$$ \forall g \in \{g^1,\ldots,g^T\}, \mathbf{p} \in \{\mathbf{p}^1,\ldots,\mathbf{p}^T\}, j \in \{1,\ldots,|\mathbf{p}|\}\ \exists \tilde{g} \in \{g^1,\ldots,g^T\}, \tilde{\mathbf{p}} \in \{\mathbf{p}^1,\ldots,\mathbf{p}^T\}, \tilde{j} \in \{1,\ldots,|\tilde{\mathbf{p}}|\} : \tau_{u([g,\mathbf{p}],j)} \leq \tau_{u([\tilde{g},\tilde{\mathbf{p}}],\tilde{j})} =: \tau_u\,. \tag{253} $$

An upper bound on the size of the neighborhood $|u([g,\mathbf{p}],j)|$ is

$$\forall g \in \{g^1,\ldots,g^T\}, \mathbf{p} \in \{\mathbf{p}^1,\ldots,\mathbf{p}^T\}, j \in \{1,\ldots,|\mathbf{p}|\}\ \exists \tilde{g} \in \{g^1,\ldots,g^T\}, \tilde{\mathbf{p}} \in \{\mathbf{p}^1,\ldots,\mathbf{p}^T\}, \tilde{j} \in \{1,\ldots,|\tilde{\mathbf{p}}|\} : \\ |u([g,\mathbf{p}],j)| \leq |u([\tilde{g},\tilde{\mathbf{p}}],\tilde{j})| =: \varsigma_u\,. \quad (254)$$

The time complexity of calculating $\bar{\underline{\mathbf{I}}}_\iota$ and $\bar{\underline{\mathbf{I}}}_\iota^{-1}$ is in $O(d)$. Therefore, the time complexity of the index transformation functions is bound by Cd, where C is a constant. Hence, the time complexity of β,$\tau_{\beta(t,l)}$ is bound by

$$\forall\, \bar{\underline{\mathbf{I}}} \in \mathbb{N}_{>0}^d, \forall t \in \{1,\ldots,N_{\text{cell}}\},\ l \in \{1,\ldots,3^d\}\ \exists C_\beta \in \mathbb{R} :\ \tau_{\beta(t,l)} \leq C_\beta d\,. \quad (255)$$

The time complexity of γ,$\tau_{\gamma(t,l)}$ is bound by

$$\forall\, \bar{\underline{\mathbf{I}}} \in \mathbb{N}_{>0}^d, \forall k \in \{1,\ldots,3^d\},\ j \in \{1,\ldots,{}^kN_{\text{cell}}^*\}\ \exists C_\gamma \in \mathbb{R} :\ \tau_{\gamma(t,l)} \leq C_\gamma d\,. \quad (256)$$

The time complexity of computing α is bound by

$$\forall\, \underline{D}_{\min}, \underline{D}_{\max} \in \mathbb{R}^d, \underline{x} \in \left[\underline{D}_{\min}, \underline{D}_{\max}\right],\ \forall j \in \{1,\ldots,N_{\text{cell}}\}\ \exists C_\alpha \in \mathbb{R} :\ \tau_{\bar{\underline{3}}_\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}-\underline{D}_{\min})\right\rfloor - \bar{\underline{\mathbf{I}}}_\iota(j)+\bar{\underline{3}}\right)} \leq C_\alpha d\,. \quad (257)$$

The time complexity for computing the index transformation in the *collect* function is bound by

$$\forall l \in \{1,\ldots,3^d\}\ \exists C_c \in \mathbb{R} : \tau_{\bar{\underline{3}}_\iota^{-1}\left(\bar{\underline{4}}-\bar{\underline{3}}_\iota(l)\right)} \leq C_c d\,. \quad (258)$$

We use these upper bounds to derive bounds on the time complexity of the present parallelization scheme on a sequential computer and on a distributed-memory parallel machine. In general, an upper bound on the time complexity of the state transition depends on the instance $[g^1,\mathbf{p}^1]$. For each instance, we can bound the number of particles by

$$\forall t \in \{1,\ldots,T\}\ \exists N_{\mathbf{p}}^{\max} \in \mathbb{N} : \left|\mathbf{p}^t\right| \leq N_{\mathbf{p}}^{\max}\,. \quad (259)$$

The time complexity of the sequential state transition τ_S is then bound by

$$\tau_{S([g^1,\mathbf{p}^1])} \leq T\left(N_{\mathbf{p}}^{\max}\left(\varsigma_u\left(N_{\mathbf{p}}^{\max}\right)\ \tau_i + \tau_u\left(N_{\mathbf{p}}^{\max}\right) + \tau_e\right) + \tau_f + \tau_e^\circ\right), \quad (260)$$

where the neighborhood-related terms $\varsigma_u(N_{\mathbf{p}}^{\max})$ and $\tau_u(N_{\mathbf{p}}^{\max})$ potentially depend on $N_{\mathbf{p}}^{\max}$. In the presented cell-list-based parallelization scheme, we can further bound the neighborhood function by exploiting that the neighborhood calculation is done separately on each process. Hence, only the particles in that process are taken into account. The number of particles in one cell is bound by

$$\forall t \in \{1,\ldots,T\},\ w \in \{1,\ldots,N_{\text{cell}}\}\ \exists n_{\max} \in \mathbb{N} :\ \left|\left\langle\left\langle \mathcal{P}^t\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right| \leq n_{\max}\,. \quad (261)$$

Then, the number of all particles in one process is bound by

$$\forall t \in \{1,\ldots,T\},\ w \in \{1,\ldots,N_{\text{cell}}\} :\ \left|\bigcirc_{l=1}^{3^d}\left\langle\left\langle \mathcal{P}^t\right\rangle_w\right\rangle_l\right| \leq 3^d n_{\max}\,. \quad (262)$$

The number of particles is bound, and the neighborhood function checks if a distance is smaller than r_c and verifies a function $\Omega(g,p_k,p_j)$. We assume

$$\forall g \in \{g^1,\ldots,g^T\}, \mathbf{p} \in \{\mathbf{p}^1,\ldots,\mathbf{p}^T\}, j,k \in \{1,\ldots,|\mathbf{p}|\}\ \exists C_u \in \mathbb{R} :\ \tau_{|\underline{x}_k-\underline{x}_j|\leq r_c} + \tau_{\Omega(g,p_k,p_j)} \leq dC_u\,. \quad (263)$$

Then, the time complexity and the size of the neighborhood function are bound by

$$\tau_u \leq 3^d n_{\max} d C_u, \qquad \varsigma_u \leq 3^d n_{\max}\,. \quad (264)$$

The time complexity of sequentially executing the distributed-memory parallel particle method $\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1)$ is determined by the time complexity of the functions $collect^{ALL}_{\tilde{g}}$, $dist^{ALL}_{(\tilde{g},\mathcal{G})}$, $step^{ALL}_{\mathcal{G}}$, and $copy^{ALL}_{\tilde{g}}$; the evolve function of the global variable for each cell; and the stop function. Then,

$$\begin{aligned}\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1) \quad \leq T\Bigg(\quad \tau_f + \underbrace{N_{\text{cell}}\tau_{\mathring{e}}}_{(global\ variable\ evolve)} + \underbrace{\prod_{k=1}^{3^d}{}^{k}N^*_{\text{cell}}\left(C_\gamma d + 3^d(C_\beta d + C_c d + n_{\max})\right)}_{(collect)} \\ + \underbrace{N_{\text{cell}} n_{\max}(C_\alpha d + 1)}_{(dist)} + \underbrace{N_{\text{cell}} n_{\max}\left(\tau_e + 3^d n_{\max} C_u d + 3^d n_{\max}\tau_i\right)}_{(step)} \\ + \underbrace{\prod_{k=1}^{3^d}{}^{k}N^*_{\text{cell}}\left(C_\gamma d + 3^d(C_\beta d + n_{\max})\right)}_{(copy)}\Bigg).\end{aligned} \tag{265}$$

We assume that particles are evenly distributed (i.e., the number of particles in each cell is approximately the same) and that the density of particles remains constant when increasing the number of initial particles $\mathbf{p}^1$. Therefore, the domain size has to increase proportionally. Hence, the number of cells N_{cell} increases, while $n_{\max}$ stays approximately the same. Under the assumption that all functions are then bound by constants, since they do not depend on N_{cell}, we can simplify

$$\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1) \leq T\left(C_f + N_{\text{cell}}\left(C_{\mathring{e}} + C_{dist} + C_{step}\right) + \prod_{k=1}^{3^d}{}^{k}N^*_{\text{cell}}\left(C_{collect} + C_{copy}\right)\right). \tag{266}$$

For a single processor we can further simplify by using $\prod_{k=1}^{3^d}{}^{k}N^*_{\text{cell}} = N_{\text{cell}}$ to get

$$\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1) \leq T\left(C_f + N_{\text{cell}}(C_{\mathring{e}} + C_{collect} + C_{dist} + C_{step} + C_{copy})\right). \tag{267}$$

When parallelizing it, we need to consider that a process is the smallest computational unit. The processes are distributed on a number of processors n_{CPU}. We define a "processor" (CPU) as running concurrently and operating on its own separate memory address space. For convenience we assume $\frac{N_{\text{cell}}}{3^d} \in \mathbb{N}$. Further, to avoid communication conflicts, the processes need to be distributed to the processors according to which checkerboard pattern they belong. All processes of the kth checkerboard pattern are the $\gamma(k,j)$-th processes with k fixed and $j \in \{1,\ldots,{}^{k}N^*_{\text{cell}}\}$. The processes on one processor must be either from one checkerboard pattern or the entire checkerboard pattern. We formulate the time complexity of the parallelized distributed-memory parallel particle method as

$$\begin{aligned}\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(n_{CPU}) \leq T\Big(C_f + \Xi_{calc}(N_{\text{cell}}, n_{CPU}, d)(C_{\mathring{e}} + C_{dist} + C_{step}) \\ + \Xi_{com}(N_{\text{cell}}, n_{CPU}, d)(C_{collect} + C_{copy})\Big),\end{aligned} \tag{268}$$

where

$$\Xi_{calc}(N_{\text{cell}}, n_{CPU}, d) := \begin{cases}\left\lceil \frac{3^d}{n_{CPU}} \right\rceil \frac{N_{\text{cell}}}{3^d} & \text{if} \quad n_{CPU} \in \{1,\ldots,3^d\} \\ M_2 & \text{if} \quad n_{CPU} \in \{3^d,\ldots,N_{\text{cell}}\} \\ 1 & \text{if} \quad n_{CPU} > N_{\text{cell}}\,,\end{cases} \tag{269}$$

and

$$\Xi_{com}(N_{\text{cell}}, n_{CPU}, d) := \begin{cases} N_{\text{cell}} & \text{if} \quad n_{CPU} \in \left\{1, \ldots, 3^d\right\} \\ n_1 M_1 + n_2 M_2 & \text{if} \quad n_{CPU} \in \left\{3^d, \ldots, N_{\text{cell}}\right\} \\ 3^d & \text{if} \quad n_{CPU} > N_{\text{cell}} \end{cases}, \tag{270}$$

where the number of checkerboard patterns that have more processors assigned to them is

$$n_1 := n_{CPU} \bmod 3^d, \tag{271}$$

the number of checkerboard patterns that have fewer processors assigned to them is

$$n_2 := 3^d - n_1, \tag{272}$$

the maximum number of processes per processor is

$$M_1 := \left\lceil \frac{N_{\text{cell}}}{3^d \left\lceil \frac{n_{CPU}}{3^d} \right\rceil} \right\rceil, \tag{273}$$

and for checkerboard patterns with fewer processors assigned to them, the maximum number of processes per processor is

$$M_2 := \left\lceil \frac{N_{\text{cell}}}{3^d \left\lfloor \frac{n_{CPU}}{3^d} \right\rfloor} \right\rceil. \tag{274}$$

For $n_{CPU} \in \left\{1, \ldots, 3^d\right\}$, the number of processors did not reach the number of checkerboard patterns. In this case, each checkerboard pattern is completely on a processor to avoid communication conflicts. The limiting factor for the calculation (Ξ_{calc}) is then the maximum number of entire checkerboard patterns on one processor. This is $\lceil \frac{3^d}{n_{CPU}} \rceil \frac{N_{\text{cell}}}{3^d}$, where $\lceil \frac{3^d}{n_{CPU}} \rceil$ is the maximum number of checkerboard patterns on one processor and $\frac{N_{\text{cell}}}{3^d}$ the number of processes in one checkerboard pattern. The communication (Ξ_{com}) is then sequential since a processor does the communication of each process sequentially, and the communication for each checkerboard pattern needs to be done sequentially.

For $n_{CPU} \in \{3^d, \ldots, N_{\text{cell}}\}$, a checkerboard pattern can be distributed on more than one processor. Therefore, the limiting factor for the calculation (Ξ_{calc}) is the maximum number of processes per processor, which is M_2. In M_2, the term $\frac{N_{\text{cell}}}{3^d}$ is again the number of processes in one checkerboard pattern and $\lfloor \frac{n_{CPU}}{3^d} \rfloor$ is the minimum number of processors per checkerboard pattern. The limiting factor for the communication (Ξ_{com}) is more complex since the communication is carried out for each checkerboard pattern separately, one after the other. Hence, there is a number (n_1) of checkerboard patterns that have one processor more assigned ($\lceil \frac{n_{CPU}}{3^d} \rceil$) to them and a number ($n_2$) of checkerboard pattern with less ($\lfloor \frac{n_{CPU}}{3^d} \rfloor$). Processes of the same color can communicate in parallel, but the algorithm needs to go sequentially through all colors. Hence, to sum up the maximum number of communications per processor for all checkerboard patterns, we calculate $n_1 M_1 + n_2 M_2$.

Each processor has exactly one process or no process for n_{CPU}, reaching or exceeding N_{cell}. The calculation (Ξ_{calc}) is saturated, and the limiting factor is 1. The communication (Ξ_{com}) is also saturated, and the limiting factor is the number of checkerboard patterns 3^d.

These upper bounds on the time complexities allow us to derive closed-form expressions for the bounds on the speedups for both strong scaling according to Amdahl's law [3] and weak scaling according to Gustafson's law [12].

First, the speedup of the cell-list scheme on one processor over the sequential state transition is

$$speedup_{\text{cell}}\left(N_{\mathrm{p}}^{\max}\right) := \frac{\tau_{S([g^1,\mathrm{p}^1])}}{\tau_{\tilde{S}([g^1,\mathrm{p}^1])}(1)} \tag{275}$$

$$\approx \frac{N_{\mathrm{p}}^{\max 2} C_u d + N_{\mathrm{p}}^{\max}\left(3^d n_{\max}\tau_i + \tau_e\right) + \tau_f + \tau_e^{\circ}}{N_{\mathrm{p}}^{\max}\left(3^d n_{\max} C_u d + 3^d n_{\max}\tau_i + \tau_e + \frac{\tau_e^{\circ}+C_{collect}+C_{dist}+C_{copy}}{n_{\max}}\right) + \tau_f} \tag{276}$$

$$\in \frac{O\left(N_{\mathrm{p}}^{\max 2}\right)}{O\left(N_{\mathrm{p}}^{\max}\right)} = O\left(N_{\mathrm{p}}^{\max}\right). \tag{277}$$

Second, Amdahl's law [3] provides an upper bound on the speedup of the cell-list scheme on multiple processors when the problem size is fixed for increasing processors n_{CPU} (strong scaling):

$$speedup_{\text{Amdal}}(n_{CPU}) = \frac{\tau_{\tilde{S}([g^1,\mathrm{p}^1])}(1)}{\tau_{\tilde{S}([g^1,\mathrm{p}^1])}(n_{CPU})} \tag{278}$$

$$\approx \frac{C_f + N_{\text{cell}}\,(C_e^{\circ}+C_{collect}+C_{dist}+C_{step}+C_{copy})}{C_f+\Xi_{calc}(N_{\text{cell}},n_{CPU},d)(C_e^{\circ}+C_{dist}+C_{step})+\Xi_{com}(N_{\text{cell}},n_{CPU},d)(C_{collect}+C_{copy})}. \tag{279}$$

Third, Gustafson's law [12] provides an upper bound on the speedup of the cell-list scheme on multiple processors when the ratio of problem size to process number is constant while increasing the number of processors (weak scaling); $\frac{N_{\text{cell}}}{n_{\text{CPU}}} = \text{const}$. We achieve this by setting $N_{\text{cell}} = n_{CPU} \cdot N'_{\text{cell}}$, where N'_{cell} is constant. For a perfectly fitting processor interconnect network topology, we predict a linear speedup on average with steps as visualized in Figure 7(c):

$$speedup_{\text{Gustafson}}(n_{CPU}) = \frac{\tau_{\tilde{S}([g^1,\mathrm{p}^1(n_{CPU})])}(1)}{\tau_{\tilde{S}([g^1,\mathrm{p}^1(n_{CPU})])}(n_{CPU})}$$

$$\approx \frac{C_f + n_{CPU}\cdot N_{\text{cell}}\,(C_e^{\circ}+C_{collect}+C_{dist}+C_{step}+C_{copy})}{C_f+\Xi_{calc}(n_{CPU}\cdot N_{\text{cell}},n_{CPU},d)(C_e^{\circ}+C_{dist}+C_{step})+\Xi_{com}(n_{CPU}\cdot N_{\text{cell}},n_{CPU},d)(C_{collect}+C_{copy})}. \tag{280}$$

Acknowledgments

We thank Pietro Incardona (University of Bonn) for discussions on practically used distributed-memory parallelization schemes.

References

[1] Yaser Afshar and Ivo F. Sbalzarini. 2016. A parallel distributed-memory particle method enables acquisition-rate segmentation of large fluorescence microscopy images. *PLoS One* 11, 4 (2016), e0152528. https://doi.org/10.1371/journal.pone.0152528

[2] B. J. Alder and T. E. Wainwright. 1957. Molecular dynamics simulation of hard sphere system. *J. Chem. Phys.* 27 (1957), 1208–1218.

[3] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference*. 483–485.

[4] Michael Bergdorf, Ivo F. Sbalzarini, and Petros Koumoutsakos. 2010. A Lagrangian particle method for reaction-diffusion systems on deforming surfaces. *J. Math. Biol.* 61 (2010), 649–663. https://doi.org/10.1007/s00285-009-0315-2

[5] George C. Bourantas, Bevan L. Cheeseman, Rajesh Ramaswamy, and Ivo F. Sbalzarini. 2016. Using DC PSE operator discretization in Eulerian meshless collocation methods improves their robustness in complex geometries. *Comput. Fluids* 136 (2016), 285–300.

[6] Janick Cardinale, Grégory Paul, and Ivo F. Sbalzarini. 2012. Discrete region competition for unknown numbers of connected regions. *IEEE Trans. Image Process.* 21, 8 (2012), 3531–3545.

[7] G. H. Cottet and S. Mas-Gallic. 1990. A particle method to solve the Navier-Stokes system. *Numer. Math.* 57 (1990), 805–827.

[8] P. Degond and S. Mas-Gallic. 1989. The weighted particle method for convection-diffusion equations. Part 1: The case of an isotropic viscosity. *Math. Comput.* 53, 188 (1989), 485–507.
[9] Jeff D. Eldredge, Anthony Leonard, and Tim Colonius. 2002. A general deterministic treatment of derivatives in particle methods. *J. Comput. Phys.* 180 (2002), 686–709.
[10] R. A. Gingold and J. J. Monaghan. 1977. Smoothed particle hydrodynamics—Theory and application to non-spherical stars. *Roy. Astron. Soc. Mon. Not.* 181 (1977), 375–378.
[11] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. 1995. *Limits to Parallel Computation: P-completeness Theory.* Oxford University Press. https://books.google.de/books?id=YZHnCwAAQBAJ
[12] John Gustafson. 1988. Reevaluating Amdahl's Law. *Commun. ACM* 31 (May 1988), 532–533. https://doi.org/10.1145/42411.42415
[13] Alexis Hérault, Annamaria Vicari, and Ciro Del Negro. 2008. A SPH thermal model for the cooling of a lava lake. In *Proceedings of the 3th SPHERIC Workshop.*
[14] R. W. Hockney and C. R. Jesshope. 2019. *Parallel Computers 2: Architecture, Programming and Algorithms.* CRC Press. https://books.google.de/books?id=c2ytDwAAQBAJ
[15] R. W. Hockney and J. W. Eastwood. 1988. *Computer Simulation Using Particles.* Institute of Physics Publishing.
[16] Pietro Incardona, Antonio Leo, Yaroslav Zaluzhnyi, Rajesh Ramaswamy, and Ivo F. Sbalzarini. 2019. OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers. *Comput. Phys. Commun.* 241 (2019), 155–177.
[17] Masaki Iwasawa, Ataru Tanikawa, Natsuki Hosono, Keigo Nitadori, Takayuki Muranushi, and Junichiro Makino. 2016. Implementation and performance of FDPS: A framework for developing parallel particle simulation codes. *PASJ* 68, 4 (2016), 54.
[18] Sven Karol, Tobias Nett, Jeronimo Castrillon, and Ivo F. Sbalzarini. 2018. A domain-specific language and editor for parallel particle methods. *ACM Trans. Math. Softw.* 44, 3 (2018), 34.
[19] Jonathan R. Karr, Jayodita C. Sanghvi, Derek N. Macklin, Miriam V. Gutschow, Jared M. Jacobs, Benjamin Bolival, Nacyra Assad-Garcia, John I. Glass, and Markus W. Covert. 2012. A whole-cell computational model predicts phenotype from genotype. *Cell* 150 (2012), 389–401.
[20] Wing Kam Liu, Sukky Jun, and Yi Fei Zhang. 1995. Reproducing kernel particle methods. *Int. J. Numer. Meth. Fluids* 20 (1995), 1081–1106.
[21] Susan M. Mniszewski, James Belak, Jean-Luc Fattebert, Christian F. A. Negre, Stuart R. Slattery, Adetokunbo A. Adedoyin, Robert F. Bird, Choongseok Chang, Guangye Chen, Stéphane Ethier, Shane Fogerty, Salman Habib, Christoph Junghans, Damien Lebrun-Grandié, Jamaludin Mohd-Yusof, Stan G. Moore, Daniel Osei-Kuffuor, Steven J. Plimpton, Adrian Pope, Samuel Temple Reeve, Lee Ricketson, Aaron Scheinberg, Amil Y. Sharma, and Michael E. Wall. 2021. Enabling particle applications for exascale computing platforms. *Int. J. High Perform. Comput. Appl.* 35, 6 (2021), 572–597. https://doi.org/10.1177/10943420211022829
[22] J. J. Monaghan. 2005. Smoothed particle hydrodynamics. *Rep. Prog. Phys.* 68 (2005), 1703–1759.
[23] Mark T. Nelson, William Humphrey, Attila Gursoy, Andrew Dalke, Laxmikant V. Kalé, Robert D. Skeel, and Klaus Schulten. 1996. NAMD: A parallel, object-oriented molecular dynamics program. *Int. J. Supercomput. Appl. High Perform. Comput.* 10, 4 (1996), 251–268. https://doi.org/10.1177/109434209601000401
[24] Johannes Pahlke and Ivo F. Sbalzarini. 2023. A unifying mathematical definition of particle methods. *IEEE Open J. Comput. Soc.* 4 (2023), 97–108. https://doi.org/10.1109/OJCS.2023.3254466
[25] Szilárd Páll and Berk Hess. 2013. A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput. Phys. Commun.* 184, 12 (2013), 2641–2650. https://doi.org/10.1016/j.cpc.2013.06.003
[26] J. V. W. Reynders, J. C. Cummings, M. Tholburn, P. J. Hinker, S. R. Atlas, S. Banerjee, M. Srikant, W. F. Humphrey, S. R. Karmesin, and K. Keahey. 1996. POOMA: A framework for scientific simulation on parallel architectures. In *Proceedings of the 1st International Workshop on High-level Programming Models and Supportive Environments*, A. Bode, M. Gerndt, R. G. Hackenberg, and H. Hellwagner (Eds.). Technical University of Munchen; Res. Centre Julich; Central Inst. Appl. Math.; 10th IEEE Int. Parallel Process. Symposium; IEEE Comput. Soc. Tech. Committee on Parallel Process.; ACM SIGARCH, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 41–49.
[27] Ivo F. Sbalzarini. 2010. Abstractions and middleware for petascale computing and beyond. *International Journal of Distributed Systems and Technologies* 1 (April 2010), 40–56. https://doi.org/10.4018/jdst.2010040103
[28] Ivo F. Sbalzarini, Anna Mezzacasa, Ari Helenius, and Petros Koumoutsakos. 2005. Effects of organelle shape on fluorescence recovery after photobleaching. *Biophys. J.* 89, 3 (2005), 1482–1492.
[29] I. F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. 2006. PPM—A highly efficient parallel particle-mesh library for the simulation of continuum systems. *J. Comput. Phys.* 215, 2 (2006), 566–588.
[30] Birte Schrader, Sylvain Reboux, and Ivo F. Sbalzarini. 2010. Discretization correction of general integral PSE operators in particle methods. *J. Comput. Phys.* 229 (2010), 4159–4182.

[31] Stuart Slattery, Samuel Temple Reeve, Christoph Junghans, Damien Lebrun-Grandié, Robert Bird, Guangye Chen, Shane Fogerty, Yuxing Qiu, Stephan Schulz, Aaron Scheinberg, Austin Isner, Kwitae Chong, Stan Moore, Timothy Germann, James Belak, and Susan Mniszewski. 2022. Cabana: A performance portable library for particle-based simulations. *J. Open Source Softw.* 7, 72 (2022), 4115. https://doi.org/10.21105/joss.04115

[32] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. 2018. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *PNAS* 115, 23 (2018), 5849–5854. https://doi.org/10.1073/pnas.1800923115

[33] Jens H. Walther and Ivo F. Sbalzarini. 2009. Large-scale parallel discrete element simulations of granular flow. *Eng. Computat.* 26, 6 (2009), 688–697.

Received 24 January 2023; revised 20 August 2024; accepted 11 September 2024