A unifying mathematical definition enables the theoretical study of the algorithmic class of particle methods.

A dissertation submitted to

TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK

to attain the degree of
Doctor rerum naturalium
(Dr. rer. nat.)

presented by

Johannes Pahlke
(geb. Bamme)
Dipl.-Math.

Dresden
2023

Examination committee:

Prof. Dr. **Jeronimo Castrillon** (chair of the committee)
Technische Universität Dresden, Dresden, Germany

Prof. Dr. **Ivo F. Sbalzarini** (1st reviewer)
Technische Universität Dresden, Dresden, Germany

Prof. Dr. **Udo Hebisch** (2nd reviewer)
Technische Universität Bergakademie Freiberg, Freiberg, Germany

Prof. Dr. **Wolfgang Lehner** (subject expert)
Technische Universität Dresden, Dresden, Germany

Prof. Dr. **Frank J. Furrer** (committee member)
Technische Universität Dresden, Dresden, Germany

Abstract

Mathematical definitions provide a precise, unambiguous way to formulate concepts. They also provide a common language between disciplines. Thus, they are the basis for a well-founded scientific discussion. In addition, mathematical definitions allow for deeper insights into the defined subject based on mathematical theorems that are incontrovertible under the given definition. Besides their value in mathematics, mathematical definitions are indispensable in other sciences like physics, chemistry, and computer science. In computer science, they help to derive the expected behavior of a computer program and provide guidance for the design and testing of software. Therefore, mathematical definitions can be used to design and implement advanced algorithms.

One class of widely used algorithms in computer science is the class of particle-based algorithms, also known as particle methods. Particle methods can solve complex problems in various fields, such as fluid dynamics, plasma physics, or granular flows, using diverse simulation methods, including Discrete Element Methods (DEM), Molecular Dynamics (MD), Reproducing Kernel Particle Methods (RKPM), Particle Strength Exchange (PSE), and Smoothed Particle Hydrodynamics (SPH). Despite the increasing use of particle methods driven by improved computing performance, the relation between these algorithms remains formally unclear. In particular, particle methods lack a unifying mathematical definition and precisely defined terminology. This prevents the determination of whether an algorithm belongs to the class and what distinguishes the class.

Here we present a rigorous mathematical definition for determining particle methods and demonstrate its importance by applying it to several canonical algorithms and those not previously recognized as particle methods. Furthermore, we base proofs of theorems about parallelizability and computational power on it and use it to develop scientific computing software.

Our definition unified, for the first time, the so far loosely connected notion of particle methods. Thus, it marks the necessary starting point for a broad range of joint formal investigations and applications across fields.

Acknowledgments

First of all, I want to thank my supervisor Ivo F. Sbalzarini. Ivo, thank you for supporting me whenever I asked for it! You gave me the freedom and the motivation to do this project which started as a detour from your question, "What can we do with interact and evolve?"

I also want to thank Wolfgang Lehner for giving me the opportunity to start my time as a Ph.D. student at the DFG Graduiertenkolleg RoSI, where I found great guidance and opportunities to develop my skill sets. From the RoSI team, I want to especially thank Lars Schütze, Thomas Kühn, Max Leuthäuser, and Christopher Werner for excellent discussions in the early phase of my project.

I thank all members of the MOSAIC group. It was a great pleasure to work with all of you! I am very grateful for all those discussions, feedback, and proofreading you did for me and for the fantastic, encouraging atmosphere. I especially thank Justina Stark, Nandu Gopan, Lennart Schulze, Suryanarayana Maddu, Anastasia Solomatina, Karl Hoffmann, Tina Šubic, Ulrik Günther and Pietro Incardona for supporting me whenever I needed it.

I thank Udo Hebisch for the fruitful discussion on the theory side of this project and for kindly agreeing to review my thesis.

I thank Georges-Henri Cottet for hosting me at the Université Grenoble Alpes for two months, where I had the chance to work on the practical aspects of my project and discuss it with Jean-Baptiste Keck, Christophe Picard, Aude Maignan, Clément Pernet, and Sofie Thery for which I am very grateful.

I also want to express my great appreciation for the organizational support from Ulrike Schöbel and Susann Gierth.

Last, I want to thank my family for their unconditional love and steady support.

This work has been 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).

Contents

1 Introduction **1**
- 1.1 The Role of Mathematical Definitions . . . 1
- 1.2 Particle Methods . . . 2
- 1.3 Scope and Contributions of this Thesis . . . 4

2 Terminology and Notation **5**

3 A Formal Definition of Particle Methods **9**
- 3.1 Introduction . . . 9
- 3.2 Definition of Particle Methods . . . 9
  - 3.2.1 Particle Method Algorithm . . . 10
  - 3.2.2 Particle Method Instance . . . 10
  - 3.2.3 Particle State Transition Function . . . 11
- 3.3 Explanation of the Definition of Particle Methods . . . 12
  - 3.3.1 Illustrative Example . . . 12
  - 3.3.2 Explanation of the Particle Method Algorithm . . . 13
  - 3.3.3 Explanation of the Particle Method Instance . . . 17
  - 3.3.4 Explanation of the State Transition Function . . . 18
- 3.4 Conclusion . . . 25

4 Algorithms as Particle Methods **27**
- 4.1 Introduction . . . 27
- 4.2 Perfectly Elastic Collision in Arbitrary Dimensions . . . 28
- 4.3 Particle Strength Exchange . . . 30
- 4.4 Smoothed Particle Hydrodynamics . . . 33
- 4.5 Lennard-Jones Molecular Dynamics . . . 39
- 4.6 Triangulation refinement . . . 42
- 4.7 Conway's Game of Life . . . 46
- 4.8 Gaussian Elimination . . . 48
- 4.9 Conclusion . . . 52

5 Parallelizability of Particle Methods **53**
- 5.1 Introduction . . . 53
- 5.2 Particle Methods on Shared Memory Systems . . . 54
  - 5.2.1 Parallelization Scheme . . . 54
  - 5.2.2 Lemmata . . . 56

5.2.3 Parallelizability . . . 59
5.2.4 Time Complexity . . . 60
5.2.5 Application . . . 63
5.3 Particle Methods on Distributed Memory Systems . . . 64
5.3.1 Parallelization Scheme . . . 64
5.3.2 Lemmata . . . 73
5.3.3 Parallelizability . . . 87
5.3.4 Bounds on Time Complexity and Parallel Scalability . . . 91
5.4 Conclusion . . . 97

6 Turing Powerfulness and Halting Decidability 100
6.1 Introduction . . . 100
6.2 Turing Machine . . . 101
6.3 Turing Powerfulness of Particle Methods Under a First Set of Constraints . 102
6.4 Turing Powerfulness of Particle Methods Under a Second Set of Constraints 110
6.5 Halting Decidability of Particle Methods . . . 129
6.6 Conclusion . . . 130

7 Particle Methods as a Basis for Scientific Software Engineering 131
7.1 Introduction . . . 131
7.2 Design of the Prototype . . . 132
7.3 Applications, Comparisons, Convergence Study, and Run-time Evaluations . 135
7.4 Conclusion . . . 138

8 Results, Discussion, Outlook, and Conclusion 140
8.1 Problem . . . 140
8.2 Results . . . 140
8.3 Discussion . . . 141
8.4 Outlook . . . 146
8.5 Conclusion . . . 147

List of Figures

3.1 Nassi-Shneiderman diagram of the state transition function S with annotated sub-functions. 12

4.1 Perfectly elastic collision of three spheres in two dimensions. The arrows indicate the direction and magnitude of the velocities. Left at $t = 0.1$. Right at $t = 1.6$. (Visualized using ParaView [85].) 29
4.2 Particle Strength Exchange simulation of diffusion in three-dimensions at $t = 0.5$. 32
4.3 Dam Break simulation using smoothed particle hydrodynamics at the time $t = \frac{2}{3}$. The wall facing the viewer is clipped for better visualization (visualized using ParaView [85]). 38
4.4 Visualization of a molecular dynamics simulation of 10 Lennard-Jones atoms in one dimension at times $t = 0$ (top) and $t = t_{end} = 10$ (bottom) (visualized using ParaView [85]). 42
4.5 Deviation of the total energy of the molecular dynamics simulation from the total initial energy at $t = 0$. 42
4.6 Visualization of the *refinement* of a single triangle $\mathbf{p}^1$ to four triangles by inserting three new vertices. 45
4.7 Simulation of the Gosper glider gun (visualized using ParaView [85]). . . . 48

5.1 Nassi-Shneiderman diagram of the parallelized state transition function for shared memory systems. The dashed lines enclose the parallel part. 55
5.2 Theoretical speed-up bounds for the shared-memory parallelization. 62
5.3 Illustrations of a cell list in one dimension (left) and two dimensions (right). Particle positions in one dimension are shown as solid vertical lines and in two dimensions as dots. Dotted lines show cell boundaries. Each process is assigned one cell (red) in the finest possible distribution. It is the center cell in the particle storage of that process with copies of the particles from the neighboring processes/cells (green) to either side, see (5.99). 67
5.4 Nassi-Shneiderman diagram of the distributed-memory parallelization of the outer loop of a particle method with pull interactions and without global operations. The dashed double lines mark parallel sections of the algorithm. 72
5.5 Theoretical speed-up bounds. The constants are chosen to be $d = 2$, $C_u = C_\alpha = C_\beta = C_\gamma = 1$, $\tau_i = \tau_e = 3$, and $\tau_f = \tau_{\overset{\circ}{e}} = 1$. For Amdahl's and Gustafson's laws, $N_{\text{cell}} = 900$. 97

7.1 UML Class diagram of the base structure of the *prototype* and its application within the three-dimensional diffusion algorithm based on PSE. 132
7.2 Experiments with our prototype software (visualized using ParaView [85]) . 136
7.3 Validation plots. 137

List of Theorems

1 Theorem (Parallelizability of particle methods on shared-memory computers) 59
2 Theorem (Parallelizability of particle methods on distributed-memory computers) 87
3 Theorem (Turing Powerfulness of particle methods under a first set of constraints) 102
4 Theorem (Turing powerfulness of particle methods under a second set of constraints) 110
5 Theorem (Halting decidability of particle methods under certain constraints) 129

Chapter 1

Introduction

1.1 The Role of Mathematical Definitions

Mathematical definitions provide a precise, unambiguous way to formulate concepts. These definitions can then be employed in further investigations and calculations. For example, the symbol of the empty set $\varnothing$ is defined as the set containing no elements $\varnothing := \{\}$, which is used to define the natural numbers and the von Neumann ordinals [66].

Definitions also provide a common language, thus the basis for a well-founded scientific discussion. By agreeing on a definition, one can reason and argue about a concept and distinguish whether a statement about it is true. For instance, if we define $B := \{\varnothing, \{\varnothing\}\}$ the statement $\varnothing \in B$ is true, whereas if we defined $B := \varnothing$, this would not be the case. Hence, it becomes necessary to use only equivalent definitions of one concept in a specific discussion to prevent misunderstandings. By providing a common language, mathematical definitions can help bridge the gap between different scientific disciplines. For example, in graph theory, where the edges can be defined as a square matrix (the adjacency matrix) and vice versa, providing each field the tools of the other.

Mathematical definitions also allow for deeper insights into the defined subject based on mathematical theorems that are incontrovertible under the given definition. For instance, a connected graph has an Euler cycle if and only if every vertex has an even degree [39]. If we assume the definitions of graph theory for all these terms, we know, thanks to the theorem, that this statement is true for all graphs that fulfill this restriction.

Besides their value in mathematics, mathematical definitions are indispensable in other sciences like physic, chemistry, or computer science. Through the definition of theoretical computer models, i.e., automata, definitions help to derive the expected behavior of a computer program, like the convergence rate of a numerical solver or the time and space complexity of an algorithm. Hence, they provide guidance for the design and testing of software.

Together, mathematical definitions form the basis for informed scientific discussions in computer science and can be used to design and implement advanced algorithms.

1.2 Particle Methods

Particle methods are a classic and widely used class of algorithms in computational science, with various applications ranging from computational plasma physics [40, 86, 9] to computational fluid dynamics [20, 62]. Historically, some of the first computer simulations in these domains used particle methods [87, 17], and the field is still under active development [22, 10]. A key advantage of particle methods is their versatility, as they can be used to simulate both discrete and continuous models stochastically or deterministically.

In simulations of discrete models, particles naturally represent the discrete entities of the examined model, such as atoms or molecules in molecular dynamics (MD) simulations [2], cars in simulations of road traffic [57], or grains of sand in discrete-element simulations of granular flows [89]. When simulating continuous models or numerically solving differential equations, particles represent mathematical collocation or Lagrangian tracer points of the discretization of the continuous fields [72, 23]. The evaluation of differential operators on these fields can directly be approximated on the particles using numerical methods such as Smoothed Particle Hydrodynamics (SPH) [34, 62, 58], Reproducing Kernel Particle Methods (RKPM) [56], Particle Strength Exchange (PSE) [25, 30], or Discretization-Corrected PSE (DC-PSE) [79, 8]. Also, simulations of hybrid discrete-continuous models are possible, as often done in plasma physics, where discrete point charges are coupled with continuous electric and magnetic fields [40, 86, 9]. Beyond the field of simulations, structurally similar algorithms have been developed, e.g., particle-based image processing methods [11, 1], point-based computer graphics [36], and computational optimization algorithms using point samples [38, 63].

In addition to their versatility, particle methods can efficiently be parallelized on shared- and distributed-memory computers [48, 43, 77, 44, 75]. Furthermore, they simplify simulations in complex [78] and time-varying [5] geometries, as no computational mesh needs to be generated and maintained. Therefore, their versatility, parallelizability, and flexibility render particle methods popular in various fields.

Despite the structural similarities of all these algorithms and methods, there is no consensus on what constitutes particle methods and if the methods discussed before are all particle methods. Some see particle methods exclusively as a method for simulating continuum mechanics based on the Lagrangian description and meshless discretization [51, 80] like fluid dynamics. In contrast, others exclusively employ particles for discrete objects of more solid materials [92, 83], others for both [43, 15, 68, 89], and others do not call their methods "particle methods" [38]. Looking deeper, we find similar techniques for the different views, e.g., representing arbitrarily shaped boundaries by placing particles with fixed positions on the boundary. In the SPH community, this is referred to as *dynamic boundary conditions* [24], and for molecular dynamics, we find this as *frozen particles method* [92]. In summary, the term "particle methods" is widely used but ambiguous.

Not only is the terminology "particle methods" widely used and ambiguous, we also find that these two properties, widespread use, and ambiguity, are also present in defining the constituents of particle methods. We identify five main constituents, which resemble five concepts but are denoted with various terms. The first constituent is the basic set of objects which get manipulated throughout the algorithms. In many cases, abstract terms are used to denote them like "particles" [40, 1, 20, 87, 89, 72, 36, 46, 33, 93, 13, 69, 94, 29, 19, 7, 60, 55, 67], "agents" [7, 45, 60, 12, 55], "points" [40, 36, 63, 13, 94], "bodies"

[87, 19], or "(discrete) elements" [89, 29, 19, 67] in other cases, they are named after the objects they represent, e.g., "rods" [40], "molecules" [87], or "pedestrians" [57]. Despite their different names, they represent the same concept of an object with properties and, in most cases, a position in some space but sometimes without a position [45]. The second constituent is these objects' ability to change each other's properties. This behavior is abstractly known as "interaction" [40, 1, 87, 89, 72, 36, 57, 46, 94, 93, 13, 19, 7, 45, 12, 55, 67], but also denoted as the action it describes, e.g. "collision" [40, 89, 46], "interpolation" [33, 69], or "influence" [60]. The third constituent is the determination when objects are "interacting". This is not only to find interaction partners but often also reduces computation. This concept is usually known as "neighborhood" [1, 72, 36, 46, 94, 93, 13, 19, 12, 67], or "surrounding" [40, 57, 13, 69, 7, 12], but also as "sampling radius" [63], "within a distance" [87], "contact" [29, 19], or described by the method they use to determine it, like "Verlet-list" [89]. The fourth constituent is the change of a particle independent of other particles. It is rarely abstracted or treated as a unique concept. We find it as "evolution" [72, 33, 94, 12, 19] but more often as the behavior it describes, like "movement" or "motion" [46, 57, 89, 67, 94, 69, 29]. The fifth and last constituent we identified is the change of the whole system in one step. This is also called "evolution" [1, 72, 36, 63, 69, 7, 45, 60, 55, 67], but more often "(time) step" [40, 20, 87, 89, 36, 57, 46, 33, 94, 93, 13, 69, 29, 7, 12, 55, 67], "update" [63, 57] or "iteration" [1, 45]

In literature, it is seen that the term "evolution" is, in fact, employed for three different sets of transitions, first for the transition of one particle [19], second for the transition of the system in one time step [67], and third for the transition of the system for the entire simulation [89]. This highlights the ambiguity of the terms related to particle methods. There are multiple terms for similar concepts and the same terms for multiple concepts.

Overall, particle methods are a widely used, parallelizable collection of loosely connected methods with a rich theory due to decades of advances but lacking a unifying mathematical definition with precisely defined terminology.

1.3 Scope and Contributions of this Thesis

In this thesis, we aim to demonstrate that a mathematical definition of the algorithmic class of particle methods unifies the so far loosely connected notion and enables joint formal investigations across methods. Therefore we follow the following five steps.

First, we give a mathematical definition of particle methods based on the commonly used terms with their intuitive understanding and their usual way of implementation.

Second, we employ this definition to express a wide variety of algorithms as particle methods, ranging from canonical examples like SPH and MD to non-canonical examples like Triangulation refinement and Gaussian elimination.

Third, we analyze the parallelizability of our definition of particle methods for shared- and distributed-memory parallel computers. Here, we provide formal parallelization schemes, sets of restrictions under which they are applicable, and requisite proofs that they return the same results as our definition of particle methods.

Fourth, we investigate the limits of the computational power of our definition of particle methods. In this regard, we prove for different sets of restrictions of our definition that it is still Turing powerful, and we prove for a more restrictive set that the definition is not anymore Turing powerful but instead the halting problem is decidable for it.

Fifth and last, we demonstrate the practical applicability of the definition as an interface for scientific computing software. For this purpose, we designed, implemented, and tested a running prototype and showcased its use by implementing and executing examples from SPH, PSE, and DEM.

Parts of this thesis have been made publicly available as

Johannes Bamme, and Ivo F. Sbalzarini. **"A Mathematical Definition of Particle Methods."** arXiv preprint arXiv:2105.05637 (2021).

Chapter 2

Terminology and Notation

We introduce the notation and terminology used and define the underlying mathematical concepts.

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

$$A^0 := \{()\},\ A^1 := A,\ A^{n+1} := A^n \times A \quad \text{for } n \in \mathbb{N}_{>0} \tag{2.1}$$

$$A^* := \bigcup_{j=0}^{\infty} A^j. \tag{2.2}$$

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

$$\mathbf{p} \in P^*. \tag{2.3}$$

Notation 2. We use **regular symbols with subscript indices** for the elements of these tuples, e.g.

$$\mathbf{p} = (p_1, ..., p_n). \tag{2.4}$$

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

$$p = (a, b, c) \in A \times B \times C. \tag{2.5}$$

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

$$p_j = (a_j, b_j, c_j). \tag{2.6}$$

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

$$\underline{v} \in A^n. \tag{2.7}$$

Definition 2. Be $a \in \mathbb{R}$, $a = z + r$ 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

$$\lfloor a \rfloor := z. \tag{2.8}$$

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

$$\lfloor \underline{v} \rfloor := \begin{pmatrix} \lfloor v_1 \rfloor \\ \vdots \\ \lfloor v_d \rfloor \end{pmatrix}. \tag{2.9}$$

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

$$|\mathbf{p}| := n. \tag{2.10}$$

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

$$|\underline{v}| = \left| \begin{matrix} v_1 \\ \vdots \\ v_d \end{matrix} \right| := \sqrt{v_1^2 + \ldots + v_d^2}. \tag{2.11}$$

Definition 6. The **Maximum-length of a vector** $\underline{v} \in \mathbb{R}^d$ is defined by the ℓ^∞-norm

$$|\underline{v}|_\infty = \left| \begin{matrix} v_1 \\ \vdots \\ v_d \end{matrix} \right|_\infty := \max\left(v_1, \ldots, v_d\right). \tag{2.12}$$

Definition 7. The **composition operator** $*_h$ of a binary function $h : A \times B \to A$ is recursively defined as:

$$*_h : A \times B^* \to A \tag{2.13}$$

$$a *_h () := a \tag{2.14}$$

$$a *_h (b_1, b_2, ..., b_n) := h(a, b_1) *_h (b_2, ..., b_n). \tag{2.15}$$

Example 1. For $-$ the ordinary arithmetic subtraction, the operator $*_-$ is:

$$9 *_- () = 9,$$
$$13 *_- (3, 4, 1) = (13 - 3) *_- (4, 1) = ((13 - 3) - 4) - 1 = 5.$$

Definition 8. The **concatenation** $\circ : A^* \times A^* \to A^*$ of tuples $(a_1, ..., a_n), (b_1, .., b_m) \in A^*$ is defined as:

$$(a_1, ..., a_n) \circ (b_1, .., b_m) := (a_1, ..., a_n, b_1, .., b_m). \tag{2.16}$$

Definition 9. We define the **construction of a subtuple** $\mathbf{b} \in A^*$ of $\mathbf{a} \in A^*$. Be $f : A^* \times \mathbb{N} \to \{\top, \bot\}$ ($\top = \mathit{true}$, $\bot = \mathit{false}$) the condition for an element a_j of the tuple $\mathbf{a}$ to be in $\mathbf{b}$. $\mathbf{b} = (a_j \in \mathbf{a} : f(\mathbf{a}, j))$ defines a subtuple of $\mathbf{a}$ as follows:

$$\begin{aligned} & \mathbf{b} = (a_j \in \mathbf{a} : f(\mathbf{a}, j)) := (a_{j_1}, ..., a_{j_n}) \\ \leftrightarrow \quad & \mathbf{a} = (a_1, ..., a_{j_1}, ..., a_{j_2}, ..., a_{j_n}, ..., a_m) \\ & \wedge \quad \forall k \in \{1, .., n\} : \ f(\mathbf{a}, j_k) = \top. \end{aligned} \tag{2.17}$$

Example 2. Be $\mathbf{a} = (4, 1, 1, 5, 66, 3, 4, 30)$ and $f(\mathbf{a}, j) := (a_j < 5)$ then

$$\begin{aligned}\mathbf{b} :=& (a_j \in \mathbf{a} : f(\mathbf{a}, j)) \\ =& (a_j \in (4, 1, 1, 5, 66, 3, 4, 30) : a_j < 5) \\ =& (4, 1, 1, 3, 4).\end{aligned}$$

Definition 10. A **permutation** σ is a bijective function mapping the finite set A ($|A| < \infty$) to itself

$$\sigma : A \to A. \tag{2.18}$$

Definition 11. Be $\mathbf{a} = (a_1, \ldots, a_n) \in A^n$ a tuple and $\sigma : \{a_1, \ldots, a_n\} \to \{a_1, \ldots, a_n\}$ a permutation. Then, the **permutation of a tuple** is defined as:

$$\sigma(\mathbf{a}) := (\sigma(a_1), \ldots, \sigma(a_n)). \tag{2.19}$$

Definition 12. Be $\alpha = (a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$ a tuple. Then, an **element** $\langle \alpha \rangle_j$ of a tuple is defined as

$$\langle \alpha \rangle_j := a_j, \tag{2.20}$$

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

$$\langle \alpha \rangle_{(j_1, \ldots, j_m)} := (a_{j_1}, \ldots, a_{j_m}). \tag{2.21}$$

Definition 13. A **subresult** ${}_k f$ of a function

$$f : A_1 \times \ldots \times A_n \to B_1 \times \ldots \times B_m \tag{2.22}$$

is defined as

$${}_k f(a_1, \ldots, a_n) := \langle f(a_1, \ldots, a_n) \rangle_k \quad \text{with} \quad k \in \{1, \ldots, m\}. \tag{2.23}$$

Notation 6. We use a **big number with over- and underline** for a vector with the same number for all entries, e.g.

$$\underline{\overline{\mathbf{1}}} := (1, \ldots, 1)^{\mathbf{T}} \in \mathbb{N}^d. \tag{2.24}$$

Definition 14. Be the dimension of the domain d, the vectorial index space

$$\mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \qquad \text{with} \quad \underline{\overline{\mathbf{I}}} = (I_1, \ldots, I_d)^{\mathbf{T}} \in N^d_{>0}, \tag{2.25}$$

and the corresponding scalar index space

$$\left\{1, \ldots, \prod_{t=1}^{d} I_t\right\} \tag{2.26}$$

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

$${}^{\underline{\overline{\mathbf{I}}}}\iota : \left\{1, \ldots, \prod_{t=1}^{d} I_t\right\} \to \mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \tag{2.27}$$

and defined as

$$
{}_{\underline{\mathbf{I}}}\iota(j) := \begin{pmatrix} (j-1) - \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1 + 1 \\ \left\lfloor \frac{j-1}{I_1} \right\rfloor - \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_2 + 1 \\ \vdots \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor I_l + 1 \\ \vdots \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{d-2} I_t} \right\rfloor - \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor I_{d-1} + 1 \\ \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor + 1 \end{pmatrix}. \tag{2.28}
$$

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

$$
{}_{\underline{\mathbf{I}}}\iota^{-1} : \mathbb{N}^d \cap \left[\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}}\right] \to \left\{1, \ldots, \prod_{t=1}^{d} I_t\right\}, \tag{2.29}
$$

and defined as

$$
{}_{\underline{\mathbf{I}}}\iota^{-1}(\underline{j}) := 1 + (j_1 - 1) + (j_2 - 1) I_1 + (j_3 - 1) I_1 I_2 + \ldots + (j_l - 1) \prod_{t=1}^{l-1} I_t + \ldots + (j_d - 1) \prod_{t=1}^{d-1} I_t. \tag{2.30}
$$

Chapter 3

A Formal Definition of Particle Methods

3.1 Introduction

As we discussed in detail in chapter 1, formal mathematical definitions provide a rigorous way of formulating concepts and form, thus the basis for a well-founded scientific discussion in computational science (see sec. 1.1). One of the classic and widely used classes of algorithms in computational science is particle methods, with various applications ranging from computational plasma physics to computational fluid dynamics (see sec. 1.2). The algorithms collected under the term "particle methods" share structural and terminological similarities but a formal description of those similarities is missing. The current understanding of particle methods mainly relies on qualitative and loose notions, such as particles and their interaction and evolution (see sec. 1.2), which have not yet been rigorously formalized and rationalized as standalone concepts or together to depict particle methods as a whole mathematically.

Here, we fill this gap by presenting a formal definition that unifies particle methods. It opens up new ways of investigating and developing particle methods and algorithms based on these principles in practical applications of scientific computing.

3.2 Definition of Particle Methods

We define particle methods based on the identified concepts and their terms from the literature (see sec. 1.2). In summary, we indemnify five basic concepts, first the manipulated object ("particle", "point", etc.), second, the objects' ability to change each other's properties ("interaction", "collision"), third, the determination when objects change each other's properties ("neighborhood", "surrounding"), fourth the change of a particle independent of other particles ("evolution", "motion"), and fifth the change of the whole system in one "iteration" ("(time) step").

We also structure our definition into three parts with the goal of separation of concerns: First, the definition of the general particle method algorithm structure, including structural components, namely data structures and functions. Here we mainly use the concepts and terms of the literature. Second, the definition of a particle method instance.

A particle method instance describes a specific problem or setting, which can then be solved or simulated using the particle method algorithm. Third, the definition of the particle state transition function. The state transition function describes how a particle method instance proceeds from the instance to the final state using the data structures and functions from the particle method algorithm. In summary, we present a definition of particle methods in the most general, sequential form.

3.2.1 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. This approach follows a similar logic as some definitions of Turing machines [52]. Both concepts describe state-transition systems working on discrete objects.

Definition 15. A **particle method algorithm** is a 7-tuple $(P, G, u, f, i, e, \mathring{e})$, consisting of the two data structures

$$P := A_1 \times A_2 \times ... \times A_n \qquad \text{the particle space,} \tag{3.1}$$

$$G := B_1 \times B_2 \times ... \times B_m \qquad \text{the global variable space,} \tag{3.2}$$

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

$$u : [G \times P^*] \times \mathbb{N} \to \mathbb{N}^* \qquad \text{the neighborhood function,} \tag{3.3}$$

$$f : G \to \{\top, \bot\} \qquad \text{the stopping condition,} \tag{3.4}$$

$$i : G \times P \times P \to P \times P \qquad \text{the interact function,} \tag{3.5}$$

$$e : G \times P \to G \times P^* \qquad \text{the evolve function,} \tag{3.6}$$

$$\mathring{e} : G \to G \qquad \text{the evolve function of the global variable.} \tag{3.7}$$

These are the only objects to be defined by the user to specify a particle method algorithm.

3.2.2 Particle Method Instance

Using the above definitions of the particle method algorithm and its data structures and functions, we define an *instance* of a particle method as a specific realization.

Definition 16. An initial state defines a **particle method instance** for a given particle method algorithm $(P, G, u, f, i, e, \mathring{e})$:

$$[g^1, \mathbf{p}^1] \in [G \times P^*]. \tag{3.8}$$

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

3.2.3 Particle State Transition Function

In a specific particle method, the elements of the tuple $(P, G, u, f, i, e, \mathring{e})$ (3.1) – (3.7) need to be specified. Given a specific starting point defined by an instance, the algorithm proceeds in iterations. Each iteration corresponds to one state transition step s that advances the current state of the particle method $[g^t, \mathbf{p}^t]$ to the next state $[g^{t+1}, \mathbf{p}^{t+1}]$, starting at the instance $[g^1, \mathbf{p}^1]$. The state transition uses the functions $u, i, e, \mathring{e}$ to determine the next state. The state transition function S generates a series of these state transition steps until the stopping function f is *true*. The so-calculated final state is the result of the state transition function. The state transition function is the same for every particle method and does not need to be defined by the user.

Definition 17. We define the *state transition function*

$$S : [G \times P^*] \to [G \times P^*] \tag{3.9}$$

with three interact sub-functions $(\iota^{\mathrm{I}}, \iota^{\mathrm{I}\times\mathrm{U}}, \iota^{\mathrm{N}\times\mathrm{U}})$, two evolve sub-functions $(\epsilon^{\mathrm{I}}, \epsilon^{\mathrm{N}})$ and the state transition step (s). These functions build upon each other. The interact sub-functions manipulate only a particle tuple $\mathbf{p}$ and ultimately compute all interactions of each particle with all its neighbors.

The first interact sub-function ι^{I} calculates one interaction and results, therefore, in the change of two particles in the particle tuple $\mathbf{p} = (p_1, ..., p_{|\mathbf{p}|})$,

$$\iota^{\mathrm{I}}_{(g,j)}(\mathbf{p}, k) := \big(p_1, .., p_{j-1}, \overline{p}_j, p_{j+1}, ..., p_{k-1}, \overline{p}_k, p_{k+1}, ..., p_{|\mathbf{p}|}\big) \quad \text{for } \big(\overline{p}_j, \overline{p}_k\big) := i\,(g, p_j, p_k)\,. \tag{3.10}$$

The second interact sub-function $\iota^{\mathrm{I}\times\mathrm{U}}$ builds on ι^{I} and calculates for one particle the interaction with all its neighbors given by the neighborhood function u. The result is a potential change of all involved particles in the particle tuple $\mathbf{p}$,

$$\iota^{\mathrm{I}\times\mathrm{U}}_g(\mathbf{p}, j) := \mathbf{p} \ *_{\iota^{\mathrm{I}}_{(g,j)}} \ u([g, \mathbf{p}], j) \tag{3.11}$$

The third interact sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$ completes the interactions. It uses $\iota^{\mathrm{I}\times\mathrm{U}}$ to calculate the interactions for all particles with their respective neighbors, leading to a change of $\mathbf{p}$ in potentially every particle,

$$\iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}]) := \mathbf{p} \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_g} (1, .., |\mathbf{p}|) \tag{3.12}$$

The first evolution sub-function ϵ^{I} calculates the evolution of one particle. The result is stored in the global variable and an intermediate particle tuple $\mathbf{q}$,

$$\epsilon^{\mathrm{I}}_{\mathbf{p}}\big([g, \mathbf{q}], j\big) := \big[\overline{g}, \mathbf{q} \circ \overline{\mathbf{q}}\big] \qquad \text{for } (\overline{g}, \overline{\mathbf{q}}) := e(g, p_j). \tag{3.13}$$

The second evolution sub-function ϵ^{N} calculates for all particles the evolution. The result is returned in the global variable and a new particle tuple,

$$\epsilon^{\mathrm{N}}\big([g, \mathbf{p}]\big) := \big[g, ()\big] \ *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} \ (1, .., |\mathbf{p}|) \tag{3.14}$$

The state transition step s brings all sub-functions together and advances the particle simulation by one iteration,

$$s\left([g, \mathbf{p}]\right) := \left[\mathring{e}(\overline{g}),\ \overline{\mathbf{p}}\right] \quad \text{for} \quad [\overline{g}, \overline{\mathbf{p}}] := \epsilon^{\mathrm{N}}\big([g,\ \iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}])]\big). \tag{3.15}$$

Finally, the state transition function S advances the particle method instance to the final state,

$$\begin{aligned} &S([g^1, \mathbf{p}^1]) = [g^T, \mathbf{p}^T] \quad \longleftrightarrow \\ &\quad f(g^T) = \top \quad \wedge \quad \forall t \in \{2, ..., T\}: \ [g^t, \mathbf{p}^t] = s\left([g^{t-1}, \mathbf{p}^{t-1}]\right) \ \wedge \ f(g^{t-1}) = \bot. \end{aligned} \tag{3.16}$$

We illustrate the state transition function and its sub-functions with the Nassi-Shneiderman diagram:

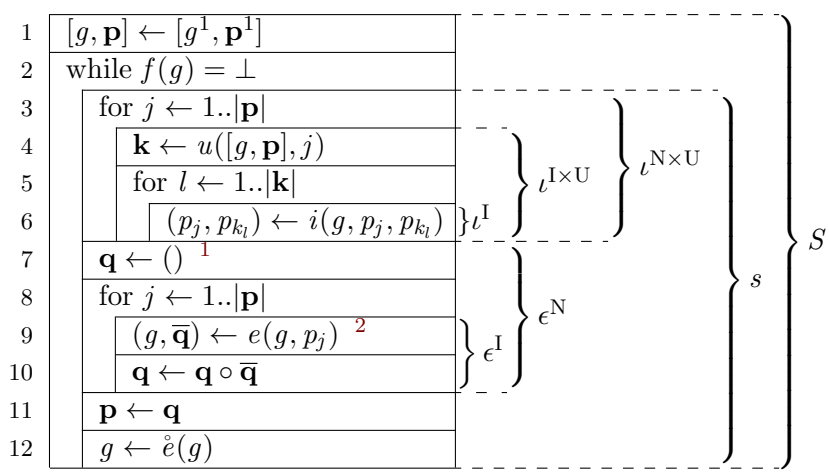


Figure 3.1: Nassi-Shneiderman diagram of the state transition function S with annotated sub-functions.

This is the most generic form of the state transition function without further constraints and for sequential computing. Further constraints can be imposed, leading to more specific state transition functions valid for a subset of particle methods.

3.3 Explanation of the Definition of Particle Methods

3.3.1 Illustrative Example

To illustrate our definition of particle methods, we use a single, simple example throughout the explanations. This example is motivated by the Discrete Element Method (DEM) [81] but is greatly simplified and reduced to its key constituents. This example serves didactic purposes and has no ambition to resemble any real physics but only to illustrate our abstract concepts in a concrete and straightforward case.

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

The example models perfectly elastic collisions between spheres of uniform diameter d and constant unit mass in a one-dimensional continuous space. The space is without boundaries, such that no boundary conditions are required. The position x of an individual sphere changes over time t, as:

$$x(t) = x_0 + \int_0^t v(t)\,\mathrm{d}t, \tag{3.17}$$

where v is the velocity of the sphere and x_0 its initial position. The explicit Euler time-stepping algorithm discretizes this equation in time using a fixed time step size Δt, yielding:

$$x^{t+\Delta t} = x^t + v(t)\Delta t, \tag{3.18}$$

which is iterated until a given final time t_{end}. A perfectly elastic collision between two spheres 1 and 2 results in them swapping their velocities. The positions before the collision x^n and the positions after the collision x^{n+1} remain the same, leading to the following collision rules:

$$x_1^{n+1} = x_1^n, \quad x_2^{n+1} = x_2^n, \quad v_1^{n+1} = v_2^n, \quad v_2^{n+1} = v_1^n. \tag{3.19}$$

Two spheres 1 and 2 are considered colliding if and only if $|x_2 - x_1| \leq d$.

3.3.2 Explanation of the Particle Method Algorithm

The particle method algorithm definition encapsulates the structural elements the user has to define. It incorporates two data structures and five functions. In this section, we explain the data structures and functions in detail and showcase these with the illustrative example (sec. 3.3.1).

Explanation of the Particle Space P

$$P := A_1 \times ... \times A_n \tag{3.1}$$

The structure of the particle space P follows directly from the observation that in any particle method, the particles are points in some space that carry/store some properties. Hence, in our definition, a particle is a tuple of properties, with its position being one of them. The particle space is the Cartesian product of the property sets $A_1, ..., A_n$ since every of the n properties of a particle can be of a different data type and hence live in a different space. Examples of particle properties can include a color, a velocity, an acceleration, a Boolean flag, a position in some space, a vector, a name, etc.

In the example from Section 3.3.1:

$$p := (x, v) \in P := \mathbb{R} \times \mathbb{R} \qquad (n = 2, \quad A_1, A_2 = \mathbb{R}) \tag{3.20}$$

In the example, particles are spheres of identical size and mass. Therefore, a particle is fully described by its position x and its velocity v. Hence, it is reasonable to choose the particles space as $P := \mathbb{R} \times \mathbb{R}$.

Explanation of the Global Variable Space G

$$G := B_1 \times ... \times B_m \tag{3.2}$$

The structure of the global variable space G is analogous to that of the particle space. A global variable is also a tuple of properties. From the perspective of computational power, the global variable is unnecessary. Still, it simplifies the formulation of many particle method algorithms by encapsulating simulation properties that are not specific to a particle. The global variable improves readability and possibly implementation efficiency. The global variable is accessible throughout the particle method. All functions can depend on it. The global variable space is the Cartesian product of the property sets $B_1, ..., B_m$. Generally, the property sets $B_1, ..., B_m$ of the m components of the global variable can be of arbitrary data type. Examples of global properties include the time step size of a simulation, the total number of particles in the simulation, the total energy of the system, and so on.

In the example from Section 3.3.1:

$$g := (d, t, \Delta t, t_{end}) \in G := \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \qquad (m = 4, \quad B_1, ..., B_4 = \mathbb{R}) \tag{3.21}$$

In the example, the global variable g is the collection of the non-particle-specific diameter d of the spheres and the time parameters of the simulation, the current time t, the simulation time step size Δt, and the stopping time t_{end}. Again, all can be chosen as real numbers. If instead of storing the time, one would store the index of the current time step, then the corresponding set could be $\mathbb{N}$ instead of $\mathbb{R}$.

Explanation of the State Space $[G \times P^*]$

$$[G \times P^*]$$

The state of a particle method collects all information about a particle method at a particular time point. Hence, the state of a particle method consists of a global variable and potentially many particles collected in a particles tuple.

The particle method's state space is the Cartesian product of the global variable space G and the set of all tuples of particles P^* from the particle space P (def. 1). An element $[g^t, \mathbf{p}^t] \in [G \times P^*]$ fully describes the state of a particle method at a time point t. We use the square brackets ($[\cdot]$) to mean that the state is one element even though we write the global variable g and the particle tuple $\mathbf{p}$ separately.

Explanation of the Neighborhood Function u

$$u : [G \times P^*] \times \mathbb{N} \to \mathbb{N}^* \tag{3.3}$$

The neighborhood function u is the only function that generally depends on the particle method's entire state $[g, \mathbf{p}]$. We introduce this function to reduce computation. Without it, each particle would need to interact with every other particle to decide if it contributes to their respective changes. Suppose the number of contributing interactions is low compared to all particles. In that case, the neighborhood function helps reduce the computation by returning the indices of only those interaction partners contributing to the result. For

instance, we can define the neighborhood by a cut-off radius beyond which particles no longer "feel" each other. We can use a Cell list [71], or Verlet list [87] to implement a neighborhood function with linear run-time. However, a neighborhood does not need to be defined geometrically but can be an arbitrary set of particle indices.

The neighborhood function, therefore, operates on particle indices. It takes an index as input, pointing to a particle in the current state, and returns a tuple of indices, indicating the neighboring particles of the input particles. We chose indices instead of particles directly because an index is an element's unique identifier in a tuple. Besides, indices provide stable identifiers of particles throughout the whole interaction phase. They remain the same, while the particle itself may change.

Formally, the neighborhood function maps a state of a particle method and an index of a particle to a tuple of particle indices. It returns the indices of all interaction partners of the particle with the input index.

In the example from Section 3.3.1:

$$u([g, \mathbf{p}], j) := (k \in (1, ..., |\mathbf{p}|) : p_k, p_j \in \mathbf{p} \ \wedge \ 0 < x_k - x_j \leq d) \tag{3.22}$$

In the example, the neighborhood function u returns a subtuple (def. 9) of all collision partners of the particle $p_j \in \mathbf{p}$. In this case, the neighborhood function considers each collision pair only once, i.e., if p_k is in the neighborhood of p_j, then p_j shall not be in the neighborhood of p_k. This is a matter of definition and is typically known as "asymmetric neighborhood" in particle methods. If the neighborhood function u would be: $u([g, \mathbf{p}], j) := (k \in (1, ..., |\mathbf{p}|) : \ p_k, p_j \in \mathbf{p} \ \wedge \ k \neq j \ \wedge \ |x_k - x_j| \leq d)$, then both collision partners would show up as neighbors of the respective other.

Explanation of the Interact Function i

$$i : G \times P \times P \to P \times P \tag{3.5}$$

A particle method proceeds by computing interactions between particles. The basic building block for this is the interact function, which specifies how two particles interact. Higher-order interactions, e.g., three-body interactions, are realized by multiple pairwise interactions. The interact function can change both particles. This provides a performance advantage in cases of symmetric interactions. Therefore, it cannot only read information from the interacting particles but also write information into the interacted particles.

Formally, the interact function maps a triple of a global variable and two particles to two particles. It describes the changes the interaction causes to the properties of both interacting particles. While, in general, i can change both particles, the change can be zero for one or both of them. The interact function is only applied for pairs of particles identified by the tuple of indices provided by the neighborhood function u.

In the example from Section 3.3.1:

$$i(g, p_j, p_k) := ((x_j, v_k), (x_k, v_j)) \tag{3.23}$$

In the example, an interaction amounts to an elastic collision between two spheres, as described by the collision rules (3.19). Hence, the two involved particles p_j and p_k keep

their positions x_j and x_k but swap their velocities v_j and v_k. In this case, the interact function i changes both particles. Therefore the neighborhood function u defines an asymmetric neighborhood, preventing duplicate interactions. Meaning, if p_k is in the neighborhood of p_j, then p_j will not be in the neighborhood of p_k. Hence, they interact just once *as a pair*. In practical applications, this type of symmetric interaction reduces the computational cost of the algorithm and helps conserve quantities such as energy. If the interaction changed only one particle, we would need an additional mechanism to ensure that no kinetic energy gets lost, e.g., by using additional properties to keep a running tally of the net velocity exchanged.

Explanation of the Evolve Function e

$$e : G \times P \to G \times P^* \tag{3.6}$$

The evolve function e changes the properties of a particle due to its properties and the global variable. This includes any change that is independent of interactions with other particles. The evolve function provides a place to update properties that need to stay constant during all interactions or to reset properties that serve as temporary accumulators used during the interactions. It is also the place to implement autonomous dynamics, i.e., dynamics that, for example, only depend on time but not on the properties of the other particles. Examples include simulations of the chemical reaction terms in a spatio-temporal reaction-diffusion simulation or autonomous stochastic processes.

Formally, the evolve function maps a global variable and a particle to a global variable and a tuple of particles. It can therefore change the global variable, for instance, to implement global reduction operations like summing a property over all particles. Since the result is a tuple of particles, e can also create or destroy particles. This is, for example, used in population dynamics simulations or adaptive-resolution methods for continuous models [72]. Hence, e potentially changes the total number of particles $|\mathbf{p}|$.

In the example from Section 3.3.1:

$$e(g, p_j) := \big(g,\ \big((x_j + \Delta t \cdot v_j,\ v_j)\big)\big) \tag{3.24}$$

In the example, the global variable g remains unchanged by the evolve function, even though this is not necessary per the definition. The position x_j of the particle p_j in the example is evolved according to the velocity v_j and the time step size $\Delta t \in g$ using explicit Euler time-stepping (3.18). The velocity v_j of the particle p_j stays the same.

Explanation of the Evolve Function of the Global Variable $\mathring{e}$

$$\mathring{e} : G \to G \tag{3.7}$$

The evolve function of the global variable $\mathring{e}$ changes the global variable based only on the current value of the global variable. We chose to have this function because otherwise, the only place to change the global variable would be the evolve function of the particles, which would make it challenging to prevent multiple evolutions of the global variable. Hence, the evolve function of the global variable provides us with a structure to describe the singleton global behavior of the simulation, such as increasing the simulation time after the completion of a time step.

Formally, the evolve function of the global variable maps a global variable to a global variable. It describes the changes to the global variable due to its values. This allows updating the global variable without requiring a global operation.

In the example from Section 3.3.1:

$$\mathring{e}(g) := (d, t + \Delta t, \Delta t, t_{end}) \tag{3.25}$$

In the example, the evolve function of the global variable $\mathring{e}$ increments the current time t by the time step size Δt. All other global properties remain unchanged.

Explanation of the Stopping Condition f

$$f : G \to \{\top, \bot\} \tag{3.4}$$

The stopping condition f only depends on the global variable. This choice is not limiting because every particle can change the global variable in the evolve function. Hence, each particle can influence the outcome of the stopping condition.

Formally, the stopping condition is a function that maps a global variable to a Boolean value, $\top$ (true) or $\bot$ (false). It determines when the algorithm ends. It ends when $f(g) = \top$.

In the example from Section 3.3.1:

$$f(g) := (t > t_{end}) \tag{3.26}$$

In the example, the algorithm terminates when the current time t exceeds the stopping time t_{end}, then $f(g) = \top$.

3.3.3 Explanation of the Particle Method Instance

The particle method instance $[g^1, \mathbf{p}^1]$ (3.8) specifies the starting point of a particle method. We chose to separate the definition of the particle method instance from the definition of the particle method algorithm to distinguish the specific problem from the general algorithm.

Formally, a particle method instance $[g^1, \mathbf{p}^1]$ is an element of the state space $[G \times P^*]$ of a particle method. Since an element $[g^t, \mathbf{p}^t] \in [G \times P^*]$ fully describes the state of a particle method at a time point t, the particle method instance $[g^1, \mathbf{p}^1]$ fully describes the state of a particle method at the beginning.

In the example from Section 3.3.1:

$$[g^1, \mathbf{p}^1] \in [G \times P^*] \qquad \text{initial state of the particle method} \tag{3.27}$$

$$\begin{aligned} g^1 :=& (d, t, \Delta t, t_{end}) \in G && \text{initial global variable} \\ d :=& \ 0.5 && \text{diameter of the spheres} \\ t :=& \ 0 && \text{initial time} \\ \Delta t :=& \ 0.1 && \text{time step size} \\ t_{end} :=& \ 0.1 && \text{stopping time} \end{aligned} \tag{3.28}$$

$$\begin{aligned} \mathbf{p}^1 :=& \big(p_1, p_2, p_3\big) \in P^* && \text{initial particle tuple} \\ p_1 :=& \ (0, 2) && \text{particle 1 with } x_1 = 0 \text{ and } v_1 = 2 \\ p_2 :=& \ (0.49, -1) && \text{particle 2 with } x_2 = 0.49 \text{ and } v_2 = -1 \\ p_3 :=& \ (2, 1) && \text{particle 3 with } x_3 = 2 \text{ and } v_3 = 1 \end{aligned} \tag{3.29}$$

This example of a particle method instance is the initial state for the didactic example from the section 3.3.1. It is a 1D elastic collision simulation of three spheres with the same diameter and mass. Since mass cancels out if it is the same for all spheres, we do not explicitly store it. This particle method instance defines the initial state $[g^1, \mathbf{p}^1]$. The initial values of the properties of the global variable g^1 are the diameter of the spheres $d = 0.5$, the initial time $t = 0$, the time step size $\Delta t = 0.1$, and the stopping time $t_{end} = 0.1$. The initial particle tuple $\mathbf{p}$ consists of three particles, representing three spheres. These particles are $p_1 = (0, 2)$, $p_2 = (0.49, -1)$, and $p_3 = (2, 1)$. As one can see, p_1 and p_2 are already colliding. Their distance is smaller than d, and their velocities point toward each other.

3.3.4 Explanation of the State Transition Function

In a specific particle method, the elements of the tuple $(P, G, u, f, i, e, \mathring{e})$ (3.1) – (3.7) need to be specified. Given a specific starting point defined by an instance, the algorithm proceeds in iterations. Each iteration corresponds to one state transition step that advances the current state of the particle method $[g^t, \mathbf{p}^t]$ to the next state $[g^{t+1}, \mathbf{p}^{t+1}]$, starting at the instance $[g^1, \mathbf{p}^1]$. The state transition function S determines the final state from the instance by going through all in-between states using the state transition step s. The state transition function is based on $u, f, i, e, \mathring{e}$. Hence, it is the same for every particle method and does not need to be defined by the user.

We defined the transition function S (3.16) using the particle method state transition step s (3.15) and five sub-functions ι^{I}, $\iota^{\mathrm{I\times U}}$, $\iota^{\mathrm{N\times U}}$, ϵ^{I}, ϵ^{N} (3.10)–(3.14). The Nassi-Shneiderman diagram (fig. 3.1) shows what state transition function S, the state transition step s, and the sub-functions $\iota^{\mathrm{I}}, \iota^{\mathrm{I\times U}}, \iota^{\mathrm{N\times U}}, \epsilon^{\mathrm{I}}, \epsilon^{\mathrm{N}}$ are calculating.

All particles interact with their respective interaction partners (possibly all particles), as given by the neighborhood function u. We disassemble the interactions into a loop over all particles $\iota^{\mathrm{N\times U}}$ (3.12), where each particle interacts with its specific neighbors $\iota^{\mathrm{I\times U}}$ (3.11). Each of these interactions, in turn, is a pairwise interaction between two

particles ι^{I} (3.10) as defined by i (3.5). After all interactions, evolutions happen. We can break it down into a loop over all particles ϵ^{N} (3.14), where each particle evolves ϵ^{I} (3.14) as defined by e (3.6). The state transition step s advances one state to the next state by using the interaction loop $\iota^{\mathrm{N}\times\mathrm{U}}$, the evolution loop ϵ^{N}, and the evolution of the global variable $\mathring{e}$ (3.7). The state transition function S uses the state transition step s and the stopping condition f (3.4) to determine the series of states from the instance to the final state.

These functions are entirely defined (3.10)–(3.16) and can be formally written down by using the formal definition of a particle method algorithm (sec. 3.2.1). Over-bars indicate intermediate results used in the subsequent sub-functions, e.g., the result $\left(\overline{p}_j, \overline{p}_k\right)$ of the interact function $i\left(g, p_j, p_k\right)$ is used in ι^{I}. The composition operator $*_h$ (def. 7) is only defined for functions h with exactly two arguments. Therefore, functions with more than two arguments are written in an indexed form. For example, the function ι^{I} has 3 arguments where one is a state. Hence, it is written as $\iota^{\mathrm{I}}_{(g,j)}(\mathbf{p}, k)$, such that only the two arguments $\mathbf{p}$ and k are arguments of the function, whereas g and j become indices.

In our definition, after each interaction, the current particle tuple is updated such that the result of the interaction can affect the following interactions. The same applies to the evolve function concerning the global variable g. We include this possibility in the definition for generality, even though many practical examples do not require it. In general, the algorithm result depends on the order of the interactions and evolutions, hence on the index ordering of the particles. The result of the algorithm becomes independent of particle ordering if a commutative operation reduces the interaction results and the global variable during the evolution, usually the addition, and accumulated in a separate particle property until the final evolve step ϵ^{N}.

For the following explanations, we use the particle method algorithm and instance as defined in the explanations (sec. 3.3.2, 3.3.3).

Explanation of the First Interaction Sub-function ι^{I}

$$\iota^{\mathrm{I}} : [G \times P^*] \times \mathbb{N} \times \mathbb{N} \to P^* \tag{3.30}$$

$$\iota^{\mathrm{I}}_{(g,j)}(\mathbf{p}, k) := \left(p_1, .., p_{j-1}, \overline{p}_j, p_{j+1}, ..., p_{k-1}, \overline{p}_k, p_{k+1}, ..., p_{|\mathbf{p}|}\right) \quad \text{for } \left(\overline{p}_j, \overline{p}_k\right) := i\left(g, p_j, p_k\right). \tag{3.10}$$

The first interaction sub-function ι^{I} describes how the interaction of two particles contributes to the state change of the particle method. It is the core component of the loop over all particles and all neighbors of each particle. The function ι^{I} evaluates the interaction function i for a given pair of particles and a given current state of the particle method. It then takes the changed particles that i returns and stores this change again in the state. We choose this encapsulation over directly evaluating i because it requires no external storage to accumulate the changes of subsequent interactions. Formally, this function maps a particle method state and two particle indices to a tuple of particles. The returned tuple of particles is the same as $\mathbf{p}$ from the input state except for the interacting particles p_j and p_k, which might be changed. The result of the interact function

$i\,(g, p_j, p_k)$, which is the tuple $\big(\overline{p}_j, \overline{p}_k\big)$, replaces these particles.

In the example from Section 3.3.1:

$$
\begin{aligned}
(\overline{p}_1, \overline{p}_2) &= i(g^1, p_1, p_2) && (3.31)\\
&= i\big(g^1, (0,2), (0.49,-1)\big) && (3.32)\\
&= \big((0,-1), (0.49,2)\big), && (3.33)\\
\iota^{\mathrm{I}}_{(g^1,1)}(\mathbf{p}^1, 2) &= (\overline{p}_1, \overline{p}_2, p_3) && (3.34)\\
&= \big((0,-1), (0.49,2), (2,1)\big). && (3.35)
\end{aligned}
$$

Here, the result of the interaction of the particles p_1 and p_2 is $\overline{p}_1$ and $\overline{p}_2$. Both $\overline{p}_1$ and $\overline{p}_2$ have the same positions as p_1 and p_2 but swapped velocities. The function ι^{I} replaces in the particle tuple $\mathbf{p}$ the interacting particles p_1 and p_2 by their interaction results $\overline{p}_1$ and $\overline{p}_2$.

Explanation of the Second Interaction Sub-function

$$
\begin{aligned}
\iota^{\mathrm{I\times U}} &: [G \times P^*] \times \mathbb{N} \to P^* && (3.36)\\
\iota^{\mathrm{I\times U}}_g(\mathbf{p}, j) &:= \mathbf{p} \;*_{\iota^{\mathrm{I}}_{(g,j)}}\; u([g, \mathbf{p}], j) && (3.11)
\end{aligned}
$$

The second interaction sub-function $\iota^{\mathrm{I\times U}}$ describes how the interaction of one particle with all its neighbor particles contributes to the state change of the particle method. It calculates the inner loop where a particle interacts with all its neighbors. Therefore, it uses the previous function ι^{I} and evaluates it for each neighbor given by u.

Formally, it maps a state of the particle method and a particle index to a tuple of particles. It uses the composition operator $*$ (def. 7) over the function $\iota^{\mathrm{I}}_{(g,j)}$, i.e., $*_{\iota^{\mathrm{I}}_{(g,j)}}$, to compute the loop over all neighbors of one particle. It, therefore, computes the interactions of the particle p_j with all its neighbors in $\mathbf{p}$. The neighborhood function $u([g, \mathbf{p}], j)$ returns the indices of these neighbors. The result of each pairwise interaction is stored in an altered tuple of particles $\mathbf{p}$. This altered tuple of particles is then used for the subsequent pairwise interaction until p_j has interacted with all its neighbors.

In the example from Section 3.3.1:

$$
\begin{aligned}
u([g^1, \mathbf{p}^1], 1) &= (2), && (3.37)\\
\iota^{\mathrm{I\times U}}_{g^1}(\mathbf{p}^1, 1) &= \mathbf{p}^1 \;*_{\iota^{\mathrm{I}}_{(g^1,1)}}\; u([g^1, \mathbf{p}^1], 1) && (3.38)\\
&= \mathbf{p}^1 \;*_{\iota^{\mathrm{I}}_{(g^1,1)}}\; (2) && (3.39)\\
&= \iota^{\mathrm{I}}_{(g^1,1)}(\mathbf{p}^1, 2) \;*_{\iota^{\mathrm{I}}_{(g^1,1)}}\; () && (3.40)\\
&= \iota^{\mathrm{I}}_{(g^1,1)}(\mathbf{p}^1, 2) && (3.41)\\
&= \big((0,-1), (0.49,2), (2,1)\big). && (3.42)
\end{aligned}
$$

The neighboring indices of the first particle in $\mathbf{p}^1$, which is p_1, are calculated by the neighborhood function $u([g^1, \mathbf{p}^1], 1)$. The result of this function is the one-element index

tuple (2) since only particle 2 collides with particle 1. Hence, the composition operator in the function $\iota^{\mathrm{I}\times\mathrm{U}}$ only needs to iterate over one index. Composition with an empty tuple () would not change anything. Hence, the result of the function $\iota^{\mathrm{I}}_{(g^1,1)}(\mathbf{p}^1, 2)$ as described in (3.35) is reduced.

Explanation of the Third Interaction Sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$

$$\iota^{\mathrm{N}\times\mathrm{U}} : [G \times P^*] \to P^* \tag{3.43}$$

$$\iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}]) := \mathbf{p} \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_g} (1, .., |\mathbf{p}|) \tag{3.12}$$

The third interaction sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$ describes how the interactions of all particles with all their respective neighbors contribute to the state change of the particle method. Therefore, it calculates both the inner and the outer loop to calculate for each particle the interaction with its neighbors, recursively using the second interaction sub-function for the inner loop.

Formally, this maps a state of the particle method to a tuple of particles. It uses the composition operator $*$ (def. 7) over the second interact sub-function $\iota^{\mathrm{I}\times\mathrm{U}}$ to iterate over all indices of the particles in the tuple $\mathbf{p}$. The function $\iota^{\mathrm{N}\times\mathrm{U}}$ calculates the interaction between these particles and their neighbors for each of these particles, hence completely computing all necessary interactions between all particles in $\mathbf{p}$.

In the example from Section 3.3.1:

$$\begin{aligned}
\iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1]) &= \mathbf{p}^1 \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (1, 2, 3) && (3.44)\\
&= \iota^{\mathrm{I}\times\mathrm{U}}_{g^1}(\mathbf{p}^1, 1) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (2, 3) && (3.45)\\
&= \underbrace{\big((0, -1), (0.49, 2), (2, 1)\big)}_{=:\hat{\mathbf{p}}^1} *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (2, 3) \qquad = \hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (2, 3) && (3.46)\\
&= \iota^{\mathrm{I}\times\mathrm{U}}_{g^1}(\hat{\mathbf{p}}^1, 2) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (3) && (3.47)\\
&= \Big(\hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}}_{(g^1,2)}} \ u([g^1, \hat{\mathbf{p}}^1], 2)\Big) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (3) && (3.48)\\
&= \Big(\hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}}_{(g^1,2)}} \ ()\Big) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (3) && (3.49)\\
&= \hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} (3) && (3.50)\\
&= \iota^{\mathrm{I}\times\mathrm{U}}_{g^1}(\hat{\mathbf{p}}^1, 3) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} () && (3.51)\\
&= \Big(\hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}}_{(g^1,3)}} \ u([g^1, \hat{\mathbf{p}}^1], 3)\Big) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} () && (3.52)\\
&= \Big(\hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}}_{(g^1,3)}} \ ()\Big) \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} () && (3.53)\\
&= \hat{\mathbf{p}}^1 \ *_{\iota^{\mathrm{I}\times\mathrm{U}}_{g^1}} () && (3.54)\\
&= \hat{\mathbf{p}}^1 && (3.55)\\
&= \big((0, -1), (0.49, 2), (2, 1)\big). && (3.56)
\end{aligned}$$

The function $\iota^{\mathrm{N\times U}}$ calculates the pairwise interactions of all particles with all their neighbors starting from the particle p_1 and its neighbors. This is already calculated in the previous step (3.37) – (3.42) with the result named $\hat{\mathbf{p}}^1$ for better readability. The next particle is p_2. It has to interact with all of its neighbors, calculated through $\iota^{\mathrm{I\times U}}_{g^1}(\hat{\mathbf{p}}^1, 2)$. In the example, particle p_2 has no neighbors in $\hat{\mathbf{p}}^1$. Hence, there is no interaction or change in $\hat{\mathbf{p}}^1$. The same is valid for particle p_3. It also has no neighbors. Hence, there is again no change in $\hat{\mathbf{p}}^1$. This makes $\hat{\mathbf{p}}^1$ the result particle tuple of the pairwise interactions of all particles with all their neighbors.

Explanation of the First Evolution Sub-function ϵ^{I}

$$\epsilon^{\mathrm{I}} : [G \times P^*] \times P^* \times \mathbb{N} \to [G \times P^*] \tag{3.57}$$

$$\epsilon^{\mathrm{I}}_{\mathbf{p}}\big([g, \mathbf{q}], j\big) := \big[\overline{g}, \mathbf{q} \circ \overline{\mathbf{q}}\big] \qquad \text{for } (\overline{g}, \overline{\mathbf{q}}) := e(g, p_j). \tag{3.13}$$

The first evolution sub-function describes how the evolution of one particle contributes to the state change of the particle method. Therefore it evolves a particle and stores the result in a new particle tuple.

Formally, it maps a state of the particle method, a particle tuple, and an index to a state of the particle method. It handles the result of the evolution of one particle and concatenates (def. 8) the result, a particle tuple $\overline{\mathbf{q}}$, to a particle tuple $\mathbf{q}$. Also, the function ϵ^{I} handles the change this particle causes to the global variable. The calculations are done using a given particle method's evolve function e.

In the example from Section 3.3.1:

$$\hat{\mathbf{p}}^1 = \big(\hat{p}_1, \hat{p}_2, \hat{p}_3\big) = \big((0, -1), (0.49, 2), (2, 1)\big), \tag{3.58}$$

$$(\overline{g}, \overline{\mathbf{q}}) = e(g^1, \hat{p}_1) \tag{3.59}$$

$$= \big(g^1,\ \big((\hat{x}_1 + \Delta t \cdot \hat{v}_1,\ \hat{v}_1)\big)\big) \tag{3.60}$$

$$= \big(g^1,\ \big((0 + 0.1 \cdot -1,\ -1)\big)\big) \tag{3.61}$$

$$= \big(g^1,\ \big((-0.1,\ -1)\big)\big)\,, \tag{3.62}$$

$$\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}\big([g^1, ()], 1\big) = \big[\overline{g},\quad () \circ \overline{\mathbf{q}}\big] \tag{3.63}$$

$$= \big[g^1,\quad () \circ \big((-0.1,\ -1)\big)\big] \tag{3.64}$$

$$= \big[g^1, \big((-0.1,\ -1)\big)\big]. \tag{3.65}$$

We use the result $\hat{\mathbf{p}}^1$ of the previous function $\iota^{\mathrm{N\times U}}$ (3.44) – (3.56) as the starting point for this example calculation since ϵ^{I} and $\iota^{\mathrm{N\times U}}$ are connected by the state transition step s. The result $(\overline{g}, \overline{\mathbf{q}})$ of the evolve function $e(g^1, \hat{p}_1)$ is the initial global variable g^1 and the particle $\hat{p}_1$ with a changed position from $\hat{x}_1 = 0$ to $\hat{x}_1 = -0.1$. The velocity stays the same $\hat{v}_1 = -1$. The function ϵ^{I} uses this result to change the state $[g^1, ()]$, which consists of the initial global variable g^1 and an empty particle tuple $()$. The global variable g^1 is changed to $\overline{g}$ according to $e(g^1, \hat{p}_1)$ is $\overline{g} = g^1$. Hence, there is no effective change in g^1. The particle tuple $\overline{\mathbf{q}} = \big((-0.1,\ -1)\big)$ is concatenated (def. 8) to the empty tuple $()$. Hence, the overall result of ϵ^{I} is the state $\big[g^1, \big((-0.1,\ -1)\big)\big]$.

Explanation of the Second Evolution Sub-function ϵ^{N}

$$\epsilon^{\mathrm{N}} : [G \times P^*] \to [G \times P^*] \tag{3.66}$$

$$\epsilon^{\mathrm{N}}\big([g, \mathbf{p}]\big) := \big[g, ()\big] \ *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} \ (1, .., |\mathbf{p}|) \tag{3.14}$$

The second evolution sub-function ϵ^{N} describes how the evolution of all particles contributes to the state change of the particle method. Therefore, it evolves all particles and stores the results.

Formally, it maps a state of the particle method to a state of the particle method. It uses the composition operator $*$ (def. 7) over the first evolution sub-function $\epsilon^{\mathrm{I}}_{\mathbf{p}}$ to iterate over all indices of the particles in the tuple $\mathbf{p}$ and compute their evolution. The function ϵ^{N} handles the evolution of each of these particles and accumulates the results in a new state starting from the state $\big[g, ()\big]$.

In the example from Section 3.3.1:

$$\hat{\mathbf{p}}^1 = \big(\hat{p}_1, \hat{p}_2, \hat{p}_3\big) = \big((0, -1), (0.49, 2), (2, 1)\big), \tag{3.44) – (3.56}$$

$$\epsilon^{\mathrm{N}}\big([g^1, \hat{\mathbf{p}}^1]\big) = \big[g^1, ()\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (1, ..., |\hat{\mathbf{p}}^1|) \tag{3.67}$$

$$= \big[g^1, ()\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (1, 2, 3) \tag{3.68}$$

$$= \epsilon^{\mathrm{I}}\big([g^1, \hat{\mathbf{p}}^1], (), 1\big) \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (2, 3) \tag{3.69}$$

$$= \big[g^1, \big((-0.1, \ -1)\big)\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (2, 3) \tag{3.70}$$

$$= \epsilon^{\mathrm{I}}\big([g^1, \hat{\mathbf{p}}^1], \big((-0.1, \ -1)\big), 2\big) \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (3) \tag{3.71}$$

$$= \big[g^1, \big((-0.1, \ -1) \circ (\hat{x}_2 + \Delta t \cdot \hat{v}_2, \ \hat{v}_2)\big)\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (3) \tag{3.72}$$

$$= \big[g^1, \big((-0.1, \ -1) \circ (0.49 + 0.1 \cdot 2, \ 2)\big)\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (3) \tag{3.73}$$

$$= \big[g^1, \big((-0.1, \ -1), (0.69, \ 2)\big)\big] \ *_{\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}} \ (3) \tag{3.74}$$

$$= \big[g^1, \big((-0.1, \ -1), (0.69, \ 2), (2.1, \ 1)\big)\big]. \tag{3.75}$$

We use the result $\hat{\mathbf{p}}^1$ of $\iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1])$ again, that was calculated above (3.44) – (3.56) for this example, knowing that ϵ^{N} and $\iota^{\mathrm{N}\times\mathrm{U}}$ are connected. Hence, it remains to calculate the evolution of the particles $\hat{p}_1$, $\hat{p}_2$, and $\hat{p}_3$ and add them up into the state $[g^1, ()]$. The evolution of $\hat{p}_1$, namely $\epsilon^{\mathrm{I}}_{\hat{\mathbf{p}}^1}\big([g^1, ()], 1\big)$, has already been calculated above (3.63)–(3.65). The same procedure is followed for the following two particles, $\hat{p}_2$ and $\hat{p}_3$. After each evolution, the resulting particle tuple, with just one element, is concatenated to the previous result, such that they combine into a particle tuple of three particles in the end. In this example, the global variable g^1 remains unchanged by the function ϵ^{N}.

Explanation of the State Transition Step s

$$s : [G \times P^*] \to [G \times P^*] \tag{3.76}$$

$$s\,([g, \mathbf{p}]) := \big[\mathring{e}(\overline{g}), \ \overline{\mathbf{p}}\big] \qquad \text{with} \quad [\overline{g}, \overline{\mathbf{p}}] := \epsilon^{\mathrm{N}}\big([g, \ \iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}])]\big). \tag{3.15}$$

The particle method state transition step s handles the interactions and evolutions necessary to advance a particle method from one state to the next. Therefore it combines the functions $\iota^{\mathrm{N}\times\mathrm{U}}$, ϵ^{N}, and $\mathring{e}$. Moreover, the state transition step s does it sequentially. Formally, the state transition function is a function that maps a state of the particle method to a state of the particle method.

In the example from section 3.3.1:

$$g^1 = (d, t, \Delta t, T) = (0.5, 0, 0.1, 0.1), \tag{3.28}$$

$$\mathbf{p}^1 = \big((0,2), (0.49, -1), (2,1)\big), \tag{3.29}$$

$$\epsilon^{\mathrm{N}}\big([g^1,\ \iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1])]\big) = \big[g^1, \underbrace{\big((-0.1,\ -1), (0.69,\ 2), (2.1,\ 1)\big)}_{=:\mathbf{p}^2}\big], \tag{3.67–3.75}$$

$$s\left([g^1, \mathbf{p}^1]\right) = \left[\mathring{e}(\overline{g}),\ \overline{\mathbf{p}}\right] \quad \text{with} \quad [\overline{g}, \overline{\mathbf{p}}] = \epsilon^{\mathrm{N}}\big([g^1,\ \iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1])]\big) \tag{3.77}$$

$$= \left[\mathring{e}(\overline{g}),\ \overline{\mathbf{p}}\right] \quad \text{with} \quad [\overline{g}, \overline{\mathbf{p}}] = [g^1, \mathbf{p}^2] \tag{3.78}$$

$$= \left[\mathring{e}(g^1),\ \mathbf{p}^2\right] \tag{3.79}$$

$$= \left[(d, t + \Delta t, \Delta t, T),\ \mathbf{p}^2\right] \tag{3.80}$$

$$= \left[(0.5, 0 + 0.1, 0.1, 0.1),\ \mathbf{p}^2\right] \tag{3.81}$$

$$= \big[\underbrace{(0.5, 0.1, 0.1, 0.1)}_{=:g^2},\ \mathbf{p}^2\big] \tag{3.82}$$

$$= \left[g^2,\ \mathbf{p}^2\right]. \tag{3.83}$$

The instance $[g^1, \mathbf{p}^1]$ in this example is given as a reminder, as well as the second evolve sub-function $\epsilon^{\mathrm{N}}\big([g^1,\ \iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1])]\big)$. We define the particle tuple of the result of ϵ^{N} as $\mathbf{p}^2$. This indicates that this particle tuple is already part of the following state $[g^2, \mathbf{p}^2]$. This is true because there will be no further change to it. Since $\epsilon^{\mathrm{N}}\big([g^1,\ \iota^{\mathrm{N}\times\mathrm{U}}([g^1, \mathbf{p}^1])]\big) = [g^1, \mathbf{p}^2]$, only the evolution of the global variable $\mathring{e}$ remains. $\mathring{e}(g^1)$ advances the current time t by one time step size Δt. Hence, the current time increases from $t = 0$ to $t = 0.1$. The rest of the properties of the global variable remain unchanged in this example. We call the new global variable g^2 because this step has no further change.

Explanation of the State Transition Function S

$$S : [G \times P^*] \to [G \times P^*], \tag{3.9}$$

$$\begin{aligned} &S([g^1, \mathbf{p}^1]) = [g^T, \mathbf{p}^T] \quad \longleftrightarrow \\ &\quad f(g^T) = \top \quad \wedge \quad \forall t \in \{2, ..., T\} : \ [g^t, \mathbf{p}^t] = s\left([g^{t-1}, \mathbf{p}^{t-1}]\right) \ \wedge \ f(g^{t-1}) = \bot. \end{aligned} \tag{3.16}$$

The state transition function S iterates over the state transition step s from the instance $[g^1, \mathbf{p}^1]$ until the stopping condition f is *true*.

In the example from Section 3.3.1:

$$[g^1, \mathbf{p}^1] = [(0.5, 0, 0.1, 10),\ \big((0,2), (0.49, -1), (2,1)\big)] \qquad (3.28),(3.29)$$

$$f(g^1) = (t \geq t_{end}) \qquad (3.84)$$
$$= (0 \geq 0.1) \qquad (3.85)$$
$$= \perp, \qquad (3.86)$$

$$s\left([g^1, \mathbf{p}^1]\right) = \big[(0.5, 0.1, 0.1, 10),\ \big((-0.1,\ -1), (0.69,\ 2), (2.1,\ 1)\big)\big] \qquad (3.77)\text{–}(3.83)$$
$$= \big[g^2,\ \mathbf{p}^2\big].$$

$$f(g^2) = (t \geq t_{end}) \qquad (3.87)$$
$$= (0.1 \geq 0.1) \qquad (3.88)$$
$$= \top, \qquad (3.89)$$

$$\rightarrow \quad S\left([g^1, \mathbf{p}^1]\right) = \underline{\big[g^2,\ \mathbf{p}^2\big]} \qquad (3.90)$$

In the example, all calculations are done using the state transition step s of the particle method (3.77) – (3.83). The particle method state transition function S iterates over the state transition step s until the stopping condition f is *true*, which is the case for the state $[g^2, \mathbf{p}^2]$. The particle method halts with the result $[g^2, \mathbf{p}^2]$.

3.4 Conclusion

Particle methods are used in a wide range of fields such as plasma physics [40], computational fluid dynamics [17, 20], image processing [11, 1], computer graphics [36], and computational optimization [38, 63]. But there was no common understanding of what constitutes a particle method.

Here, we presented a general mathematical definition of particle methods and showed its applicability in developing a common interface for this important class of algorithms in computational science. The proposed definition highlights the algorithmic commonalities across applications, enabling a sharp classification of particle methods.

We formulated the presented definition in the most general way to encompass everything called a "particle method". However, most practical instances do not exploit the full generality of the definition.

Even though the presented definition is general a particle method could potentially have a worse time and space complexity than a non-particle algorithm, especially for non-canonical problems but also for algorithms where only a part of the particles are active like for treatment of boundary conditions. Further, our definition is limited by its monolithic nature. An algorithm composed of smaller algorithms, such as a solver for the incompressible Navier-Stokes equation, would become very large and complex with several nested cases when explicitly formulated in our definition. Importantly, our definition is not unique. Alternative, possibly more compact, but equivalent definitions are possible. We chose the presented formulation for its similarities to practical implementations. Notwithstanding these limitations, the present definition establishes a rigorous algorithmic class

that contrasts the so-far loose and empirical notion of particle methods in practice. This rigorousness paves the way for further research both in the theoretical and algorithmic foundations of particle methods and the engineering of their software implementation.

Future work could develop a less monolithic definition that allows modular combinations of different particle methods. While this could lead to a formulation that can potentially be exploited directly in software engineering or the design of domain-specific programming languages for particle methods [48, 50], one would first need to solve some theoretical problems: How can different types of particles from different methods interact, e.g., during interpolating stored values from one set of particles to another? How can access be restricted to a particle subset, e.g., for boundary conditions? Solving these problems might lead to additional data structures or functions in the presented definition.

Overall, formal definitions reveal the concepts upon which a method is founded, and they render it possible to rationalize the fundamental characteristics of a method. The presented definition of particle methods is a necessary first step toward a sound and formal understanding of what particle methods are, what they can do, and how efficient and powerful they can be.

Chapter 4

Algorithms as Particle Methods

4.1 Introduction

Our definition of particle methods is designed to provide a rigorous, mathematical structure to express a broad range of algorithms. Even though our definition of particle methods is derived from the formulation and practical implementation of various particle methods such as Smoothed Particle Hydrodynamics (SPH) [34, 58], Particle Strength Exchange (PSE) [26, 25, 30, 79], Molecular Dynamics (MD) [53], Discrete Element Methods (DEM) and others, it remains to be shown that the structure of the definition is naturally capable of expressing these classic particle methods.

Here, we fill this gap by presenting a range of classic particle methods and also non-canonical algorithms in the structure of our particle methods definition. We show how classic particle methods can be formalized using our definition by considering the examples of SPH and PSE as continuous particle methods and MD and DEM as discrete particle methods. We further show how other algorithms, not generally recognized as particle methods, can also be cast in terms of our definition. As examples, we consider triangulation refinement [28], Conway's game of life [32], and Gaussian elimination.

4.2 Perfectly Elastic Collision in Arbitrary Dimensions

The perfectly elastic collision example is the same as the illustrative example from the section 3.3.1 but in an arbitrary dimension. Hence, it models perfectly elastic collisions between spheres of uniform diameter d and constant unit mass in a continuous n-dimensional space. The space is without boundaries, such that no boundary conditions are required.

In this example, position x of an individual sphere changes over time t, as shown in section 3.3.1. The explicit Euler time-stepping algorithm discretizes this again in time using a fixed time step size Δt, yielding:

$$\underline{x}^{t+\Delta t} = \underline{x}^{t} + \underline{v}(t)\Delta t, \tag{4.1}$$

which is iterated until a given final time t_{end}. A perfectly elastic collision between two spheres j and k in n-dimensions results in them swapping the portion of their velocities in the direction of the other particle. Hence, the change in velocity is the difference between the two velocities projected down onto the normalized direction vector from one particle center to the other. The positions before the collision $\underline{x}^n$ and the positions after the collision $\underline{x}^{n+1}$ remain the same, leading to the following collision rules:

$$\underline{\Delta v}_{jk} := \frac{\underline{x}_{jk}}{|\underline{x}_{jk}|^2}\left(\underline{x}_{jk} \cdot \underline{v}_{jk}\right) \qquad \text{with} \quad \underline{v}_{jk} := \underline{v}_k - \underline{v}_j, \quad \underline{x}_{jk} := \underline{x}_k - \underline{x}_j \tag{4.2}$$

$$\underline{x}_j^{n+1} = \underline{x}_j^n, \tag{4.3}$$

$$\underline{x}_k^{n+1} = \underline{x}_k^n, \tag{4.4}$$

$$\underline{v}_j^{n+1} = \underline{v}_j^n + \underline{\Delta v}_{jk}, \tag{4.5}$$

$$\underline{v}_k^{n+1} = \underline{v}_k^n - \underline{\Delta v}_{jk}. \tag{4.6}$$

Two spheres j and k are considered colliding if and only if $|\underline{x}_{jk}| \leq d$. Note that this particle method can express unphysical behavior beyond the obvious if the particles are initialized closer than d and if more than one particle is closer to a particle than d at a time step. This defines the particle method:

$$p := (\underline{x}, \underline{v}) \quad \text{for } p \in P := \mathbb{R}^n \times \mathbb{R}^n, \tag{4.7}$$

$$g := (d, \Delta t, t_{end}, t) \quad \text{for } g \in \mathbb{R}^4, \tag{4.8}$$

$$u([g, \mathbf{p}], j) := (k \in (1, ..., |\mathbf{p}|) : \; p_k, p_j \in \mathbf{p} \;\wedge\; |\underline{x}_{jk}| \leq d \wedge k > j), \tag{4.9}$$

$$f(g) := (t > t_{end}), \tag{4.10}$$

$$i(g, p_j, p_k) := \left(\begin{pmatrix} \underline{x}_j \\ \underline{v}_j + \underline{\Delta v}_{jk} \end{pmatrix}^{\mathbf{T}}, \begin{pmatrix} \underline{x}_k \\ \underline{v}_k - \underline{\Delta v}_{jk} \end{pmatrix}^{\mathbf{T}} \right) \tag{4.11}$$

$$e(g, p_j) := \left(g, \left(\begin{pmatrix} \underline{x}_j + \Delta t \; \underline{v}_j \\ \underline{v}_j \end{pmatrix}^{\mathbf{T}} \right) \right), \tag{4.12}$$

$$\mathring{e}(g) := (d, \Delta t, t_{end}, t + \Delta t). \tag{4.13}$$

In this particle method algorithm, a particle has a position $\underline{x}$ and a velocity $\underline{v}$. The global variable g has the sphere diameter d that is the same for all particles and the properties for the time management, the time step size Δt, the end time t_{end}, and the current time t.

The neighborhood function ensures that just closer particles than d are interacting. Since the interaction is symmetric, u takes care that if p_k is in the neighborhood of p_j, p_j is not in the neighborhood of p_k. The simulation stops if the current time t exceeds the end time t_{end}, defined in the stopping function f. The interact function i calculates what happens during a collision (4.2)–(4.6). The evolve function e calculates the Euler time step (4.1). The evolve function of the global variable $\mathring{e}$ advances the current time t by the time step size Δt.

We need to fix the parameters and the initial condition to define a specific instance of this particle method. Here we choose a small instance with just three particles in two dimensions.

$g^1 := (d, \Delta t, t_{end}, t)$	initial global variable
$d := 0.5$	sphere diameter
$\Delta t := 0.1$	time step size
$t_{end} := 10$	end time of the simulation
$t := 0$	current time
$\mathbf{p}^1 := (p_1, p_2, p_3)$	initial particle tuple
$p_j := (\underline{x}_j, \underline{v}_j)$	j-th particle
$p_1 := \left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}0.25\\0\end{pmatrix}\right)$	
$p_2 := \left(\begin{pmatrix}1\\0.2\end{pmatrix}, \begin{pmatrix}-0.5\\0\end{pmatrix}\right)$	
$p_3 := \left(\begin{pmatrix}-0.1\\0.8\end{pmatrix}, \begin{pmatrix}0.2\\-0.7\end{pmatrix}\right)$	

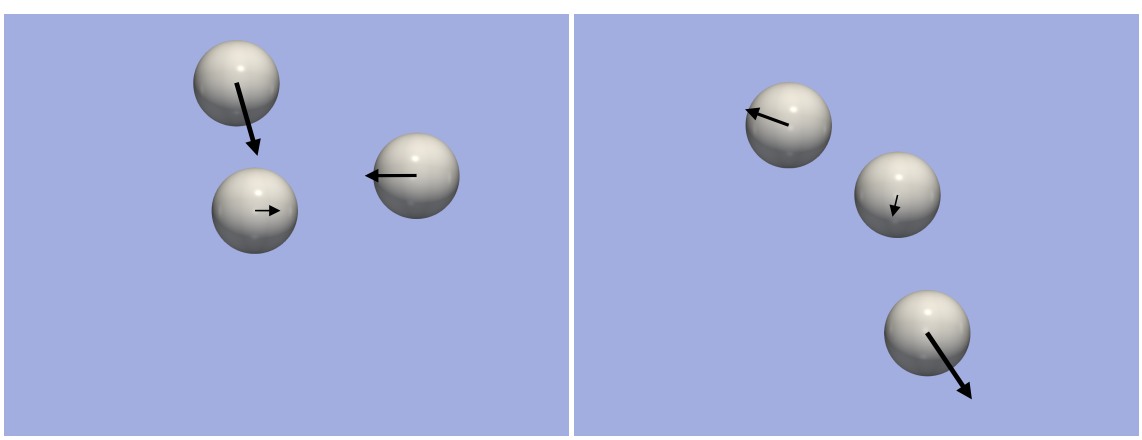


Figure 4.1: Perfectly elastic collision of three spheres in two dimensions. The arrows indicate the direction and magnitude of the velocities. Left at $t = 0.1$. Right at $t = 1.6$. (Visualized using ParaView [85].)

4.3 Particle Strength Exchange

Particle Strength Exchange (PSE) is a classic particle method that numerically solves partial differential equations in time and space [26, 25, 30, 79]. It provides a general framework for numerically approximating differential operators over sets of irregularly placed collocation points called particles. Here, we consider the example of using PSE to numerically solve the isotropic, homogeneous, and normal diffusion equation in three dimensions:

$$\frac{\partial w(\underline{x}, t)}{\partial t} = D \; \mathbf{\Delta} w(\underline{x}, t) \tag{4.14}$$

for the continuous and sufficiently smooth function $w(\underline{x}, t) : \mathbb{R}^4 \to \mathbb{R}$. We use the explicit Euler method for time integration and PSE for space discretization on equidistant points with spacing h. PSE approximates the Laplace operator $\mathbf{\Delta} w$, a second-order differential operator in space, at location $\underline{x}_j$ using the surrounding particles at positions $\underline{x}_k$ as [26]:

$$\mathbf{\Delta} w \; (\underline{x}_j) \approx \frac{h^3}{\epsilon^2} \sum_{k=1}^{N} \big(w(\underline{x}_k) - w(\underline{x}_j) \big) \, \eta_\epsilon(\underline{x}_j - \underline{x}_k). \tag{4.15}$$

Using PSE theory, we determine the operator kernel η_ϵ such as to yield an approximation error that converges with the square of the kernel width ϵ:

$$\eta_\epsilon(\underline{x}) = \frac{15}{\epsilon^3 \pi^2} \frac{1}{\left(\frac{|\underline{x}|}{\epsilon}\right)^{10} + 1}. \tag{4.16}$$

The kernel's support is $[-\infty, \infty]$. However, the exponential quickly drops below the machine precision of a digital computer, so it is custom to introduce a cut-off radius r_c to limit particle interactions to non-trivial computations. The approximation of the Laplace operator then is:

$$\mathbf{\Delta} w \; (\underline{x}_j) \approx \frac{15 h^3}{\epsilon^3 \pi^2} \sum_{\underline{x}_k :\, 0 < |\underline{x}_k - \underline{x}_j| \leq r_c} \frac{w(\underline{x}_k) - w(\underline{x}_j)}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1} \tag{4.17}$$

The explicit Euler method allows the approximation of the continuous time derivative $\frac{\partial w}{\partial t}$ at discrete points in time t_n, $n \in \mathbb{N}$ with time step size $\Delta t := t_{n+1} - t_n$:

$$\frac{\partial w}{\partial t}(t_n) \approx \frac{w(t_{n+1}) - w(t_n)}{\Delta t} \tag{4.18}$$

Hence, the above differential equation is discretized as:

$$w(\underline{x}_j, t_{n+1}) \tag{4.19}$$

$$\approx w(\underline{x}_j, t_n) + D \; \Delta t \; \mathbf{\Delta} w(\underline{x}_j) \tag{4.20}$$

$$\approx w(\underline{x}_j, t_n) + \tfrac{15 h^3 D \Delta t}{\epsilon^3 \pi^2} \sum_{x_k : 0 < |\underline{x}_k - \underline{x}_j| \leq r_c} \frac{w(\underline{x}_k, t_n) - w(\underline{x}_j, t_n)}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1}. \tag{4.21}$$

To numerically solve (4.14), this expression is evaluated over the particles at locations $\underline{x}_j$ with property w_j at time points t_n. For simplicity, we consider a free-space simulation

without boundary conditions. Hence, we assume that an instance of this particle method has enough particles with no or low concentration w_j around the region of interest in the initial tuple of particles. We further assume that the particles are regularly spaced with inter-particle spacing h such that $\frac{h}{\epsilon} \leq 1$. This is a theoretical requirement in PSE known as the "overlap condition". Without it, the numerical method is not consistent. This defines the particle method algorithm data structures:

$$p := (\underline{x}, w, \Delta w) \quad \text{for } p \in P := \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R}, \tag{4.22}$$

$$g := (D, h, \epsilon, r_c, \Delta t, t_{end}, t) \quad \text{for } g \in \mathbb{R}^7, \tag{4.23}$$

and functions:

$$u\left([g, \mathbf{p}], j\right) := \left(k \in (1, ..., |\mathbf{p}|) : \; p_k, p_j \in \mathbf{p} \; \wedge |\underline{x}_k - \underline{x}_j| \in (1, r_c]\right), \tag{4.24}$$

$$f(g) := (t > t_{end}), \tag{4.25}$$

$$i\left(g, p_j, p_k\right) := \left(\begin{pmatrix} \underline{x}_j \\ w_j \\ \Delta w_j + \frac{w_k - w_j}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1} \end{pmatrix}^{\mathbf{T}}, \; p_k \right), \tag{4.26}$$

$$e\left(g, p_j\right) := \left(g, \left(\begin{pmatrix} \underline{x}_j \\ w_j + \Delta t \frac{15 D h^3}{\epsilon^5 \pi^2} \Delta w_j \\ 0 \end{pmatrix}^{\mathbf{T}} \right) \right), \tag{4.27}$$

$$\mathring{e}\left(g\right) := \left(D, h, \epsilon, r_c, \Delta t, t_{end}, t + \Delta t\right). \tag{4.28}$$

Each particle p represents a collocation point of the numerical scheme. It is a collection of three properties, each of which is a real vector/ number: the position $\underline{x}$, the concentration w, and the accumulator variable Δw that collects the concentration in the interact function i. An accumulator variable is required here to render the computation result independent of the indexing order of the particles.

The global variable g is a collection of seven real-valued properties that are accessible throughout the whole calculation: the diffusion constant D, the spacing between particles h, the kernel width ϵ, the cut-off radius r_c, the time step size Δt, the end time of the simulation t_{end}, and the current time t.

The neighborhood function u returns the surrounding particles no further away than the cut-off radius r_c and different from the query particle itself. The stopping condition f is true ($\top$) if the current time t exceeds the end time t_{end}. Then the simulation halts.

The interact function i evaluates the sum in the PSE approximation (4.17). Each particle p_j accumulates its concentration change in Δw_j during the interactions with the other particles. In the present example, we choose an asymmetric/ pull interact function i, just changing particle p_j. The neighborhood function u accounts for this. However, this is unnecessary, and symmetric formulations of PSE are also possible.

The evolve function e uses the accumulated change Δw_j to update the concentration w_j of particle p_j using the explicit Euler method (4.21). For that, it also uses D, h, ϵ, and Δt from the global variable g. In addition, the evolve function e resets the accumulator Δw_j to 0. In this example, the evolve function does not change the global variable g. That is exclusively done in $\mathring{e}$, which updates the current time t by adding the time step size Δt.

We need to fix the parameters and the initial condition to define a specific instance of this particle method. We choose a box where $w = 0$ for all particles except for the center, where we place a concentration peak.

$$g^1 := \big(\underbrace{0.01}_{D}, \underbrace{0.02}_{h}, \underbrace{0.02}_{\epsilon}, \underbrace{0.06}_{r_c}, \underbrace{0.005}_{\Delta t}, \underbrace{0.5}_{t_{end}}, \underbrace{0}_{t}\big) \tag{4.29}$$

$$\mathbf{p}^1 := (p_1, ..., p_{51^3}), \qquad p_j := (\underline{x}_j, w_j, \Delta w_j) \tag{4.30}$$

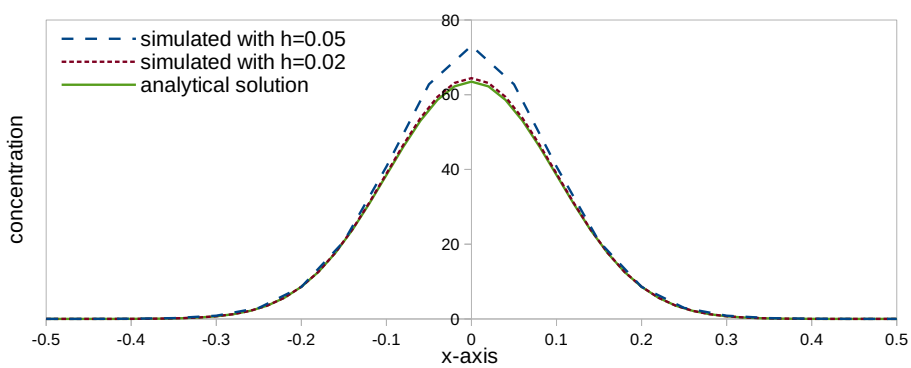
For $j \in \{1, ..., 51^3\} \backslash \{\frac{51^3+1}{2}\}$ we can uniquely represent j as $j = j_1 + j_2 51 + j_3 51^2$ where $j_1, j_2, j_3 \in \{0, ..., 50\}$, then we set p_j as

$$p_j := ((h(j_1 - 25), h(j_2 - 25), h(j_3 - 25)), 0, 0), \tag{4.31}$$

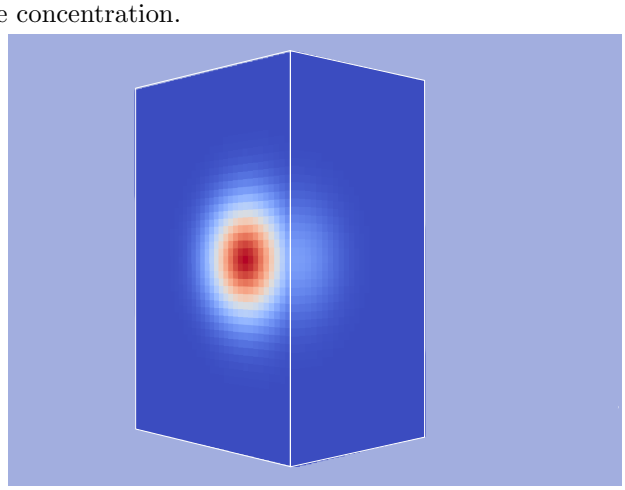
and we set

$$p_{\frac{51^3+1}{2}} := \big((0, 0, 0), h^{-3}, 0\big). \tag{4.32}$$

The result of executing this instance is visualized for time $t = 0.5$ in figure 4.2.



(a) Simulation compared with the analytical solution. It is the plot of the values along the x-axis vs. the concentration.



(b) Visualization of the simulation with a clipped domain for better visibility (visualized using ParaView [85]).

Figure 4.2: Particle Strength Exchange simulation of diffusion in three-dimensions at $t = 0.5$.

4.4 Smoothed Particle Hydrodynamics

We use Smoothed Particle Hydrodynamics (SPH) [34, 58] to showcase our definition applies to one of the most popular particle methods. The example we chose is a dam break fluid simulation based on the Navier-Stokes equations.

PSE and SPH are related. However, they follow different but similar approaches. The main difference is that in SPH, the kernels for the derivative interpolation are derivatives of the function interpolation kernel, while in PSE, all kernels are potentially unrelated.

The Navier-Stokes equation of momentum

$$\frac{D\underline{v}}{Dt} = -\frac{1}{\rho}\nabla\mathfrak{p} + \nabla\cdot\left(\nu\left(\nabla\underline{v} + \nabla\underline{v}^{\mathbf{T}}\right)\right) + \underline{g} \tag{4.33}$$

describes the motion of a viscous fluid. The parameter ν denotes the kinematic viscosity, and $\underline{g}$ is the gravitational force. The unknown fields are the velocity $\underline{v}$, the density ρ, and the pressure $\mathfrak{p}$. The density convergence equation

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot\underline{v} \tag{4.34}$$

describes mass conservation. The momentum equation and the density convergence equation are, in this case, coupled with the Cole [18] equation of state

$$\mathfrak{p}(\rho) := \frac{c_0^2\ \rho_0}{\gamma}\ \left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right). \tag{4.35}$$

The parameter c_0 is the so-called speed of sound of the fluid, ρ_0 is the reference density, and γ is the polytropic index. The choice of the parameters of the equation of state strongly influences the density fluctuations.
The system of equations is discretized by the method of SPH [34, 62, 58] using a smoothing kernel function. We use a fifth-order Wendland kernel [90]

$$W_{jk} = \begin{cases} \frac{21}{16\ \pi\ h^3}\left(1 - \frac{|\underline{x}_{jk}|}{2h}\right)^4\left(1 + 2\frac{|\underline{x}_{jk}|}{h}\right) & \text{if } 0 \le |\underline{x}_{jk}| \le 2h, \\ 0 & \text{else} \end{cases}, \tag{4.36}$$

and its derivative

$$\nabla W_{jk} = -\underline{x}_{jk}\ F_{jk} = \begin{cases} -\underline{x}_{jk}\left(\underbrace{\frac{-5\cdot 21}{16\pi\ h^5}}_{=:\overline{\delta}}\underbrace{\left(1 - \frac{|\underline{x}_{jk}|}{2h}\right)^3}_{=:\overline{F}_{jk}}\right) & \text{if } 0 \le |\underline{x}_{jk}| \le 2h, \\ 0 & \text{else} \end{cases}, \tag{4.37}$$

where h scales the kernel support and $\underline{x}_{jk} := \underline{x}_k - \underline{x}_j$. Using the Wendland kernel and its derivative, we arrive at the discretized momentum equation

$$\frac{D\underline{v}_j}{Dt} \approx \sum_{k\in K}\left(\underbrace{\frac{\mathfrak{p}_j}{\rho_j^2} + \frac{\mathfrak{p}_k}{\rho_k^2}}_{=:\ \Omega_{jk}} + \frac{\Pi_{jk}}{\rho_j}\right)\underline{x}_{jk}\ \overline{\delta}\ m_j\ \overline{F}_{jk} + \underline{g}, \tag{4.38}$$

and discretized density convergence equation

$$\frac{D\rho_j}{Dt} = \sum_{k \in K} \underline{v}_{jk} \cdot \underline{x}_{jk} \; \overline{\delta} \; m_j \; \overline{F}_{jk}, \tag{4.39}$$

with $\underline{v}_{jk} := \underline{v}_k - \underline{v}_j$. The parameter m_j denotes the mass of a particle, which we choose to be and stay the same for all particles, and

$$\frac{\Pi_{jk}}{\rho_j} := -\frac{10\nu}{\rho_j} \; \frac{\underline{v}_{jk} \cdot \underline{x}_{jk}}{|\underline{x}_{jk}|^2} \tag{4.40}$$

is the artificial viscosity term describing the viscous contribution of the momentum equation. Having the spatial discretization of the partial differential equation system, we now use a predictor-corrector integration scheme [95] to discretize time. The scheme has two phases. The first calculates a half step used in the second phase to calculate a corrected full step.
First phase:

$$\underline{\Delta v}^n = \underline{\Delta v}\left(\underline{x}^n, \underline{v}^n, \rho^n\right) = \left.\frac{D\underline{v}}{Dt}\right|_{t=t_n} \tag{4.41}$$

$$\Delta\rho^n = \Delta\rho\left(\underline{x}^n, \underline{v}^n, \rho^n\right) = \left.\frac{D\rho}{Dt}\right|_{t=t_n} \tag{4.42}$$

$$\underline{x}^{n+\frac{1}{2}} = \underline{x}^n + \frac{\Delta t}{2} \; \underline{v}^n \tag{4.43}$$

$$\underline{v}^{n+\frac{1}{2}} = \underline{v}^n + \frac{\Delta t}{2} \; \underline{\Delta v}^n \tag{4.44}$$

$$\rho^{n+\frac{1}{2}} = \rho^n + \frac{\Delta t}{2} \; \Delta\rho^n \tag{4.45}$$

Second phase:

$$\underline{\Delta v}^{n+\frac{1}{2}} = \underline{\Delta v}\left(\underline{x}^{n+\frac{1}{2}}, \underline{v}^{n+\frac{1}{2}}, \rho^{n+\frac{1}{2}}\right) \tag{4.46}$$

$$\Delta\rho^{n+\frac{1}{2}} = \Delta\rho\left(\underline{x}^{n+\frac{1}{2}}, \underline{v}^{n+\frac{1}{2}}, \rho^{n+\frac{1}{2}}\right) \tag{4.47}$$

$$\underline{x}^{n+1} = \underline{x}^n + \Delta t\left(\underline{v}^n + \frac{\Delta t}{2} \; \underline{\Delta v}^{n+\frac{1}{2}}\right) \tag{4.48}$$

$$\underline{v}^{n+1} = \underline{v}^n + \Delta t \;\; \underline{\Delta v}^{n+\frac{1}{2}} \tag{4.49}$$

$$\rho^{n+1} = \rho^n + \Delta t \;\; \Delta\rho^{n+\frac{1}{2}} \tag{4.50}$$

So far, we have described the behavior of the fluid, not the geometry in which it flows. Hence we need boundary conditions. We choose a pool with a pillar. SPH doesn't need boundary conditions for free surfaces. Hence it is enough to describe the solid walls of the pool and the pillar. We use the so-called dynamic boundary conditions [24], where fluid particles represent the boundary. These boundary particles have constant zero velocity and therefore do not move. Thus, the remaining properties of the particles do not differ

from the standard fluid particles.

In the particle method algorithm, we have these two types of particles, the fluid particles and the boundary particles, the property b indicates this. For boundary particles, is $b = \top$, and for fluid particles, $b = \bot$. In addition, each particle has a position $\underline{x}$, velocity $\underline{v}$, density ρ, and for each of these exists a storage for the previous value $(\underline{x}^{old}, \underline{v}^{old}, \rho^{old})$. Furthermore, the particles have two accumulators for the velocity change $\underline{\Delta v}$ and density change $\Delta\rho$.

The global variable g consists of the parameter φ that indicates the current predictor-corrector phase, $\varphi = 0$ for the first phase, and $\varphi = 1$ for the second, and further SPH- and physical parameters as well as parameters relevant for the time management. This defines the particle method algorithm as follows:

$$
\begin{aligned}
p :=& (b, \underline{x}, \underline{x}^{old}, \underline{v}, \underline{v}^{old}, \rho, \rho^{old}, \underline{\Delta v}, \Delta\rho) && (4.51)\\
&\text{for } p \in P := \{\bot, \top\} \times \left(\mathbb{R}^3\right)^4 \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^3 \times \mathbb{R},\\
g :=& (\varphi, \underline{\mathfrak{g}}, \rho_0, m, h, r_c, c_0, \nu, \gamma, \Delta t, t_{end}, t) && (4.52)\\
&\text{for } g \in G := \{0,1\} \times \mathbb{R}^3 \times (\mathbb{R})^{10},\\
u([g, \mathbf{p}], j) :=& (k \in (1, ..., |\mathbf{p}|) : \ p_k, p_j \in \mathbf{p} \ \wedge \ |\underline{x}_{jk}| \leq r_c \ \wedge \ j < k), && (4.53)\\
f(g) :=& (t > t_{end}), && (4.54)
\end{aligned}
$$

$$
i(g, p_j, p_k) := \left(\begin{pmatrix} b_j \\ \underline{x}_j \\ \underline{x}_j^{old} \\ \underline{v}_j \\ \underline{v}_j^{old} \\ \rho_j \\ \rho_j^{old} \\ \underline{\Delta v}_j + \left(\Omega_{jk} + \frac{\Pi_{jk}}{\rho_j}\right) \underline{x}_{jk} \overline{F}_{jk} \\ \Delta\rho_j + \underline{v}_{jk} \cdot \underline{x}_{jk} \ \overline{F}_{jk} \end{pmatrix}^{\mathbf{T}}, \begin{pmatrix} b_k \\ \underline{x}_k \\ \underline{x}_k^{old} \\ \underline{v}_k \\ \underline{v}_k^{old} \\ \rho_k \\ \rho_k^{old} \\ \underline{\Delta v}_k - \left(\Omega_{jk} + \frac{\Pi_{jk}}{\rho_k}\right) \underline{x}_{jk} \overline{F}_{jk} \\ \Delta\rho_k + \underline{v}_{jk} \cdot \underline{x}_{jk} \ \overline{F}_{jk} \end{pmatrix}^{\mathbf{T}} \right), \tag{4.55}
$$

$$
e(g, p_j) := \left(g, \begin{pmatrix} b_j \\ \begin{cases} \underline{x}_j + \frac{\Delta t}{2} v_j & \text{if } \; b_j = \perp \wedge \varphi = 0 \\ \underline{x}_j^{old} + \Delta t \left(\underline{v}_j^{old} + \frac{\Delta t}{2} (\underline{\mathfrak{g}} + \delta \; \underline{\Delta v}_j) \right) & \text{if } \; b_j = \perp \wedge \varphi = 1 \\ \underline{x}_j & \text{else} \end{cases} \\ \begin{cases} \underline{x}_j & \text{if } \; b_j = \perp \wedge \varphi = 0 \\ \underline{x}_j^{old} & \text{else} \end{cases} \\ \begin{cases} \underline{v}_j + \frac{\Delta t}{2} (\underline{\mathfrak{g}} + \delta \; \underline{\Delta v}_j) & \text{if } \; b_j = \perp \wedge \varphi = 0 \\ \underline{v}_j^{old} + \Delta t (\underline{\mathfrak{g}} + \delta \; \underline{\Delta v}_j) & \text{if } \; b_j = \perp \wedge \varphi = 1 \\ \underline{v}_j & \text{else} \end{cases} \\ \begin{cases} \underline{v}_j & \text{if } \; b_j = \perp \wedge \varphi = 0 \\ \underline{v}_j^{old} & \text{else} \end{cases} \\ \begin{cases} \rho_j + \frac{\Delta t}{2} \delta \; \Delta \rho_j & \text{if } \; \varphi = 0 \\ \rho_j^{old} + \Delta t \; \delta \; \Delta \rho_j & \text{else} \end{cases} \\ \begin{cases} \rho_j & \text{if } \; \varphi = 0 \\ \rho_j^{old} & \text{else} \end{cases} \\ 0 \\ 0 \end{pmatrix}^{\mathbf{T}} \right), \tag{4.56}
$$

$$
\mathring{e}(g) := \begin{cases} (1, \underline{\mathfrak{g}}, \rho_0, m, h, r_c, c_0, \nu, \gamma, \Delta t, t_{end}, t) & \text{if } \varphi = 0 \\ (0, \underline{\mathfrak{g}}, \rho_0, m, h, r_c, c_0, \nu, \gamma, \Delta t, t_{end}, t + \Delta t) & \text{else} \end{cases} . \tag{4.57}
$$

The neighborhood function for symmetric interactions depends on the distance between two particles. The stopping condition f is true if the current time t exceeds the end time t_{end}. The interaction function i takes care of the accumulation of the velocity change $\underline{\Delta v}$ and density change $\Delta\rho$. It evaluates the calculations inside the sums of (4.38) and (4.39) without the $\overline{\delta}\ m$. We combine the two parameters to $\delta := \overline{\delta}\ m$. The factor of δ is accounted for in the evolve method.

The evolve method is a bit more complex. It needs to distinguish between boundary and fluid particles and between the first and second predictor-corrector phases. Hence, it calculates the time integration, sets the storage of the previous properties, and sets the accumulators back to 0.

The evolve method of the global variable switches the predictor-corrector phase φ back and forth and advances the current time in the second phase $\varphi = 1$.

We need to fix the parameters and the initial condition to define a specific instance of this particle method. Here we set boundary particles in the form of a pool with a column and place fluid particles in a block inside the pool. The parameters of the global variable

are chosen such that the fluid particles behave roughly like water.

$g^1 := (\varphi, \underline{\mathfrak{g}}, \rho_0, m, h, r_c, c_0, \nu, \gamma, \Delta t, t_{end}, t)$	initial global variable
$\varphi := 0$	phase of the algorithm
$\underline{\mathfrak{g}} := (0, 0, -9.81)$	directed gravitation
$\rho_0 := 1000$	reference density
$m := \dfrac{\rho_0}{64^3}$	particle mass
$h := \dfrac{1.3}{64}$	characteristic kernel length
$r_c := 2h$	interaction cut-off radius
$c_0 := 45$	speed of sound
$\nu := 10^{-4}$	viscosity parameter
$\gamma := 7$	polytropic index
$\Delta t := 0.00005$	time step size
$t_{end} := 3.5$	end time of the simulation
$t := 0$	current time

$\mathbf{p}^1 := \mathbf{p}^{fluid} \circ \mathbf{p}^{lW} \circ \mathbf{p}^{rW} \circ \mathbf{p}^{fW} \circ \mathbf{p}^{bW} \circ \mathbf{p}^{dW} \circ \mathbf{p}^{lP} \circ \mathbf{p}^{rP} \circ \mathbf{p}^{fP} \circ \mathbf{p}^{bP} \circ \mathbf{p}^{tP}$	initial particle tuple containing fluid, wall, and pillar particles
$p_j := (b_j, \underline{x}_j, \underline{x}_j^{old}, \underline{v}_j, \underline{v}_j^{old}, \rho_j, \rho_j^{old}, \underline{\Delta v}_j, \Delta\rho_j)$	j-th particle
$b_j := \begin{cases} 0 & \text{if } p_j \in \mathbf{p}^{fluid} \\ 1 & \text{else} \end{cases}$	indicator if a particle is a fluid or a boundary particle

$x^{fluid}_{\iota^{33,\ 63,\ 65}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j+1,\ k+1,\ l+1)^{\mathbf{T}}$	position of fluid particles
$x^{lW}_{\iota^{2,\ 69,\ 65}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j-2,\ k-2,\ l)^{\mathbf{T}}$	position of left wall particles
$x^{rW}_{\iota^{2,\ 69,\ 65}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j+193,\ k-2,\ l)^{\mathbf{T}}$	position of right wall particles
$x^{fW}_{\iota^{193,\ 2,\ 65}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j,\ k-2,\ l)^{\mathbf{T}}$	position of front wall particles
$x^{bW}_{\iota^{193,\ 2,\ 65}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j,\ k+65,\ l)^{\mathbf{T}}$	position of back wall particles
$x^{dW}_{\iota^{197,\ 69,\ 2}_{j,\ k,\ l}} := \dfrac{1}{64}\ (j-2,\ k-2,\ l-2)^{\mathbf{T}}$	position of bottom wall particles

$$x^{lP}_{\iota^{2,\ 21,\ 65}_{j,\ k,\ l}} := \frac{1}{64}\ (j+126,\ k+22,\ l)^{\mathbf{T}} \qquad \text{position of left pillar particles}$$

$$x^{rP}_{\iota^{2,\ 21,\ 65}_{j,\ k,\ l}} := \frac{1}{64}\ (j+145,\ k+22,\ l)^{\mathbf{T}} \qquad \text{position of right pillar particles}$$

$$x^{fP}_{\iota^{17,\ 2,\ 65}_{j,\ k,\ l}} := \frac{1}{64}\ (j+128,\ k+22,\ l)^{\mathbf{T}} \qquad \text{position of front pillar particles}$$

$$x^{bP}_{\iota^{17,\ 2,\ 65}_{j,\ k,\ l}} := \frac{1}{64}\ (j+128,\ k+41,\ l)^{\mathbf{T}} \qquad \text{position of back pillar particles}$$

$$x^{tP}_{\iota^{21,\ 21,\ 2}_{j,\ k,\ l}} := \frac{1}{64}\ (j+126,\ k+22,\ l+65)^{\mathbf{T}} \qquad \text{position of top pillar particles}$$

$$\underline{x}^{old}_j := (0,0,0)^{\mathbf{T}} \qquad \text{storage for previous position}$$

$$\underline{v}_j := (0,0,0)^{\mathbf{T}} \qquad \text{velocity}$$

$$\underline{v}^{old}_j := (0,0,0)^{\mathbf{T}} \qquad \text{storage for previous velocity}$$

$$\rho_j := 1000 \qquad \text{density}$$

$$\rho^{old}_j := 0 \qquad \text{storage for previous density}$$

$$\underline{\Delta v}_j := (0,0,0)^{\mathbf{T}} \qquad \text{accumulator for velocity change}$$

$$\Delta\rho_j := 0 \qquad \text{accumulator for density change}$$

The result of the execution is visualized in figure 4.3 for time $t = \frac{2}{3}$.

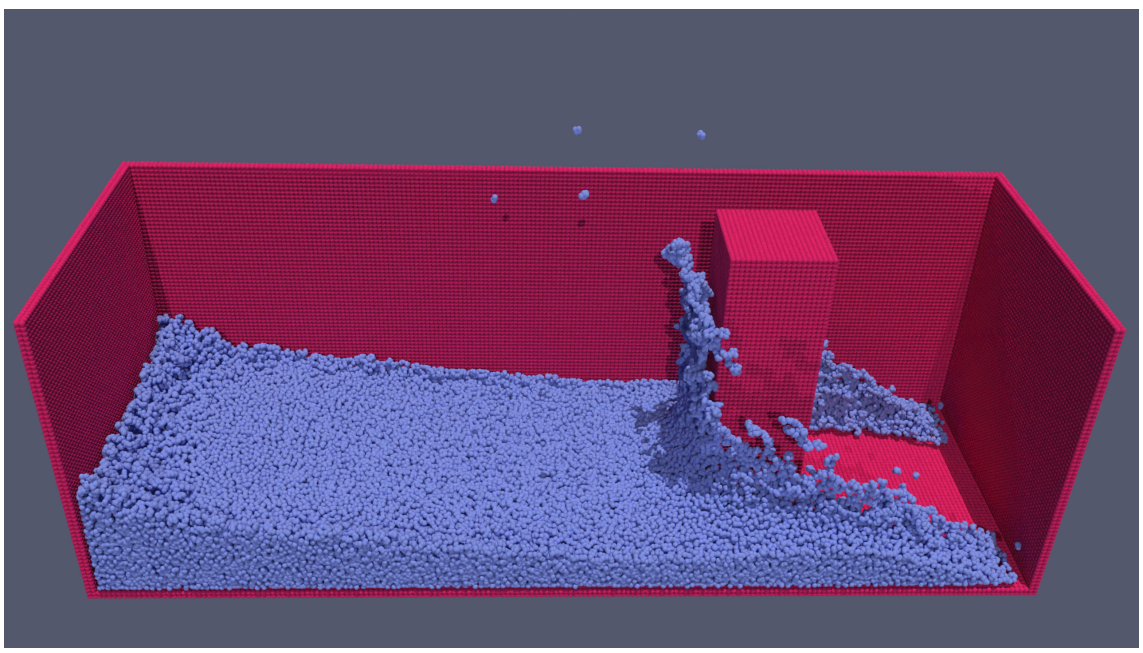


Figure 4.3: Dam Break simulation using smoothed particle hydrodynamics at the time $t = \frac{2}{3}$. The wall facing the viewer is clipped for better visualization (visualized using ParaView [85]).

4.5 Lennard-Jones Molecular Dynamics

Particle methods cannot only be used to discretize continuous models, such as the PDE from the previous example but also to simulate discrete models. A famous example is molecular dynamics, where the Newtonian mechanics of a collection of discrete atoms or molecules is simulated by evaluating their interaction forces. A classic molecular dynamics simulation is that of a so-called Lennard-Jones gas, a dilute collection of electrically neutral, inert, and mono-atomic molecules that interact with each other according to the Lennard-Jones potential [53]

$$U(r) = 4\epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right). \tag{4.58}$$

This potential defines the energy of the interaction as a function of the distance r between two atoms. It is a good approximation of how noble gases behave in function of two parameters: σ that controls the equilibrium distance between atoms and hence the target density of the gas, and ϵ that defines the strength of the interactions by setting the energy scale of the potential.

From this interaction potential, the force between any two atoms a distance r apart can be computed as $F = -\frac{\mathrm{d}U(r)}{\mathrm{d}r}$. From the forces summed up over all other atoms, the acceleration of each atom is computed from Newton's law $F = ma$, where m is the atom's mass. The acceleration is then used to update the velocity and position of the atoms using a symplectic (i.e., energy-conserving) time-stepping method, such as the velocity Verlet scheme [87, 82].

For simplicity, we choose $\epsilon = \sigma = m = 1$. The acceleration between two atoms j and k a distance $r_{jk} := x_k - x_j$ apart then is:

$$a(r_{jk}) = \frac{24}{r_{jk}^{7}} - \frac{48}{r_{jk}^{13}}. \tag{4.59}$$

This acceleration is then used in the velocity-Verlet method [87] to update the velocity v and position x of an atom over a time step Δt, from t_n to t_{n+1}, as:

$$\text{compute } a(t_n) \text{ from } x(t_n), \tag{4.60}$$

$$v(t_n) = v(t_{n-\frac{1}{2}}) + \frac{\Delta t}{2}\, a(t_n), \tag{4.61}$$

$$v(t_{n+\frac{1}{2}}) = v(t_n) + \frac{\Delta t}{2}\, a(t_n), \tag{4.62}$$

$$x(t_{n+1}) = x(t_n) + \Delta t\; v(t_{n+\frac{1}{2}}). \tag{4.63}$$

We choose to have periodic boundary conditions at the boundaries of the simulation domain $x \in [0, D)$. A directed distance function imposes these boundary conditions

$$d_D(x, y) := \begin{cases} y - x - D & \text{if} \quad y - x > \frac{1}{2}D, \\ y - x + D & \text{if} \quad y - x \leq -\frac{1}{2}D, \\ y - x & \text{else} \end{cases} \tag{4.64}$$

and through the modulo operator

$$a \bmod \ b := r \quad \Longleftrightarrow \quad a = b \cdot c + r \quad \text{with} \quad a, b \in \mathbb{R},\ c \in \mathbb{Z},\ r \in [0, |b|)\,, \tag{4.65}$$

that we use to adapt (4.63) to

$$x(t_{n+1}) = \left(x(t_n) + \Delta t\ v(t_{n+\frac{1}{2}})\right) \bmod D. \tag{4.66}$$

Each atom is represented as a distinct particle in a particle method simulation of Lennard-Jones molecular dynamics. They interact pairwise to compute the resultant sum of all accelerations of each atom and then evolve their position and velocity using velocity-Verlet time stepping. For simplicity, we again consider a one-dimensional domain. This defines the following particle method in our framework:

$$p := (x, v, a) \qquad \text{for } p \in P := \mathbb{R}^3, \tag{4.67}$$

$$g := (r_c, D, \Delta t, t_{end}, t) \qquad \text{for } g \in \mathbb{R}^5, \tag{4.68}$$

$$u\left([g, \mathbf{p}], j\right) := (k \in (1, ..., |\mathbf{p}|) :\ p_k, p_j \in \mathbf{p}\ \wedge\ 0 < d_D(x_j, x_k) \leq r_c), \tag{4.69}$$

$$f(g) := (t \geq t_{end})\,, \tag{4.70}$$

$$i\left(g, p_j, p_k\right) := \begin{pmatrix} \begin{pmatrix} x_j \\ v_j \\ a_j + \frac{24}{d_D(x_j,x_k)^7} - \frac{48}{d_D(x_j,x_k)^{13}} \end{pmatrix}^{\mathbf{T}} \\ \begin{pmatrix} x_k \\ v_k \\ a_k + \frac{24}{d_D(x_k,x_j)^7} - \frac{48}{d_D(x_k,x_j)^{13}} \end{pmatrix}^{\mathbf{T}} \end{pmatrix}^{\mathbf{T}}, \tag{4.71}$$

$$e\left(g, p_j\right) := \left(g, \left(\begin{pmatrix} (x_j + \Delta t\ \ (v_j + \Delta t \cdot a_j)) \bmod\ D \\ v_j + \Delta t \cdot a_j \\ 0 \end{pmatrix}^{\mathbf{T}}\right)\right), \tag{4.72}$$

$$\mathring{e}\left(g\right) := (r_c, D, \Delta t, t_{end}, t + \Delta t)\,. \tag{4.73}$$

In this example, a particle p is the collection of properties of one atom, namely its position x, velocity v, acceleration a, and previous acceleration a^{old}, as required for the velocity-Verlet method. All properties are real numbers. The global variable g is also a collection of real numbers: the cut-off radius r_c for the interactions, the time-step size Δt, the end time of the simulation t_{end}, and the current time t.

The neighborhood function u returns the surrounding particles, which are no further away than the cut-off radius r_c, are not the particle itself and are to the right of the particle (absence of absolute value in the distance computation). This hence defines an asymmetric neighborhood, which is used for symmetric interactions. The stopping condition f is true ($\top$) if the current time t reaches or exceeds the end time t_{end}.

The interact function i sums up all forces acting on a particle. Because the neighborhood is asymmetric, the interactions are symmetric, changing both involved particles. In

each pairwise interaction of two particles p_j and p_k, these particles add the contribution of the Lennard-Jones acceleration into their current acceleration. The evolve function e computes for a particle p_j the new position x_j and the new velocity v_j from the current acceleration a_j (calculated in i) and the previous acceleration a_j^{old} (stored). It also uses the time step size Δt from the global variable g. In the velocity-Verlet method, the position x is calculated before the acceleration a. Since the acceleration a is calculated in i, the position x needs to be calculated afterward. Hence, x for the next time step is calculated in e using $a(t_{n+1})$ and $v(t_{n+1})$. The evolve function also overwrites a_j^{old} with a_j and resets a_j to 0 for the next time step. In this example, it does not change the global variable g. That is only done in $\mathring{e}$, which advances the current time t by adding the time step size Δt.

An instance is again defined by fixing the parameters and instance of the particle method. In this example, we choose to initially place ten particles with a linearly increasing spacing between them and initialize all other properties to 0, hence:

$g^1 := (r_c, D, \Delta t, t_{end}, t)$	initial global variable
$r_c := 3$	cut-off radius
$D := 19$	domain size
$\Delta t := 0.0001$	time step size
$t_{end} := 10$	end time of the simulation
$t := 0$	current time
$\mathbf{p}^1 := (p_1, ..., p_{10})$	initial particle tuple
$p_j := (x_j, v_j, a_j)$	j-th particle
$x_j := j(0.9 + 0.11j)$	initial particle positions
$v_j := 0$	initial particle velocities
$a_j := 0$	initial particle accelerations

The result of executing this instance is shown in figure 4.4 at the initial and final time. As a validation, we show in figure 4.5 that the system's total (i.e., kinetic plus potential) energy is behaving as expected for a symplectic time integrator like the velocity-Verlet scheme used here. Due to numerical round-off errors, the global energy difference fluctuates around zero with errors comparable to the square of machine epsilon for double-precision floating-point arithmetics, as expected from the second-order time integration method used.

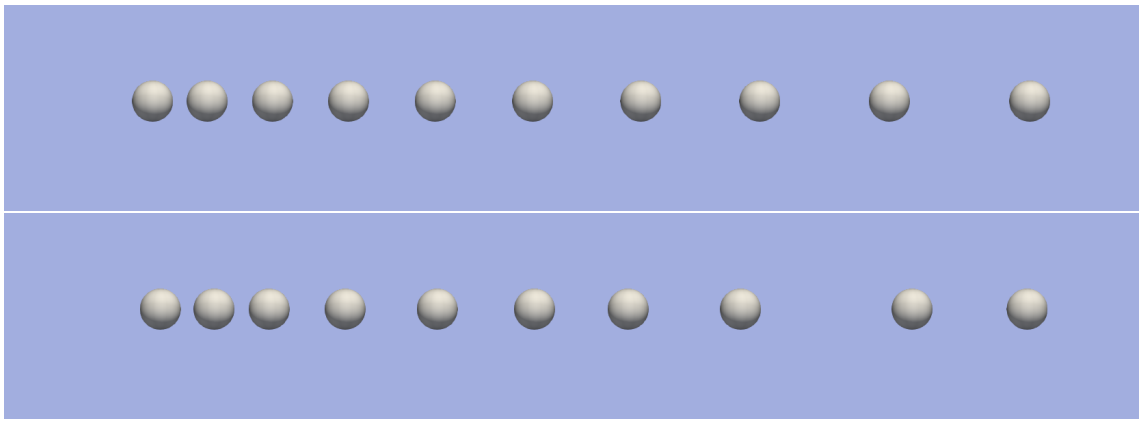


Figure 4.4: Visualization of a molecular dynamics simulation of 10 Lennard-Jones atoms in one dimension at times $t = 0$ (top) and $t = t_{end} = 10$ (bottom) (visualized using ParaView [85]).

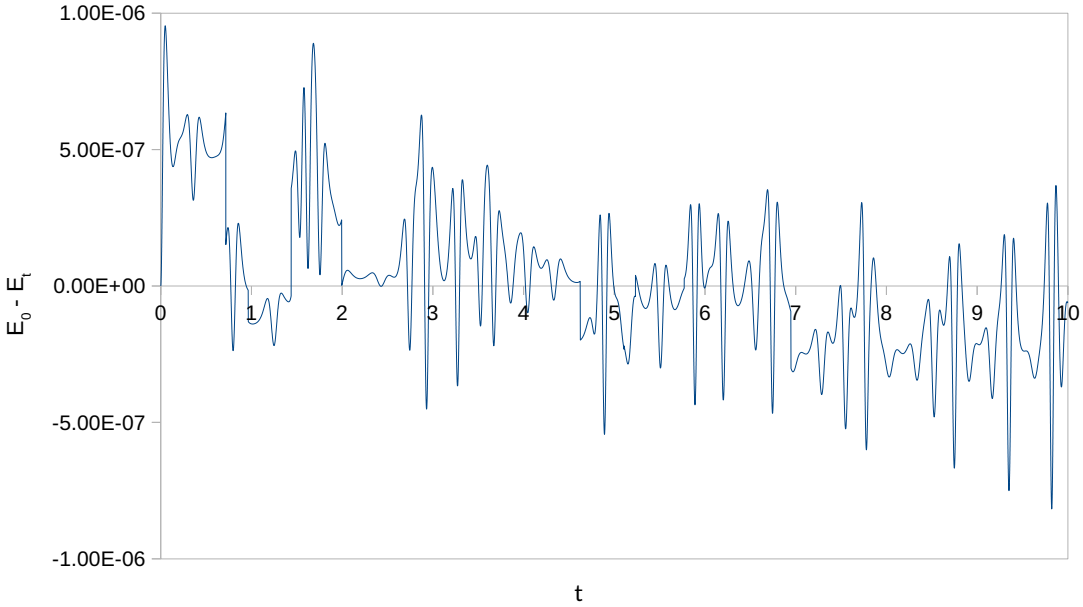


Figure 4.5: Deviation of the total energy of the molecular dynamics simulation from the total initial energy at $t = 0$.

4.6 Triangulation refinement

After seeing classic particle methods, we also show three examples of how we can formulate algorithms not generally recognized as particle methods in our definition. This does not imply one should implement them as practice methods but to show the generality of our definition and the value of having a formal framework.

The first example considers an algorithm for triangulation refinement as often used

in computer graphics [28]. It refines a triangulation by replacing each triangle with four smaller triangles. It creates three new vertices for each existing triangle, one at the midpoint of each edge. Three new edges connect then these vertices. Together with the existing vertices and edges, this creates a refined triangulation. We illustrate the process in figure 4.6.

There are multiple ways of formulating this process as a particle method. An obvious choice might be to represent each vertex of the triangular mesh by a particle and to store the incoming or outgoing edges as (integer-valued) properties of the particles. Another choice we follow here for illustration is to represent each triangle by a particle. Each particle (triangle) then stores a unique index, a vector of three vertices, the indices of its face-connected neighbors, and a "reverse index" storing the information which neighbor (0th, 1st, or 2nd) it is from the perspective of its neighbor particles.

Through the interact function i, this neighbor information is exchanged. The evolve function e is then used to calculate the neighbor indices of the new, smaller particles (triangles). The old particles (triangles) are deleted by the evolve function e, and four new particles (triangles) are created. We choose this representation because it illustrates a case where the evolve function creates and destroys particles. Assuming that everything happens in a two-dimensional space, the resulting particle method reads:

$$\begin{aligned} p &:= \left(\iota, \begin{pmatrix} {}^0\underline{v} \\ {}^1\underline{v} \\ {}^2\underline{v} \end{pmatrix}, \begin{pmatrix} {}^0\beta \\ {}^1\beta \\ {}^2\beta \end{pmatrix}, \begin{pmatrix} {}^0\gamma \\ {}^1\gamma \\ {}^2\gamma \end{pmatrix}\right) \\ &\quad \text{for } p \in P := \mathbb{N}_0 \times \left(\mathbb{R}^2\right)^3 \times \left(\mathbb{N}_0 \cup \{-1\}\right)^3 \times \{0,1,2\}^3, \end{aligned} \tag{4.74}$$

$$g := (t_{end}, t) \qquad \text{for } g \in G := \mathbb{N}_0 \times \mathbb{N}_0,$$

$$u\left([g, \mathbf{p}], j\right) := \left({}^0\beta_j, {}^1\beta_j, {}^2\beta_j\right), \tag{4.75}$$

$$f(g) := (t \geq t_{end}), \tag{4.76}$$

$$i(g, p_j, p_k) := \tag{4.77}$$

$$\left(\left(\iota_j, \begin{pmatrix} {}^0\underline{v}_j \\ {}^1\underline{v}_j \\ {}^2\underline{v}_j \end{pmatrix}, \begin{pmatrix} {}^0\beta_j \\ {}^1\beta_j \\ {}^2\beta_j \end{pmatrix}, \begin{cases} \begin{pmatrix} r \\ {}^1\gamma_j \\ {}^2\gamma_j \end{pmatrix} & \text{with } \iota_j = {}^r\beta_k \quad \text{if } \iota_k = {}^0\beta_j \\ \begin{pmatrix} {}^0\gamma_j \\ r \\ {}^2\gamma_j \end{pmatrix} & \text{with } \iota_j = {}^r\beta_k \quad \text{if } \iota_k = {}^1\beta_j \\ \begin{pmatrix} {}^0\gamma_j \\ {}^1\gamma_j \\ r \end{pmatrix} & \text{with } \iota_j = {}^r\beta_k \quad \text{if } \iota_k = {}^2\beta_j \end{cases}, p_k\right)\right),$$

$$
\begin{aligned}
&e(g, p_j) := \\
&\left(g, \left(\begin{array}{c}
\left(4\,\iota_j, \begin{pmatrix} {}^0\underline{v}_j \\ \frac{{}^0\underline{v}_j + {}^1\underline{v}_j}{2} \\ \frac{{}^0\underline{v}_j + {}^2\underline{v}_j}{2} \end{pmatrix}, \begin{pmatrix} \begin{cases} 4\,{}^0\beta_j + ({}^0\gamma + 1) \bmod 3 & \text{if} \quad {}^0\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \\ 4\iota_j + 3 \\ \begin{cases} 4\,{}^2\beta_j + {}^2\gamma & \text{if} \quad {}^2\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\right) \\
\left(4\,\iota_j + 1, \begin{pmatrix} {}^1\underline{v}_j \\ \frac{{}^1\underline{v}_j + {}^2\underline{v}_j}{2} \\ \frac{{}^0\underline{v}_j + {}^1\underline{v}_j}{2} \end{pmatrix}, \begin{pmatrix} \begin{cases} 4\,{}^1\beta_j + ({}^1\gamma + 1) \bmod 3 & \text{if} \quad {}^1\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \\ 4\iota_j + 3 \\ \begin{cases} 4\,{}^0\beta_j + {}^0\gamma & \text{if} \quad {}^0\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\right) \\
\left(4\,\iota_j + 2, \begin{pmatrix} {}^2\underline{v}_j \\ \frac{{}^0\underline{v}_j + {}^2\underline{v}_j}{2} \\ \frac{{}^1\underline{v}_j + {}^2\underline{v}_j}{2} \end{pmatrix}, \begin{pmatrix} \begin{cases} 4\,{}^2\beta_j + ({}^2\gamma + 1) \bmod 3 & \text{if} \quad {}^2\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \\ 4\iota_j + 3 \\ \begin{cases} 4\,{}^1\beta_j + {}^1\gamma & \text{if} \quad {}^1\beta_j \neq -1 \\ -1 & \text{else} \end{cases} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\right) \\
\left(4\,\iota_j + 3, \begin{pmatrix} \frac{{}^0\underline{v}_j + {}^1\underline{v}_j}{2} \\ \frac{{}^1\underline{v}_j + {}^2\underline{v}_j}{2} \\ \frac{{}^0\underline{v}_j + {}^2\underline{v}_j}{2} \end{pmatrix}, \begin{pmatrix} 4\,\iota_j + 1 \\ 4\,\iota_j + 2 \\ 4\,\iota_j \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\right)
\end{array}\right)^{\mathbf{T}}\right), \qquad (4.78)
\end{aligned}
$$

$$
\mathring{e}(g) := (t_{end}, t + 1)\,. \qquad (4.79)
$$

In this example, a particle p is the collection of properties of one triangle: the index ι, the vertices $\left({}^0\underline{v}, {}^1\underline{v}, {}^2\underline{v}\right)^T$, the indices of the neighbor particles (triangles) $\left({}^0\beta, {}^1\beta, {}^2\beta\right)^T$, and the storage for the information which neighbor (0th, 1st, or 2nd) the particle is from the perspective of its neighbor particles $\left({}^0\gamma, {}^1\gamma, {}^2\gamma\right)^T$. The index is a natural number. Each vertex is a real 2D vector storing the vertex position in a 2D plane. The neighbor indices are natural numbers but include -1 to indicate if there is no neighbor. The storage order is 0, 1, 2. The global variable g is a collection of two natural numbers, including zero: the number of refinement steps t_{end} to be made, and the current refinement step t.

The neighborhood function u returns the indices of the neighbor particles (triangles) $\left({}^0\beta, {}^1\beta, {}^2\beta\right)^T$. The stopping condition f is true ($\top$) if the current refinement step t reaches the number of refinement steps t_{end}.

The interact function i exchanges the information which neighbor (0th, 1st, or 2nd) the particle p_j is from the perspective of particle p_k and saves this information in the γ_j corresponding to the position of the index i_k of the particle p_k in β_j. The evolve function e creates the new particles and uses the information stored in γ_j to determine the indices of the new particles. The old particle no longer exists in the following state (refined triangulation). Instead, a tuple of new particles is created. In this evolve function, the global variable g remains unchanged. The evolve method of the global variable $\mathring{e}$ increments the

refinement step t by 1.

An instance of this particle method has to specify the initial set of triangles (particles) and the number of refinement steps. We recapitulate the situation in figure 4.6, performing one refinement step of a single triangle into four smaller triangles. The method therefore starts from one particle with index 0 and vertices $((0,0),(4,0),(2,4))^T$. The neighbor indices are all set to ${}^r\beta_1 = -1$ $(r = 0,1,2)$ to indicate that there are no neighbors yet. The reverse indices are arbitrarily set to ${}^r\gamma_1 = 0$ $(r = 0,1,2)$, but this has no meaning because there are no neighbors yet. This leads to the instance:

$$
\begin{aligned}
g^1 &:= (t, t_{end}) && \text{initial global variable} \\
t_{end} &:= 1 && \text{number of refinement steps} \\
t &:= 0 && \text{current refinement step} \\
\\
\mathbf{p}^1 &:= (p_1^1) && \text{initial particle} \\
p_1^1 &= \left(\iota_1, \begin{pmatrix} {}^0\underline{v}_1 \\ {}^1\underline{v}_1 \\ {}^2\underline{v}_1 \end{pmatrix}, \begin{pmatrix} {}^0\beta_1 \\ {}^1\beta_1 \\ {}^2\beta_1 \end{pmatrix}, \begin{pmatrix} {}^0\gamma_1 \\ {}^1\gamma_1 \\ {}^2\gamma_1 \end{pmatrix}\right) && \text{initial triangle} \\
&:= \left(0, \begin{pmatrix} (0,0) \\ (4,0) \\ (2,4) \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\right)
\end{aligned}
$$

The result of executing this is shown in figure 4.6.

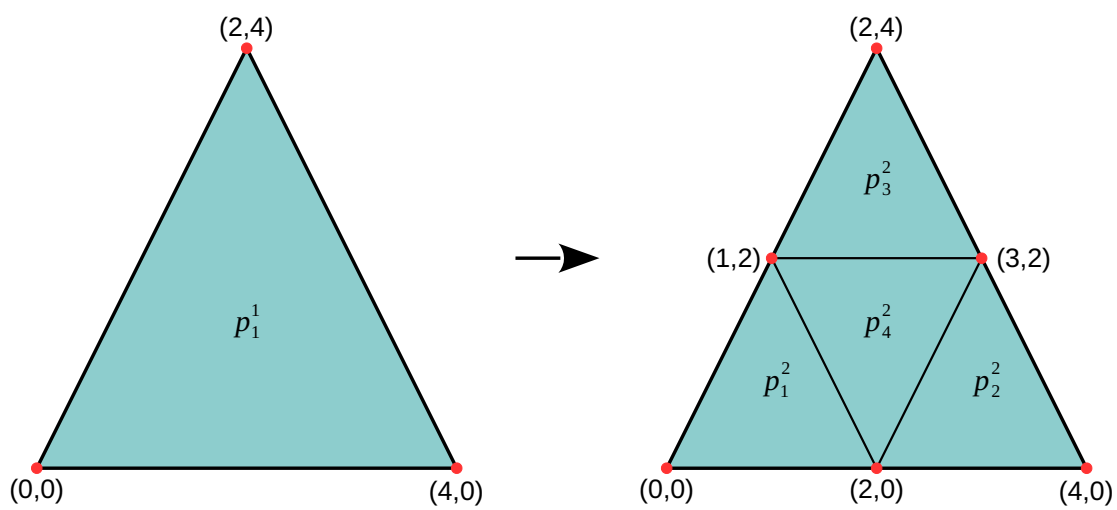


Figure 4.6: Visualization of the *refinement* of a single triangle $\mathbf{p}^1$ to four triangles by inserting three new vertices.

4.7 Conway's Game of Life

As a second example of a non-canonical particle method, we formulate John H. Conway's solitary game "life" [32] using our definition. The game "life" is based on a two-dimensional cellular automaton. Cellular automata [64, 65] can be interpreted as abstract parallel processing computers [59, 70], where all cells change their state simultaneously in each step. A cell's change is only influenced by its own state and the state of its neighbors. One-dimensional cellular automata became famous for the complex patterns they are able to generate, despite their extremely simple mechanics. Thus, they build the starting point for the "computational universe" of Stephen Wolfram [91].

In Conway's game of life, the cells are, in theory, on a two-dimensional infinite lattice. Each cell can either be "dead" or "alive". To determine if a cell is "dead" or "alive" in the next iteration, the Moore neighborhood (all eight adjacent cells) of each cell and the following three rules are considered.

1. An alive cell dies if four or more neighbors are alive or if just one or no neighbor is alive.
2. An alive cell stays alive if two or three neighbors are alive.
3. A dead cell gets alive if exactly three neighbors are alive. In all other cases, a dead cell stays dead.

We translate this into a particle method by identifying each cell as a particle. Then we sum up all alive neighbors and transform each cell according to the rules. The particle method reads as follows.

$$p := (\mathbf{l}, a, n) \ \in P := \mathbb{N}^2 \times \{0,1\} \times \{0,1,...,8\}, \tag{4.80}$$

$$g := (t_{end}, t) \in G := \mathbb{N} \times \mathbb{N}, \tag{4.81}$$

$$u\left([g, \mathbf{p}], j\right) := \left(k \in (1,...,|\mathbf{p}|) : |\mathbf{l}_k - \mathbf{l}_j|_\infty \leq 1\right), \tag{4.82}$$

$$f(g) := \left(t > t_{end}\right), \tag{4.83}$$

$$i(g, p_j, p_k) := \begin{cases} ((\mathbf{l}, a, n+1), p_k) & \text{if } a_k = 1 \\ (p_j, p_k) & \text{else} \end{cases}, \tag{4.84}$$

$$e(g, p_j, p_k) := \begin{cases} (g, ((\mathbf{l}, 1, 0))) & \text{if } n_j = 3 \vee (n_j = 2 \wedge a_j + 1) \\ (g, ((\mathbf{l}, 0, 0))) & \text{else} \end{cases}, \tag{4.85}$$

$$\mathring{e}(g) := (t_{end}, t+1) \tag{4.86}$$

In this example, a particle consists of the two-dimensional index $\mathbf{l}$, the aliveness flag a, and the counter for the alive neighbors n. The global variable consists of the number of

steps t_{end} and the step counter t. Hence, the stop function f is true if the counter exceeds the number of steps. The neighborhood function u returns the indices of the particle that are in the Moore neighborhood of a particle. The interact function i counts the alive neighbors by increasing the counter n by one for each interaction with an alive particle. The particles are set to their new aliveness in the evolve function e, where also the counter is set back to zero. At last, the evolve function of the global variable increases the step counter t by one.

The instance of this particle method realizes the Gosper glider gun [4] on a lattice of 50 times 35 cells for 100 iterations. This leads to the instance with a global variable of

$$g^1 := (t, t_{end}) \tag{4.87}$$

$$t_{end} := 100 \qquad \text{number of iteration} \tag{4.88}$$

$$t := 1 \qquad \text{current iteration,} \tag{4.89}$$

and the initial particle tuple of

$$\mathbf{p}^1 := \left(p_1^1, ..., p_{1850}^1\right) \qquad \text{initial particles} \tag{4.90}$$

$$p_1^1 = \left(\underbrace{\begin{pmatrix}0\\0\end{pmatrix}}_{\mathbf{l}_1^1}, a_1^1, \underbrace{0}_{n_1^1}\right) \tag{4.91}$$

$$p_2^1 = \left(\begin{pmatrix}1\\0\end{pmatrix}, a_2^1, 0\right) \tag{4.92}$$

$$\vdots$$

$$p_{50}^1 = \left(\begin{pmatrix}49\\0\end{pmatrix}, a_{50}^1, 0\right) \tag{4.93}$$

$$p_{51}^1 = \left(\begin{pmatrix}0\\1\end{pmatrix}, a_{51}^1, 0\right) \tag{4.94}$$

$$\vdots$$

$$p_{1850}^1 = \left(\begin{pmatrix}49\\34\end{pmatrix}, a_{1850}^1, 0\right). \tag{4.95}$$

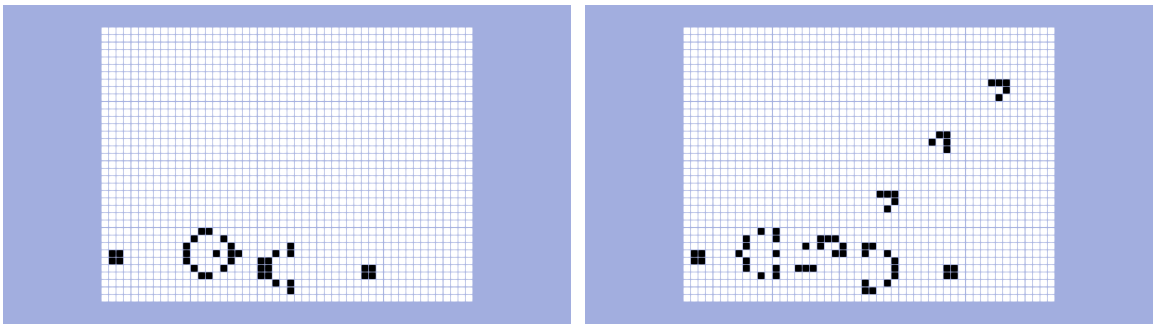
The alive cells are the ones with an index $\mathbf{l} \in L$ with

$$\begin{aligned} L := \Bigg\{ &\begin{pmatrix}1\\5\end{pmatrix}, \begin{pmatrix}1\\6\end{pmatrix}, \begin{pmatrix}2\\5\end{pmatrix}, \begin{pmatrix}2\\6\end{pmatrix}, \begin{pmatrix}11\\5\end{pmatrix}, \begin{pmatrix}11\\6\end{pmatrix}, \begin{pmatrix}11\\7\end{pmatrix}, \begin{pmatrix}12\\4\end{pmatrix}, \begin{pmatrix}12\\8\end{pmatrix}, \begin{pmatrix}13\\3\end{pmatrix}, \begin{pmatrix}13\\9\end{pmatrix}, \\ &\begin{pmatrix}14\\3\end{pmatrix}, \begin{pmatrix}14\\9\end{pmatrix}, \begin{pmatrix}15\\6\end{pmatrix}, \begin{pmatrix}16\\4\end{pmatrix}, \begin{pmatrix}16\\8\end{pmatrix}, \begin{pmatrix}17\\5\end{pmatrix}, \begin{pmatrix}17\\6\end{pmatrix}, \begin{pmatrix}17\\7\end{pmatrix}, \begin{pmatrix}18\\6\end{pmatrix}, \begin{pmatrix}21\\3\end{pmatrix}, \\ &\begin{pmatrix}21\\4\end{pmatrix}, \begin{pmatrix}21\\5\end{pmatrix}, \begin{pmatrix}22\\3\end{pmatrix}, \begin{pmatrix}22\\4\end{pmatrix}, \begin{pmatrix}22\\5\end{pmatrix}, \begin{pmatrix}23\\2\end{pmatrix}, \begin{pmatrix}23\\6\end{pmatrix}, \begin{pmatrix}25\\1\end{pmatrix}, \begin{pmatrix}25\\2\end{pmatrix}, \begin{pmatrix}25\\6\end{pmatrix}, \\ &\begin{pmatrix}25\\7\end{pmatrix}, \begin{pmatrix}35\\3\end{pmatrix}, \begin{pmatrix}35\\4\end{pmatrix}, \begin{pmatrix}36\\3\end{pmatrix}, \begin{pmatrix}36\\4\end{pmatrix} \Bigg\}. \end{aligned} \tag{4.96}$$

Hence,

$$a_j^1 = \begin{cases} 1 & \text{if } \mathbf{l}_j^1 \in L \\ 0 & \text{else} \end{cases}. \tag{4.97}$$

The result of executing this is shown in figure 4.7.



(a) Initial state of the cell lattice at $t = 1$. (b) Final state of the cell lattice at $t = 100$.

Figure 4.7: Simulation of the Gosper glider gun (visualized using ParaView [85]).

4.8 Gaussian Elimination

As a third example of a non-canonical particle method, we formulate the classic Gaussian elimination algorithm in the framework of our definition. Gaussian elimination is a classic algorithm to invert a matrix or solve a linear system of equations. It requires a cubic number of computation operations with the linear dimension of the matrix.

In our example, we use Gaussian elimination to solve systems of N linear equations with N unknowns. Each row of the resulting $N \times N$ matrix represents one equation's coefficients. The right-hand side values are concatenated to the right of the matrix, as shown below.

Gaussian elimination uses three operations: swapping two rows, multiplying all entries in a row with a non-zero number, and adding one row to another. These operations are combined in the following steps to reduce the matrix to a reduced-row echelon form, i.e., an upper triangular matrix where the first non-zero entry in each row is 1, and no other entry in any leading-1 column is non-zero:

1. Start with the entry in the first row and the first column.

2. If the entry is 0, find the first row with a non-zero entry in the same column and swap the rows. If there is no non-zero entry, proceed to the next column but stay in the same row and start this step over. If the entry is non-zero, make all other entries below it in the same column zero by adding corresponding multiples of the row to the other rows. Finally, multiply the row with a non-zero number such that its leading element becomes 1.

3. Proceed to the next row and column and repeat step 2. This results in an upper triangular matrix with leading ones once they arrive at the last column.

4. Finally, start from the bottom row and add multiples of the current row to the other rows to get 0 entries in all columns containing a leading 1. This scheme results in the reduced row echelon form of the linear equation system.

In our particle method implementation of this algorithm, a particle represents a row (equation) of the linear equation system. Interactions between particles (rows) change the entries in the rows, such as to create the upper triangular matrix and then the reduced row echelon form. The evolve function normalizes the rows to have leading ones. In the language of our formal definition, this can be expressed as:

$$g := (N, m, n) \qquad \text{for } g \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}, \tag{4.98}$$

$$p := \left(\left({}^{l}a \right)_{l=1}^{N}, b, \mu \right) \qquad \text{for } p \in P := \mathbb{R}^N \times \mathbb{R} \times \mathbb{R}, \tag{4.99}$$

$$u\left([g, \mathbf{p}], j\right) := \begin{cases} (N, ..., n+1) & \text{if} \quad j = n,\ m \leq N \\ (1, ..., n-1) & \text{if} \quad j = n,\ m > N \\ () & \text{else,} \end{cases} \tag{4.100}$$

$$f\left(g\right) := \left(n = 1 \wedge m > N\right), \tag{4.101}$$

$$i(g, p_j, p_k) := \begin{cases} \left(p_j, \left(\begin{pmatrix} \left({}^{l}a_k - {}^{l}a_j \frac{{}^{m}a_k}{{}^{m}a_j} \right)_{l=m}^{N} \\ b_k - b_j \frac{{}^{m}a_k}{{}^{m}a_j} \\ \mu_k \end{pmatrix}^{\mathbf{T}} \right) \right) & \text{if} \quad m \leq N,\ {}^{m}a_j \neq 0 \\ (p_k, p_j) & \text{if} \quad m \leq N,\ {}^{m}a_j = 0,\ {}^{m}a_k \neq 0 \\ (p_j, p_k) & \text{if} \quad m \leq N,\ {}^{m}a_j = 0,\ {}^{m}a_k = 0 \\ \left(p_j, \left(\begin{pmatrix} \left({}^{l}a_k - {}^{\mu_j}a_k\ {}^{l}a_j \right)_{l=\mu_j}^{N} \\ b_k - {}^{\mu_j}a_k\ b_j \\ \mu_k \end{pmatrix}^{\mathbf{T}} \right) \right) & \text{else,} \end{cases} \tag{4.102}$$

$$e(g, p_j) := \left(\left(N, m, \begin{cases} n+1 & \text{if} \quad j = n,\ m < N,\ {}^{m}a_j \neq 0 \\ n-1 & \text{if} \quad j = n,\ m = N,\ {}^{m}a_j = 0 \\ n & \text{else} \end{cases} \right), \left(\begin{cases} \left(\left(\frac{{}^{l}a_j}{{}^{m}a_j} \right)_{l=m}^{N}, \frac{b_j}{{}^{m}a_j}, m \right) & \text{if} \quad j = n,\ m \leq N,\ {}^{m}a_j \neq 0 \\ p_j & \text{else} \end{cases} \right) \right), \tag{4.103}$$

$$\mathring{e}(g) := \left(N, m+1, \begin{cases} n-1 & \text{if} \quad m > N \\ n & \text{else} \end{cases} \right). \tag{4.104}$$

Here, the global variable g is a collection of three natural numbers: the number of rows and columns N of the square matrix, the index of the current column m ($m > N$ indicates

that the algorithm is in step 4), and the index of the current row n. Each particle stores the matrix entries in the row represented by the particle in a vector $\left({}^{l}a\right)_{l=1}^{N}$ with b the corresponding right-hand side of the linear equation system. For simplicity, the column index of the leading 1 is saved in a separate property μ.

The neighborhood function u ensures that only the current row ($j = n$) has neighbors. Therefore, only this one row will participate in the pivoting process. As long as $m \leq N$, the neighbor particles are the rows below in reverse order. The reverse order ensures that if there is a zero row, it will be sorted to the end by the interaction function i. For step 4 of the algorithm ($m > N$), the order of the neighbor particles is irrelevant as the neighbor particles are then all the rows above. The stopping condition f is true ($\top$) if the current row is the top row again ($n = 1$) and the algorithm is in step 4 ($m > N$).

The first three cases in the interact function i are for steps 1 to 3 of the algorithm, the last case for step 4. The first case is the addition of a suitable multiple of the current row to a neighboring row to create zeros in the m-th column. The second case handles the swapping of two rows if the m-th entry of the current row p_j is zero but that of the neighbor row p_k is non-zero. If the m-th entry of both interacting rows, p_j and p_k, is zero, nothing happens (third case). The fourth case creates zeros in the μ_j-th (leading 1 entry) column of the interacting rows p_k.

The evolve function e is only active as long as $m \leq N$ and only if the evolving particle is the current row of $j = n$. Otherwise, it amounts to the identity map. In the non-trivial case, it normalizes the leading entry ${}^{m}a_j$ of the current particle (row) to 1 by dividing all non-zero coefficients with ${}^{m}a_j$. It also saves the index of the leading 1, m, in the particle property μ_j. The global variable g is changed, incrementing the index of the current row n if the current column is not the last one ($m < N$) and if the m-th coefficient of the row is not zero (${}^{m}a_j \neq 0$). The turning point of the algorithm, where it switches from iterating downward through the rows to iterating back up again, is when $m = N$. At this point, the index of the current row n remains unchanged unless the last entry is zero (${}^{m}a_j = 0$) because then the row is completely zero and hence has no leading one. In this case, the algorithm can already go to the row above ($n - 1$).

The evolve function of the global variable $\mathring{e}$ increments the current column m by one so that the algorithm proceeds one column to the right in each iteration. During step 4, m increments beyond N, which is irrelevant. Then, however, n is decremented by one in each iteration to move up one row at a time.

An instance of this particle method is defined by specifying the linear system's coefficients as the initial set of particles and the size of the system. The property μ can be arbitrarily initialized, as it is overwritten before being used. As an example, we consider the instance for the 3×3 equation system:

$$\begin{bmatrix} 1 & 2 & 5 \\ 1 & -1 & -4 \\ 2 & 6 & 16 \end{bmatrix} \vec{x} = \begin{bmatrix} 2 \\ -4 \\ 8 \end{bmatrix}$$

with unknown variable $\vec{x}$. For this example, $N = 3$, and the particle method instance is:

$g := (N, m, n)$	initial global variable
$N := 3$	number of rows and columns
$m := 1$	start from the first column
$n := 1$	start from the first row
$\mathbf{p}^1 := (p_1, p_2, p_3)$	initial particle tuple
$p := ((^0a,\ ^1a,\ ^2a), b, \mu)$	particle prototype
$p_1 := ((1,\ 2,\ 5),\ 2,\ 0)$	particle 1 (first row)
$p_2 := ((1, -1, -4), -4, 0)$	particle 2 (second row)
$p_3 := ((2,\ 6,\ 16),\ 8,\ 0)$	particle 3 (third row)

Executing the algorithm for this instance produces four iterations:

$$\left[\begin{array}{rrr|r} 1 & 2 & 5 & 2 \\ 1 & -1 & -4 & -4 \\ 2 & 6 & 16 & 8 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 2 & 5 & 2 \\ 0 & -3 & -9 & -6 \\ 0 & 2 & 6 & 4 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 2 & 5 & 2 \\ 0 & -3 & -9 & -6 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

$$\to \left[\begin{array}{rrr|r} 1 & 2 & 5 & 2 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 0 & -1 & -2 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 \end{array}\right].$$

4.9 Conclusion

Our definition of particle methods is designed for a broad range of algorithms. Even though our definition of particle methods is derived from the formulation and practical implementation of various classic particle methods, its applicability to such algorithms needs to be shown.

The presented definition unifies particle methods and allows the formulation of particle methods for non-canonical algorithms. We showcased that by formulating classic particle methods and algorithms generally not recognized as particle methods. We chose SPH, PSE, MD, and DEM as examples of classic particle methods. As examples of non-canonical algorithms, we considered triangulation refinement, Conway's game of life, and Gaussian elimination.

Even though the presented definition is general and easily applicable to, e.g., SPH, MD, and PSE, an algorithm formulated according to our particle method definition could potentially have a worse time and space complexity than a non-particle algorithm, especially for non-canonical problems. Additionally, not all algorithms are naturally fitting into the definition. For example, Gaussian elimination turned out to be challenging to translate into our definition and is less efficient since most particles are inactive throughout the interaction and evolution phase even though the time complexity is the same ($\mathcal{O}(n^3)$ for n being the number of unknowns) since the difference is only in the prefactor. The triangulation refinement fits well into the structure, but its formulation was also not straightforward. Conway's game of life, on the other hand, fits perfectly into our definition. Further, our definition is limited by its monolithic nature. An algorithm composed of smaller algorithms could become very large and complex with several nested cases when explicitly formulated in our definition. This tendency is already visible in the SPH example. Notwithstanding these limitations, the present applications cover a broad range of algorithms.

Future work could expand the application of the definition into further non-canonical fields to clarify the practical abilities and limitations of the presented definition. Also, multiple different formulations of the same classic particle methods in our definition can help to identify advantageous properties of specific formulation strategies.

Overall, the presented applications show the unifying nature of our formal particle methods definition and its applicability which is not limited to classic particle methods.

Chapter 5

Parallelizability of Particle Methods

5.1 Introduction

High-performance computing (HPC) is becoming increasingly important in research, especially in the life sciences [88, 49], 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 parallelization. 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 to bridge the gap between accessible programming and heterogeneous-hardware parallelization [76]. At the foundation of these frameworks are generic and parallelizable numerical algorithms such as particle methods. Even though the frameworks are generic, the conditions when an algorithm is parallelizable are derived application-wise.

Our mathematical definition of particle methods (chapter 3) enables their formal study across applications. Leveraging this mathematical definition, it is possible to prove the parallelizability of particle methods on shared- and distributed-memory systems under certain conditions independent of applications.

Here, we provide the conditions and proofs of the equivalents of the parallelized particle methods to their sequential counterparts, i.e., in the sense that they compute the same results for any possible input. Our study is independent of a specific application or a specific numerical method. The proof covers a broad class of particle-based algorithms. Moreover, the parallelization schemes we use and analyze are well-known and commonly used in practical implementations.

The considered parallelization scheme for shared-memory systems is based on one-sided interactions, and the scheme for distributed-memory systems is based on the classic cell-list algorithm [41] and a checkerboard-like domain decomposition. We formalize the schemes in mathematical equations as well as Nassi-Shneiderman diagrams. We then provide for each scheme an exhaustive list of conditions a particle method must fulfill for the schemes to be correct. Under these conditions, we prove the equivalence of the presented parallelization schemes to the underlying sequential particle method. Furthermore, we use the presented

formal analysis to infer the scheme's time complexity and parallel scalability bounds.

5.2 Particle Methods on Shared Memory Systems

We consider the parallelization of particle methods on shared-memory computers. For the parallelization on shared-memory systems, we assume that parallel reading from one storage location is possible but not parallel writing.

5.2.1 Parallelization Scheme

Conditions

We need to ensure that the calculations can be done in parallel and that we do not have race conditions or other conflicts while writing. Therefore, we need the five conditions: condition one, pull-interaction

$$i(g, p_j, p_k) = (\overline{p}_j, p_k), \tag{5.1}$$

where the first particle p_j is changed while the second p_k stays the same.

Condition two, interaction independence of previous interactions

$${}_1 i_g(p_j, {}_1 i_g(p_k, p_{k'})) = {}_1 i_g(p_j, p_k) \quad \text{for} \quad {}_1 i_g(p_j, p_k) := \langle i(g, p_j, p_k) \rangle_1, \tag{5.2}$$

condition three, neighborhood independence of previous interactions

$$u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j), \tag{5.3}$$

condition four, constant number of particles

$$e(g, p) = (\overline{g}, (\overline{p})), \tag{5.4}$$

condition five, global variable independence of particles

$$e(g, p) = (g, \mathbf{q}). \tag{5.5}$$

Scheme as Nassi-Shneiderman diagram

These conditions are used to construct a parallelization scheme for particle methods on shared-memory systems. The Nassi-Shneiderman diagram provides a visual overview of how the state transition function S (3.16) is parallelized.

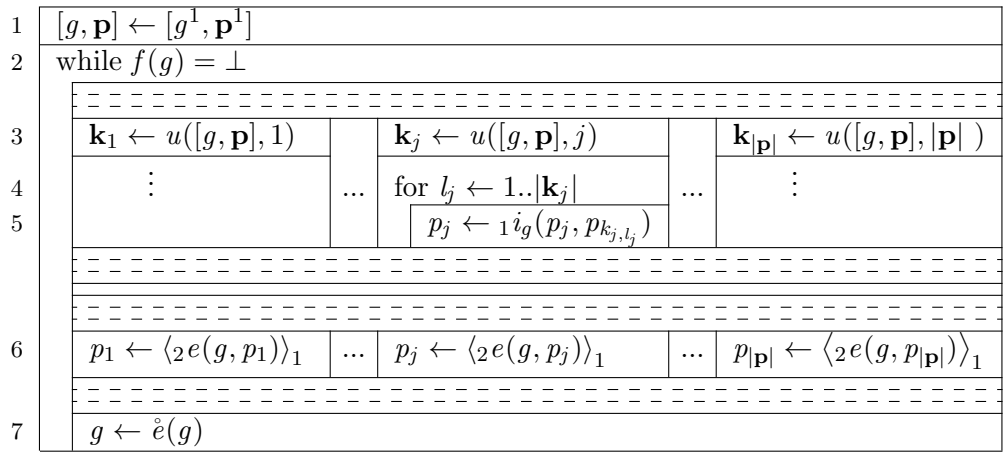


Figure 5.1: Nassi-Shneiderman diagram of the parallelized state transition function for shared memory systems. The dashed lines enclose the parallel part.

The scheme proceeds like the standard state transition function (fig. 3.1) except for the two loops over all particles for the iteration and evolution. These loops are replaced by parallel execution.

Scheme in Formulas

The parallel sections of the Nassi-Shneiderman diagram (fig. 5.1), hence the difference to the sequential state transition (fig. 3.1) can be translated to formulas step by step. The entry $[\mathbf{k}_j \leftarrow u([g, \mathbf{p}], j)]$ (line 3) translates to

$$\mathbf{k}_j = u([g, \mathbf{p}], j), \tag{5.6}$$

and [for $l_j \leftarrow 1..|\mathbf{k}_j|$] together with $\left[p_j \leftarrow {}_1 i_g(p_j, p_{k_{j,l_j}})\right]$ (lines 4, 5) to

$$\overline{p}_j = p_j *_1 i_g \left(p_{\langle \mathbf{k}_j \rangle_1}, ..., p_{\langle \mathbf{k}_j \rangle_{|\mathbf{k}_j|}}\right). \tag{5.7}$$

Combining (5.6) with (5.7) for all particles results in the tuple

$$\left(p_1 *_1 i_g \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ ..., \ \ p_{|\mathbf{p}|} *_1 i_g \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right). \tag{5.8}$$

All particles are constantly overwritten at this interaction part of the scheme. Also the evolution part $[p_1 \leftarrow \langle {}_2 e(g, p_1) \rangle_1]$ (line 6) overwrites each particle. Hence, inside these parts are potential writing conflicts, but not between them because they are executed after each other. Therefore, we can take them together as follows

$$\left(\left\langle {}_2 e \left(p_1 *_1 i_g \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}\right)\right\rangle_1, ..., \left\langle {}_2 e \left(p_{|\mathbf{p}|} *_1 i_g \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)\right\rangle_1\right). \tag{5.9}$$

Adding to this, the evolution of the global variable function results in a parallelized step of the particle method state transition function.

$$\tilde{s}\left([g,\mathbf{p}]\right) = \left[\mathring{e}(g), \left(\left\langle {}_2e\left(p_1 *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},1)}\right)\right\rangle_1, \ ..., \ \left\langle {}_2e\left(p_{|\mathbf{p}|} *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)\right\rangle_1\right)\right]. \quad (5.10)$$

The rest of the state transition is identical to the sequential state transition. Hence, it is

$$\begin{aligned}\tilde{S}([g^1,\mathbf{p}^1]) := [g^T,\mathbf{p}^T] \qquad &\Longleftrightarrow \\ f(g^T) = \top \ \wedge \ \forall t \in \{2,...,T\}: \ [g^t,\mathbf{p}^t] = \tilde{s}\left([g^{t-1},\mathbf{p}^{t-1}]\right) \ \wedge \ f(g^{t-1}) = \bot. \quad (5.11)\end{aligned}$$

5.2.2 Lemmata

We need five intermediate results to prove the equivalence of the parallel state transition and the sequential one.

Lemma 1. *If the interaction is a pull interaction, all particle interactions with its neighbors do not change any particle besides the particle itself.*

$$\begin{aligned}&\forall p_j, p_k \in P, g \in G: \ i(g,p_j,p_k) = (\overline{p}_j, p_k) \\ &\qquad \Longrightarrow \quad \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,j)}} (k_1,...,k_n) \ = \left(p_1,..., \ p_j *_{1i_g} (p_{k_1},...,p_{k_n}), ..., p_{|\mathbf{p}|}\right), \quad (5.12)\end{aligned}$$

Proof.

$$\begin{aligned}&\forall p_j, p_k \in P, g \in G: \ i(g,p_j,p_k) = (\overline{p}_j, p_k) && (5.13)\\ \Longrightarrow \\ &\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,j)}} (k_1,...,k_n) && (5.14)\\ &= \left(p_1,...,{}_1i(g,p_j,p_{k_1}),...,p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} (k_2,...,k_n) && (5.15)\\ &= \left(p_1,...,{}_1i_g\left({}_1i_g(p_j,p_{k_1}),p_{k_2}\right),...,p_{|\mathbf{p}|}\right) \\ &\qquad *_{\iota^{\mathrm{I}}_{(g,j)}} (k_3,...,k_n) && (5.16)\\ &= \left(p_1,..., \ p_j *_{1i_g} (p_{k_1},p_{k_2}), ..., p_{|\mathbf{p}|}\right) \\ &\qquad *_{\iota^{\mathrm{I}}_{(g,j)}} (k_3,...,k_n) && (5.17)\\ &= \left(p_1,..., \ p_j *_{1i_g} (p_{k_1},...,p_{k_n}), ..., p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} () && (5.18)\\ &\underline{= \left(p_1,..., \ p_j *_{1i_g} (p_{k_1},...,p_{k_n}), ..., p_{|\mathbf{p}|}\right).} && (5.19)\end{aligned}$$

□

Lemma 2. *If the result of the interact function is independent of an interaction of the second particle, then it is independent of all its previous interactions.*

$$\begin{aligned}&\forall p_j, p_k, p_{k'} \in P, g \in G : {}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) = {}_1i_g(p_j, p_k)\\ &\quad\Longrightarrow\ {}_1i_g\left(p_j,\ p_k *_{{}_1i_g} (p_{k'_1}, ..., p_{k'_n})\right) = {}_1i_g\left(p_j,\ p_k\right)\end{aligned} \tag{5.20}$$

Proof.

$$\forall p_j, p_k, p_{k'} \in P, g \in G : {}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) = {}_1i_g(p_j, p_k) \tag{5.21}$$

$$\Longrightarrow {}_1i_g\left(p_j,\ p_k *_{{}_1i_g} (p_{k'_1}, ..., p_{k'_n})\right) \tag{5.22}$$

$$= {}_1i_g\big(p_j,\ \big(p_k *_{{}_1i_g} (p_{k'_1}, ..., p_{k'_{n-1}})\big) *_{{}_1i_g} (p_{k'_n})\big) \tag{5.23}$$

$$= {}_1i_g\big(p_j,\ {}_1i_g\big(p_k *_{{}_1i_g} (p_{k'_1}, ..., p_{k'_{n-1}}),\ p_{k'_n}\big)\big) \tag{5.24}$$

$$= {}_1i_g\big(p_j,\ p_k *_{{}_1i_g} (p_{k'_1}, ..., p_{k'_{n-1}})\big) \tag{5.25}$$

$$\vdots$$

$$= {}_1i_g\left(p_j,\ p_k *_{{}_1i_g} ()\right) \tag{5.26}$$

$$\underline{= {}_1i_g\left(p_j,\ p_k\right)}. \tag{5.27}$$

□

Lemma 3. *If the result of the neighborhood function is independent of a previous interaction it is independent of all previous interactions.*

$$\begin{aligned}&\forall j, k', k'' \in \{1, ..., |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*] :\ u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j) \quad \Longrightarrow\\ &u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,l_1)}} (k_{1,1}, ..., k_{1,n_1}) *_{\iota^{\mathrm{I}}_{(g,l_2)}} ... *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,1}, ..., k_{m,n_1})], j) \ = u([g, \mathbf{p}], j)\end{aligned} \tag{5.28}$$

Proof.

$$\begin{aligned}&\forall j, k', k' \in \{1, ..., |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*] :\\ &\qquad u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j) \quad \Longrightarrow\\ &u([g, \underbrace{\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,l_1)}} (k_{1,1}, ..., k_{1,n_1}) *_{\iota^{\mathrm{I}}_{(g,l_2)}} ... *_{\iota^{\mathrm{I}}_{(g,l_m)}}}_{=:\mathbf{q}} (k_{m,1}, ..., k_{m,n_1})], j)\end{aligned} \tag{5.29}$$

$$= u([g, \underbrace{\mathbf{q} *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,1}, ..., k_{m,n_1}}_{=:\overline{\mathbf{q}}})], j) \tag{5.30}$$

$$= u([g, \overline{\mathbf{q}} *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,n_1})], j) \tag{5.31}$$

$$\underline{= u([g, \mathbf{p}], j)}. \tag{5.32}$$

□

Lemma 4. *Under the constraints that the interact function i is a pull-interaction (5.1) and is independent of the previous interactions (5.2) and the neighborhood is independent of previous interactions (5.3), the outer interaction loop is parallelizable. In our notation:*

$$\forall [g, \mathbf{p}] \in [G, P^*]: \ \iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}]) = \Big(p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ \ldots, \ p_{|\mathbf{p}|} *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\Big). \tag{5.33}$$

Proof.

$$\iota^{\mathrm{N}\times\mathrm{U}}([g, \mathbf{p}]) \tag{5.34}$$

$$= \mathbf{p} *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (1, \ldots, |\mathbf{p}|) \tag{5.35}$$

$$= \Big(\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,1)}} u(g, \mathbf{p}, 1)\Big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (2, \ldots, |\mathbf{p}|) \tag{5.36}$$

$$\overset{\text{Lemma 1}}{=} \underbrace{\Big(\underbrace{p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}}_{\overline{p}_1}, \ p_2, \ldots, p_{|\mathbf{p}|}\Big)}_{=:\, {}^1\overline{\mathbf{p}}} *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (2, \ldots, |\mathbf{p}|) \tag{5.37}$$

$$= \big({}^1\overline{\mathbf{p}} *_{\iota^{\mathrm{I}}_{(g,2)}} u(g, {}^1\overline{\mathbf{p}}, 2)\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (3, \ldots, |\mathbf{p}|) \tag{5.38}$$

$$\overset{\text{Lemma 1}}{=} \underbrace{\big(\overline{p}_1, \underbrace{p_2 *_{1 i_g} \langle {}^1\overline{\mathbf{p}} \rangle_{u(g,{}^1\overline{\mathbf{p}},2)}}_{\overline{p}_2}, \ p_3, \ldots, p_{|\mathbf{p}|}\big)}_{=:\, {}^2\overline{\mathbf{p}}} *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (3, \ldots, |\mathbf{p}|) \tag{5.39}$$

$$\vdots$$

$$= \big({}^{j-1}\overline{\mathbf{p}} *_{\iota^{\mathrm{I}}_{(g,j)}} u(g, {}^{j-1}\overline{\mathbf{p}}, j)\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (j+1, \ldots, |\mathbf{p}|) \tag{5.40}$$

$$\overset{\text{Lemma 1}}{=} \big(\overline{p}_1, \ \overline{p}_2, \ \ldots, \ p_j *_{1 i_g} \langle {}^{j-1}\overline{\mathbf{p}} \rangle_{u(g,\, {}^{j-1}\overline{\mathbf{p}},j)}, \ \ldots, \ p_{|\mathbf{p}|}\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (j+1, \ldots, |\mathbf{p}|) \tag{5.41}$$

$$\overset{\text{Lemma 3}}{=} \big(\overline{p}_1, \ \overline{p}_2, \ \ldots, \ p_j *_{1 i_g} \langle {}^{j-1}\overline{\mathbf{p}} \rangle_{u(g,\mathbf{p},j)}, \ \ldots, \ p_{|\mathbf{p}|}\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (j+1, \ldots, |\mathbf{p}|) \tag{5.42}$$

$$\overset{\text{Lemma 2}}{=} \big(\overline{p}_1, \ \overline{p}_2, \ \ldots, \ p_j *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},j)}, \ \ldots, \ p_{|\mathbf{p}|}\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (j+1, \ldots, |\mathbf{p}|) \tag{5.43}$$

$$= \big(p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ p_2 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},2)}, \ \ldots, \tag{5.44}$$

$$p_j *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},j)}, \ \ldots, p_{|\mathbf{p}|}\big) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}} (j+1, \ldots, |\mathbf{p}|) \tag{5.45}$$

$$= \underline{\big(p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ \cdots, \ p_{|\mathbf{p}|} *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\big)} \tag{5.46}$$

□

Lemma 5. *Under the constraints that the number of particles stays constant (5.4) and the global variable is independent of all particles (5.5), the evolve loop is parallelizable. In our notation:*

$$\epsilon^{\mathrm{N}}\left([g,\mathbf{p}]\right) = \left[g,\ \left(\left\langle\ {}_2e(p_1)\right\rangle_1, ..., \left\langle\ {}_2e(p_{|\mathbf{p}|})\right\rangle_1\right)\right] \tag{5.47}$$

Proof.
Under the constraints (5.4) and (5.5), we can rewrite the first evolution sub-function (3.13) to

$$e^{I}\left([g,\mathbf{p}],\mathbf{q},j\right) = \left[g, \mathbf{q}\circ\left(\left\langle\ {}_2e(g,p_j)\right\rangle_1\right)\right]. \tag{5.48}$$

Using this result, we can rewrite the second evolution sub-function (3.14) to

$$\begin{aligned}
&\epsilon^{\mathrm{N}}\big([g,\mathbf{p}]\big) && (5.49)\\
&= \left[g,()\right]\ *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}\ (1,..,|\mathbf{p}|) && (5.50)\\
&= \epsilon^{\mathrm{I}}\left([g,\mathbf{p}],(),1\right) *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} (2,...,|\mathbf{p}|) && (5.51)\\
&= \left[g,()\circ\left(\left\langle\ {}_2e(g,p_1)\right\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} (2,...,|\mathbf{p}|) && (5.52)\\
&= \epsilon^{\mathrm{I}}\left([g,\mathbf{p}],\left(\left\langle\ {}_2e(g,p_1)\right\rangle_1\right),2\right) *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} (3,...,|\mathbf{p}|) && (5.53)\\
&= \left[g,\left(\left\langle\ {}_2e(g,p_1)\right\rangle_1\right)\circ\left(\left\langle\ {}_2e(g,p2)\right\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} (3,...,|\mathbf{p}|) && (5.54)\\
&= \left[g,\left(\left\langle\ {}_2e(g,p_1)\right\rangle_1, \left\langle\ {}_2e(g,p2)\right\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} (3,...,|\mathbf{p}|) && (5.55)\\
&\vdots\\
&= \left[g,\ \left(\left\langle\ {}_2e(p_1)\right\rangle_1, ..., \left\langle\ {}_2e(p_{|\mathbf{p}|})\right\rangle_1\right)\right] && (5.56)
\end{aligned}$$

□

5.2.3 Parallelizability

Theorem 1 (Parallelizability of particle methods on shared-memory computers)**.** *The parallelized state transition function $\tilde{S}$ returns the same result as the sequential state transition function S under the five conditions (5.1) to (5.5).*

$$\forall[g,\mathbf{p}]\in[G,P^*]:\ \ S\left([g,\mathbf{p}]\right) = \tilde{S}\left([g,\mathbf{p}]\right). \tag{5.57}$$

Proof.
The parallelized state transition function $\tilde{S}$ and sequential state transition function S differ only in their state transition step. Hence, it is sufficient to prove that

$$s\left([g,\mathbf{p}]\right) = \tilde{s}\left([g,\mathbf{p}]\right). \tag{5.58}$$

Using the lemmata 1 to 5, we can prove parallelized and sequential state transition function are equivalent, i.e., they produce identical results. We use Lemma 4 and Lemma 5 to prove it. We insert

$$\epsilon^{\mathrm{N}}\left([g,\mathbf{p}]\right) = \left[g,\ \left(\left\langle {}_2e(p_1)\right\rangle_1, ...,\ \left\langle {}_2e(p_{|\mathbf{p}|})\right\rangle_1\right)\ \right], \tag{5.47}$$

and

$$\iota^{\mathrm{N}\times\mathrm{U}}\left([g,\mathbf{p}]\right) = \left(p_1 *_{1 i_g} \langle\mathbf{p}\rangle_{u(g,\mathbf{p},1)}, ..., \; p_{|\mathbf{p}|} *_{1 i_g} \langle\mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right) \tag{5.33}$$

into

$$s\left([g,\mathbf{p}]\right) := \left[\mathring{e}(\overline{g}),\; \overline{\mathbf{p}}\right] \;\text{ for }\; [\overline{g},\overline{\mathbf{p}}] := \epsilon^{\mathrm{N}}\left([g,\; \iota^{\mathrm{N}\times\mathrm{U}}([g,\mathbf{p}])]\right). \tag{3.15}$$

and get

$$\begin{aligned} s\left([g,\mathbf{p}]\right) &= \left[\mathring{e}(g), \left(\left\langle {}_2e\left(p_1 *_{1 i_g} \langle\mathbf{p}\rangle_{u(g,\mathbf{p},1)}\right)\right\rangle_1, ..., \; \left\langle {}_2e\left(p_{|\mathbf{p}|} *_{1 i_g} \langle\mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)\right\rangle_1\right)\right] && (5.10) \\ &= \underline{\tilde{s}\left([g,\mathbf{p}]\right)}. && (5.59) \end{aligned}$$

From this, we can conclude

$$\forall [g,\mathbf{p}] \in [G, P^*] : \; S\left([g,\mathbf{p}]\right) = \tilde{S}\left([g,\mathbf{p}]\right). \tag{5.60}$$

Therefore, the parallelized state transition function $\tilde{S}$ returns the same result as the sequential state transition function S under the five conditions (5.1) to (5.5). □

5.2.4 Time Complexity

The time complexity of an algorithm describes the asymptotic behavior of the run-time as a function of the input size. In the case of the state transition function, the input size is the length of the initial particle tuple $\mathbf{p}^1$, assuming that a constant bounds the size of the global variable and the particles. The condition that the evolve function does not change the number of particles (5.4) restricts the number of particles to be constant over all state transition steps

$$\forall t \in \{1, ..., T\} : \left|\mathbf{p}^t\right| = \left|\mathbf{p}^1\right|. \tag{5.61}$$

We assume a maximum time complexity τ for each function for all state transition steps. We indicate the corresponding function with a subscript, τ_i, τ_e, $\tau_{\mathring{e}}$, τ_f, τ_u. We also assume a maximum size of the neighborhood ς_u for each particle and all steps. We assume further that the maximum time complexities are independent of the particle method instance $[g^1, \mathbf{p}^1]$ except for the neighborhood. This is true for many canonical particle methods. We indicate the neighborhood function's possible dependency on $\mathbf{p}^1$ with a superscript. Using these maxima of the time complexities, we can derive an upper bound for the time complexity of the sequential state transition function

$$\tau_{S([g^1,\mathbf{p}^1])} \leq T\left(\left|\mathbf{p}^1\right|\left(\varsigma_u^{\mathbf{p}^1}\tau_i + \tau_u^{\mathbf{p}^1} + \tau_e\right) + \tau_f + \tau_{\mathring{e}}\right), \tag{5.62}$$

and for the parallelized state transition function on one processor (CPU)

$$\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1) \leq T\left(\left|\mathbf{p}^1\right|\left(\varsigma_u^{\mathbf{p}^1}\tau_i + \tau_u^{\mathbf{p}^1} + \tau_e\right) + \tau_f + \tau_{\mathring{e}}\right), \tag{5.63}$$

and for the time complexity of the parallelized state transition function on n_{CPU} processors, where all processors share the same memory

$$\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(n_{CPU}) \leq T\left(\Xi\left(|\mathbf{p}^1|, n_{CPU}\right)\left(\varsigma_u^{\mathbf{p}^1}\tau_i + \tau_u^{\mathbf{p}^1} + \tau_e\right) + \tau_f + \tau_{\mathring{e}}\right), \tag{5.64}$$

where

$$\Xi(n_{particle}, n_{CPU}) := \left\lceil \frac{n_{particle}}{n_{CPU}} \right\rceil . \tag{5.65}$$

We use ceil $\left\lceil \frac{n_{particle}}{n_{CPU}} \right\rceil$ to account for the atomic nature of the computation of one particle. In general, it can not be split across processors. Note that we define a processor here as a single sequential processing unit that shares its memory with all other processors.

These upper bounds on the time complexities allow us to derive closed-form expressions for the bounds on the speed-ups for both strong scaling according to Amdahl's law [3] and weak scaling according to Gustafson's law [37].

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

$$speedup_{\text{Amdal}}(n_{CPU}) = \frac{\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1)}{\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(n_{CPU})} \tag{5.66}$$

$$\approx \frac{\left|\mathbf{p}^1\right| (\varsigma_u^{\mathbf{p}^1} \tau_i + \tau_u^{\mathbf{p}^1} + \tau_e) + \tau_f + \tau_{\mathring{e}}}{\Xi\left(|\mathbf{p}^1|, n_{CPU}\right) (\varsigma_u^{\mathbf{p}^1} \tau_i + \tau_u^{\mathbf{p}^1} + \tau_e) + \tau_f + \tau_{\mathring{e}}}. \tag{5.67}$$

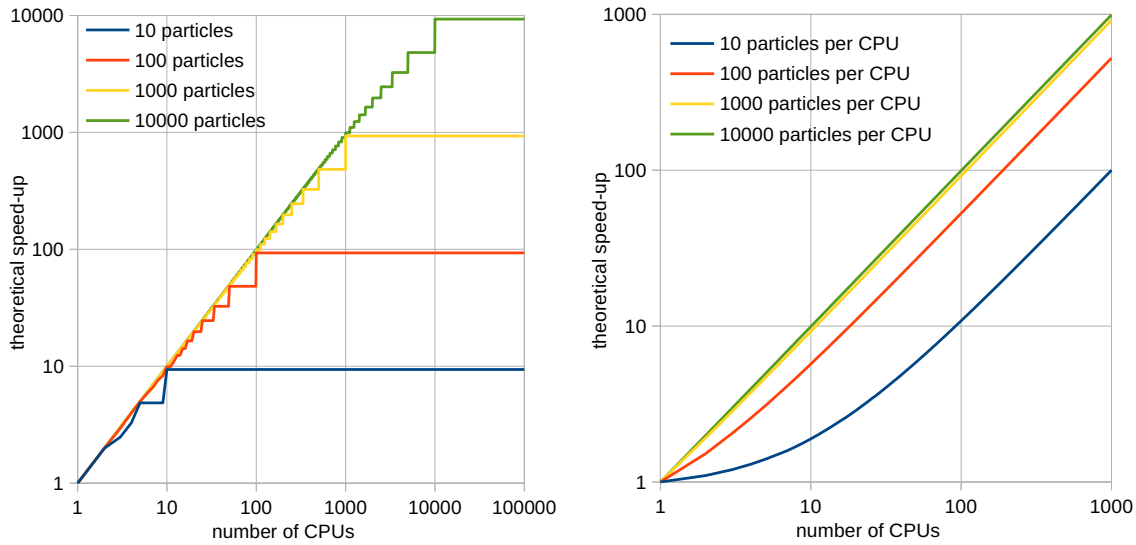
In this case, we can increase n_{CPU} until we reach the number of particles $|\mathbf{p}^1|$. After that, there will be no further speed-up. But also, until then, we find a step-like behavior with increasing n_{CPU} because particles cannot be split across processors as visualized in figure 5.2a.

Second, Gustafson's law [37] provides an upper bound on the speed-up of the scheme on multiple processors when the ratio of problem size to processor number is constant while increasing the number of processors (weak scaling); $\frac{|\mathbf{p}^1|}{n_{\text{CPU}}} = \text{const}$. We achieve this by setting the initial particle tuple such that the number of particles $|\mathbf{p}^1_{n_{CPU}}| = n_{CPU} \cdot |\mathbf{p}^1|$, where $|\mathbf{p}^1|$ is constant. For a perfectly fitting processor interconnect network topology, we predict a linear speed-up on average as visualized in figure 5.2b.

$$speedup_{\text{Gustafson}}(n_{CPU}) = \frac{\tau_{\tilde{S}\left([g^1,\mathbf{p}^1_{n_{CPU}}]\right)}(1)}{\tau_{\tilde{S}\left([g^1,\mathbf{p}^1_{n_{CPU}}]\right)}(n_{CPU})} \tag{5.68}$$

$$\approx \frac{n_{CPU} \left|\mathbf{p}^1\right| (\varsigma_u^{\mathbf{p}^1_{n_{CPU}}} \tau_i + \tau_u^{\mathbf{p}^1_{n_{CPU}}} + \tau_e) + \tau_f + \tau_{\mathring{e}}}{\Xi\left(|\mathbf{p}^1_{n_{CPU}}|, n_{CPU}\right) (\varsigma_u^{\mathbf{p}^1_{n_{CPU}}} \tau_i + \tau_u^{\mathbf{p}^1_{n_{CPU}}} + \tau_e) + \tau_f + \tau_{\mathring{e}}}. \tag{5.69}$$

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



(a) Speed-up according to Amdahl's law. The parameters are chosen to be $\tau_f = \tau_{\overset{\circ}{e}} = 1$, $\tau_i = \tau_e = 3$, and $\tau_u = \varsigma_u = 6$.

(b) Speed-up according to Gustafson's law. The parameters are chosen to be $\tau_f = 1$, $\tau_{\overset{\circ}{e}} = 1000$, $\tau_i = \tau_e = 3$, and $\tau_u = \varsigma_u = 2$.

Figure 5.2: Theoretical speed-up bounds for the shared-memory parallelization.

Note that we fixed the neighborhood time and space complexity to a constant. We also used in figure 5.2b an unusually high time complexity for the evolve function of the global variable to visually demonstrate the influence of the number of particles per processor. More commonly, we expect for small numbers of particles per processor a more similar scaling to the higher numbers of particles per processor.

5.2.5 Application

We can now use the theorem to check if we can parallelize the particle method from chapter 4.

The three-dimensional diffusion application based on PSE and Euler integration is formulated with a pull interaction. To change particle properties, the interact function uses only properties that are not changed by it. Hence, it is impossible to transfer the result of an interaction through an interaction to another particle. Meaning, the interact function is independent of previous interactions. The same argumentation holds for the neighborhood function. The interact function does not change any property used by the neighborhood function. We formally check these conditions. Independence of the interact function from previous interactions:

$$
\begin{aligned}
&{}_1 i_g(p_j, {}_1 i_g(p_k, p_{k'})) && (5.70)\\
&= {}_1 i_g \left(p_j, \left(\underline{x}_k,\ w_k,\ \Delta w_k + \frac{(w_{k'} - w_k)}{\left(\frac{|\underline{x}_{k'} - \underline{x}_k|}{\epsilon}\right)^{10} + 1} \right) \right) && (5.71)\\
&= \left(\underline{x}_j,\ w_j,\ \Delta w_j + \frac{(w_k - w_j)}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1} \right) && (5.72)\\
&\underline{=_1\ i_g(p_j, p_k)} && (5.73)
\end{aligned}
$$

Independence of the neighborhood function from previous interactions.

$$
\begin{aligned}
&u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j) && (5.74)\\
&= u([g, (p_1, ..., {}_1 i_g(p_{k'}, p_{k''}), ..., p_{|\mathbf{p}|})], j) && (5.75)\\
&= u\left(\left[g, \left(p_1, ..., \begin{pmatrix} \underline{x}_{k'} \\ w_{k'} \\ \Delta w_{k'} + \frac{(w_{k''} - w_{k'})}{\left(\frac{|\underline{x}_{k''} - \underline{x}_{k'}|}{\epsilon}\right)^{10} + 1} \end{pmatrix}^{\mathbf{T}}, ..., p_{|\mathbf{p}|} \right) \right], j \right) && (5.76)\\
&= (k \in (1, ..., |\mathbf{p}|) : p_k, p_j \in \mathbf{p} \wedge 0 < |\underline{x}_k - \underline{x}_j| \leq r_c) && (5.77)\\
&\underline{= u([g, \mathbf{p}], j)} && (5.78)
\end{aligned}
$$

Note that the interaction of particle $p_{k'}$ with $p_{k''}$ does not change the position of $p_{k'}$. Hence, the neighborhood function is not influenced by the interaction.

The other two conditions are directly matched by the evolve method. There is a constant number of particles, and the global variable is independent of all particles.

Therefore, the diffusion example is parallelizable with this shared memory scheme.

But also Conway's game of life (sec. 4.7) fulfills these conditions and others like the SPH example (sec. 4.4) could be rewritten a bit to fulfill the constraints. For the SPH example the change would be a simple swap from a symmetric interaction to a pull-interaction by not changing the second particle anymore in the interact function and account for this in the neighborhood function by having the full neighborhood instead of the asymmetric one.

5.3 Particle Methods on Distributed Memory Systems

We consider here the parallelization of particle methods on distributed-memory computers. The resulting scheme is valid for particle methods with pull interactions and without global operations. It is based on cell lists and checkerboard-like domain decomposition, a well-known approach often used in practical code implementations.

5.3.1 Parallelization Scheme

Conditions

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

$$p = (\ldots, \underline{x}, \ldots) \in P, \tag{5.79}$$

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

$$u\left([g, \mathbf{p}], j\right) := \left(k \in (1, \ldots, |\mathbf{p}|) : p_k, p_j \in \mathbf{p} \ \wedge |\underline{x}_k - \underline{x}_j| \leq r_c \wedge \Omega(g, p_k, p_j)\right) \\ \text{with } \Omega : G \times P \times P \to \{\top, \bot\}, \tag{5.80}$$

where Ω is an additional constraint for more flexibility, e.g., to avoid self interactions $\Omega(g, p_k, p_j) := (p_k \neq p_j)$. The position $\underline{x}$ of a particle is limited to a cuboidal spatial domain of dimension d for all states/times t

$$\forall t \in \{1, \ldots, T\} \ \forall p_j^t \in \mathbf{p}^t : \underline{x}_j^t \in [\underline{D}_{\min}, \underline{D}_{\max}) \subset \mathbb{R}^d. \tag{5.81}$$

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

$$\forall t \in \{1, \ldots, T-1\} \ \forall p_j^t \in \mathbf{p}^t : |\underline{x}_j^t - \underline{x}_j^{t+1}| \leq r_c. \tag{5.82}$$

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, and DEM. Here, we require this restriction because we will consider the limit of the finest-possible domain decomposition, where each sub-domain is of the smallest possible edge length r_c. In practice, unless the numerical scheme requires otherwise, the condition can be relaxed to require that no particle travels further than one sub-domain per transition step.

Further, the interaction is limited to a pull interaction:

$$\forall p_j, p_k \in P, g \in G : \ i(g, p_j, p_k) = (\overline{p}_j, p_k), \tag{5.83}$$

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

$$\forall p_j, p_k, p_{k'} \in P, g \in G : \ {}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) = {}_1i_g(p_j, p_k) \tag{5.84}$$

$$\text{for} \quad {}_1i_g(p_j, p_k) := \langle i(g, p_j, p_k) \rangle_1, \tag{5.85}$$

and be order-independent

$$\forall p_j, p_k, p_{k'} \in P, g \in G: \ {}_1 i_g({}_1 i_g(p_j, p_k), p_{k'}) = {}_1 i_g({}_1 i_g(p_j, p_{k'}), p_k). \tag{5.86}$$

Order independence is not strictly required, but it is not limiting in practice and avoids the sorting of particles. The neighborhood function u needs to be independent of previous interactions, so the cell list can be prepared before the interactions happen. Here, $*_{\iota^{\mathrm{I}}}$ is the composition operator defined in definition 7 and used on ι^{I}, whereas ι^{I} is the first transition sub-function defined in (3.10).

$$\forall j, k', k'' \in \{1, \ldots, |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*]: \ u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j). \tag{5.87}$$

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

$$\forall g \in G, p \in P: e(g, p) = (g, \mathbf{q}), \tag{5.88}$$

which avoids global operations and their $\mathcal{O}(\log(\textit{number of processes}))$ communication complexity [35].

Scheme in Formulas

These conditions are used to construct a parallelization scheme for particle methods on distributed-memory systems. The Nassi-Shneiderman diagram provides a visual overview of how the state transition function S (3.16) is parallelized.

Under these constraints on a particle method, we formulate the parallelization of any particle method onto multiple (distributed-memory) processes. A "process" is an independent and self-contained unit of computation. Processes can execute in parallel (i.e., at the same time) and possess their own exclusive memory. This follows the standard model of a distributed-memory parallel computer [42]. The parallelization scheme is based on the classic cell-list algorithm [41]. 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

$$\underline{\overline{\mathbf{I}}} = \begin{pmatrix} I_1 \\ \vdots \\ I_d \end{pmatrix} := \left\lfloor \tfrac{1}{r_c}(\underline{D}_{\max} - \underline{D}_{\min}) + \underline{\overline{\mathbf{1}}} \right\rfloor. \tag{5.89}$$

The total number of cells is

$$N_{\text{cell}} := \prod_{l=1}^{d} I_l. \tag{5.90}$$

The highest degree of parallelism is reached when assigning each cell-list cell to a separate process. 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 simultaneously and that the network behaves as a fully connected graph. Then, the communications between processes are at risk of overlapping, potentially leading to processes having to wait for one another. Hence, one should avoid simultaneous communication with identical partners. The simplest way to

avoid overlapping communications is if all communicating processes have at least two inactive processes between them. This results in a checkerboard-like pattern. The number of active processes/ cells in each dimension is given by the vector

$$ {}^{k}\underline{\overline{\mathbf{I}}}^{*} = \begin{pmatrix} {}^{k}I_1^* \\ \vdots \\ {}^{k}I_d^* \end{pmatrix} := \left\lfloor \tfrac{1}{3}\left(\underline{\overline{\mathbf{I}}} - \underline{\overline{\mathbf{3}}}\iota(k) + \underline{\overline{\mathbf{3}}}\right)\right\rfloor . \tag{5.91} $$

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

$$ {}^{k}N_{\text{cell}}^{*} := \prod_{l=1}^{d} {}^{k}I_l^{*}. \tag{5.92} $$

We address the communicating processes for each k by

$$ \gamma(k,j) := \underline{\overline{\mathbf{I}}}\iota^{-1}\left({}^{k}\underline{\overline{\mathbf{I}}}^{*}\iota(j) \cdot 3 + \underline{\overline{\mathbf{3}}}\iota(k) - \underline{\overline{\mathbf{3}}}\right). \tag{5.93} $$

The l-th neighbor cell of the t-th process is

$$ \beta(t,l) := \underline{\overline{\mathbf{I}}}\iota^{-1}\left(\underline{\overline{\mathbf{I}}}\iota(t) + \underline{\overline{\mathbf{3}}}\iota(l) - \underline{\overline{\mathbf{2}}}\right). \tag{5.94} $$

The result of β is undefined unless the result of $\underline{\overline{\mathbf{I}}}\iota^{-1}$ (def. 14 and (5.89)) 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 5.3 as a condition for the initial particle distribution. Hence, each cell k initially contains a tuple of particles $\mathbf{p}_k^1$. To ensure no particle is lost, we require that the concatenation of these particle tuples is a permutation π of the initial particle tuple $\mathbf{p}^1$

$$ \mathbf{p}_1^1 \circ \ldots \circ \mathbf{p}_{N_{\text{cell}}}^1 = \pi\left(\mathbf{p}^1\right) \tag{5.95} $$

and that particles are distributed according to their position

$$ \forall p_j^1 \in \mathbf{p}^1 : p_j^1 \in \mathbf{p}_w^1 \quad \text{with} \quad w = \underline{\overline{\mathbf{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^1 - \underline{D}_{\min})\right\rfloor + \underline{\overline{\mathbf{1}}}\right). \tag{5.96} $$

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 5.3. Therefore, the particle storage of each process is compartmentalized cell-wise:

$$ \underline{\mathbf{p}} = \begin{pmatrix} \langle \underline{\mathbf{p}} \rangle_1 \\ \vdots \\ \langle \underline{\mathbf{p}} \rangle_{3^d} \end{pmatrix}^{\top} \in (P^*)^{3^d}, \tag{5.97} $$

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.

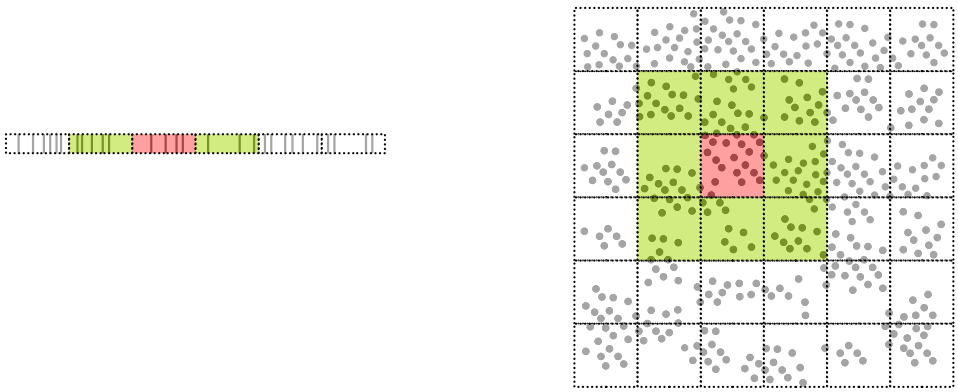


Figure 5.3: Illustrations of a cell list in one dimension (left) and two dimensions (right). Particle positions in one dimension are shown as solid vertical lines and in two dimensions as dots. Dotted lines show cell boundaries. Each process is assigned one cell (red) in the finest possible distribution. It is the center cell in the particle storage of that process with copies of the particles from the neighboring processes/cells (green) to either side, see (5.99).

The particle storages of all processes (PROC) together is

$$\mathcal{P} = \begin{pmatrix} {}^{[\mathrm{PROC}1]}\underline{\mathbf{p}} \\ \vdots \\ {}^{[\mathrm{PROC}N_{\mathrm{cell}}]}\underline{\mathbf{p}} \end{pmatrix}^\top \in \left((P^*)^{3^d} \right)^{N_{\mathrm{cell}}}. \tag{5.98}$$

The initial particle tuples in each process' particle storage are

$${}^{[\mathrm{PROC}n]}\underline{\mathbf{p}}^1 = \Big((), \ldots, (), \underbrace{\mathbf{p}_n^1}_{\frac{3^d+1}{2}\text{-th entry}}, (), \ldots, () \Big). \tag{5.99}$$

Then, the initial particle storage of all processes is

$$\mathcal{P}^1 := \begin{pmatrix} {}^{[\mathrm{PROC}1]}\underline{\mathbf{p}}^1 \\ \vdots \\ {}^{[\mathrm{PROC}N_{\mathrm{cell}}]}\underline{\mathbf{p}}^1 \end{pmatrix}^\top. \tag{5.100}$$

The global variable storage of all processes is

$$\mathcal{G}^1 := (g^1, \ldots, g^1) \in G^{N_{\mathrm{cell}}}. \tag{5.101}$$

We introduce a second global variable $\tilde{g}$ that contains the cell-list specific parameters (5.81),(5.89)

$$\tilde{g} = \left(\underline{D}_{\min}, \underline{D}_{\max}, d, \underline{\overline{\mathbf{I}}}\right). \tag{5.102}$$

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

$$copy_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l) := \left(\left\langle \underline{\mathbf{p}} \right\rangle_1, \ldots, \left\langle \underline{\mathbf{p}} \right\rangle_{l-1}, {}^{[\mathrm{PROC}\beta(w,l)]}\left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}}, \left\langle \underline{\mathbf{p}} \right\rangle_{l+1}, \ldots, \left\langle \underline{\mathbf{p}} \right\rangle_{3^d}\right). \tag{5.103}$$

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

$$copy_{\tilde{g}}^{active}(\mathcal{P}, k) := \begin{pmatrix} {}^{[\mathrm{PROC}1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}\gamma(k,1)-1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{[\mathrm{PROC}\gamma(k,1)]}\left\langle \underline{\mathbf{p}} \right\rangle *_{copy_{(\tilde{g},\mathcal{P},\gamma(k,1))}} \left(1, \ldots, 3^d\right) \\ {}^{[\mathrm{PROC}\gamma(k,1)+1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}\gamma(k,l)-1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{[\mathrm{PROC}\gamma(k,l)]}\left\langle \underline{\mathbf{p}} \right\rangle *_{copy_{(\tilde{g},\mathcal{P},\gamma(k,l))}} \left(1, \ldots, 3^d\right) \\ {}^{[\mathrm{PROC}\gamma(k,l)+1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^*_{\mathrm{cell}}\right)-1\right]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^*_{\mathrm{cell}}\right)\right]}\left\langle \underline{\mathbf{p}} \right\rangle *_{copy_{\left(\tilde{g},\mathcal{P},\gamma\left(k,{}^{k}N^*_{\mathrm{cell}}\right)\right)}} \left(1, \ldots, 3^d\right) \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^*_{\mathrm{cell}}\right)+1\right]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}N_{\mathrm{cell}}]}\left\langle \underline{\mathbf{p}} \right\rangle \end{pmatrix}^\top. \tag{5.104}$$

Doing so for all distinct checkerboard-like configurations results in

$$copy_{\tilde{g}}^{ALL}(\mathcal{P}) := \mathcal{P} *_{copy_{\tilde{g}}^{active}} \left(1, \ldots, 3^d\right). \tag{5.105}$$

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

$$interaction_g(\underline{\mathbf{p}}) := \begin{pmatrix} \left\langle \mathbf{q} \right\rangle_1 \; *_{1\, i_g} \left\langle \bigcirc_{w=1}^{3^d} \left\langle \underline{\mathbf{p}} \right\rangle_w \right\rangle_{u\left(\left[g, \bigcirc_{w=1}^{3^d} \left\langle \underline{\mathbf{p}} \right\rangle_w\right], z+1\right)} \\ \vdots \\ \left\langle \mathbf{q} \right\rangle_{|\mathbf{q}|} \; *_{1\, i_g} \left\langle \bigcirc_{w=1}^{3^d} \left\langle \underline{\mathbf{p}} \right\rangle_w \right\rangle_{u\left(\left[g, \bigcirc_{w=1}^{3^d} \left\langle \underline{\mathbf{p}} \right\rangle_w\right], z+|\mathbf{q}|\right)} \end{pmatrix}^\top, \tag{5.106}$$

where $z = \left| \bigcirc_{w=1}^{\frac{3^d+1}{2}-1} \langle \underline{\mathbf{p}} \rangle_w \right|$ and $\mathbf{q} = \langle \underline{\mathbf{p}} \rangle_{\frac{3^d+1}{2}}$. 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:

$$step_{(\tilde{g},g)}(\underline{\mathbf{p}}) := \begin{pmatrix} \langle \underline{\mathbf{p}} \rangle_1 \\ \vdots \\ \langle \underline{\mathbf{p}} \rangle_{\frac{3^d+1}{2}-1} \\ {}_2\epsilon^{\mathrm{N}} \left(\left[g, interaction_g \left(\underline{\mathbf{p}} \right) \right] \right) \\ \langle \underline{\mathbf{p}} \rangle_{\frac{3^d+1}{2}+1} \\ \vdots \\ \langle \underline{\mathbf{p}} \rangle_{3^d} \end{pmatrix}^\top . \tag{5.107}$$

Calculating the state transition step on all processes results in

$$step^{ALL}_{(\tilde{g},\mathcal{G})}(\mathcal{P}) := \left(step_{\left(\tilde{g},\ {}^{[\mathrm{PROC1}]}g\right)} \left({}^{[\mathrm{PROC1}]}\underline{\mathbf{p}} \right),\ \ldots,\ step_{\left(\tilde{g},\ {}^{[\mathrm{PROC}|\mathcal{P}|]}g\right)} \left({}^{[\mathrm{PROC}|\mathcal{P}|]}\underline{\mathbf{p}} \right) \right), \tag{5.108}$$

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 $step^{ALL}_{(\tilde{g},\mathcal{G})}$, 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 re-assigned 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

$$dist_{(\tilde{g},g,j)}(\underline{\mathbf{p}}, q) := \begin{pmatrix} \langle \underline{\mathbf{p}} \rangle_1 \\ \vdots \\ \langle \underline{\mathbf{p}} \rangle_{\alpha-1} \\ \langle \underline{\mathbf{p}} \rangle_\alpha \circ (q) \\ \langle \underline{\mathbf{p}} \rangle_{\alpha+1} \\ \vdots \\ \langle \underline{\mathbf{p}} \rangle_{3^d} \end{pmatrix}^\top , \tag{5.109}$$

where

$$q = (\ldots, \underline{x}, \ldots) \in P\,, \quad \alpha := {}^{\overline{\mathbf{3}}}\iota^{-1} \left(\left\lfloor \tfrac{1}{r_c} (\underline{x} - \underline{D}_{min}) \right\rfloor - {}^{\overline{\mathbf{I}}}\iota(j) + \overline{\mathbf{3}} \right). \tag{5.110}$$

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

$$dist^N_{(\tilde{g},g,j)}(\mathbf{q}) := ((), \ldots, ()) *_{dist_{(\tilde{g},g,j)}} \mathbf{q}, \tag{5.111}$$

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

$$dist^{ALL}_{(\tilde{g},\mathcal{G})}(\mathcal{P}) := \begin{pmatrix} dist^{N}_{(\tilde{g},\ {}^{[\mathrm{PROC1}]}g,1)} \left({}^{[\mathrm{PROC1}]} \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}} \right) \\ \vdots \\ dist^{N}_{(\tilde{g},\ {}^{[\mathrm{PROC}|\mathcal{P}|]}g,|\mathcal{P}|)} \left({}^{[\mathrm{PROC}|\mathcal{P}|]} \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}} \right) \end{pmatrix}^{\top}, \tag{5.112}$$

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 $dist^{ALL}_{(\tilde{g},\mathcal{G})}$ 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 w-th process is done by the function

$$collect_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l) := \begin{pmatrix} \left\langle \underline{\mathbf{p}} \right\rangle_1 \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}-1} \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}} \circ {}^{[\mathrm{PROC}\beta(w,l)]} \left\langle \underline{\mathbf{p}} \right\rangle_{\underline{\overline{\mathbf{3}}}_{\iota}{}^{-1}\left(\underline{\overline{\mathbf{4}}}-\underline{\overline{\mathbf{3}}}_{\iota}(l)\right)} \\ \left\langle \underline{\mathbf{p}} \right\rangle_{\frac{3^d+1}{2}+1} \\ \vdots \\ \left\langle \underline{\mathbf{p}} \right\rangle_{3^d} \end{pmatrix}. \tag{5.113}$$

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 k-th

checkerboard-like pattern, the collection function is

$$collect_{\tilde{g}}^{N}(\mathcal{P},k) := \begin{pmatrix} {}^{[\mathrm{PROC}1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}\gamma(k,1)-1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{[\mathrm{PROC}\gamma(k,1)]}\left\langle \underline{\mathbf{p}} \right\rangle *_{collect_{(\tilde{g},\mathcal{P},\gamma(k,1))}} \left(1,\ldots,3^d\right) \\ {}^{[\mathrm{PROC}\gamma(k,1)+1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}\gamma(k,l)-1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{[\mathrm{PROC}\gamma(k,l)]}\left\langle \underline{\mathbf{p}} \right\rangle *_{collect_{(\tilde{g},\mathcal{P},\gamma(k,l))}} \left(1,\ldots,3^d\right) \\ {}^{[\mathrm{PROC}\gamma(k,l)+1]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^{*}_{\mathrm{cell}}\right)-1\right]}\left\langle \underline{\mathbf{p}} \right\rangle \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^{*}_{\mathrm{cell}}\right)\right]}\left\langle \underline{\mathbf{p}} \right\rangle *_{collect_{\left(\tilde{g},\mathcal{P},\gamma\left(k,{}^{k}N^{*}_{\mathrm{cell}}\right)\right)}} \left(1,\ldots,3^d\right) \\ {}^{\left[\mathrm{PROC}\gamma\left(k,{}^{k}N^{*}_{\mathrm{cell}}\right)+1\right]}\left\langle \underline{\mathbf{p}} \right\rangle \\ \vdots \\ {}^{[\mathrm{PROC}|\mathcal{P}|]}\left\langle \underline{\mathbf{p}} \right\rangle \end{pmatrix}^{\top}, \tag{5.114}$$

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

$$collect_{\tilde{g}}^{ALL}(\mathcal{P}) := \mathcal{P} *_{collect_{\tilde{g}}^{N}} \left(1,\ldots,3^d\right), \tag{5.115}$$

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:

$$\tilde{s}_{\tilde{g}}\left([\mathcal{G},\mathcal{P}]\right) = \left[\begin{pmatrix} \overset{\circ}{e}\left(\langle \mathcal{G} \rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\langle \mathcal{G} \rangle_{N_{\mathrm{cell}}}\right) \end{pmatrix}^{\top}, collect_{\tilde{g}}^{ALL}\left(dist_{(\tilde{g},\mathcal{G})}^{ALL}\left(step_{\mathcal{G}}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P})\right)\right)\right) \right]. \tag{5.116}$$

We use this parallel state-transition step $\tilde{s}_{\tilde{g}}$ to define the parallel state transition function similar to (3.16)

$$\begin{aligned} \tilde{S}\left(\left[\mathcal{G}^1,\mathcal{P}^1\right]\right) := \left[\mathcal{G}^T,\mathcal{P}^T\right] \quad \longleftrightarrow \\ f\left(\left\langle \mathcal{G}^T \right\rangle_1\right) = \top \\ \wedge \quad \forall t \in \{2,\ldots,T\}: \ \left[\mathcal{G}^t,\mathcal{P}^t\right] = \tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^{t-1},\mathcal{P}^{t-1}\right]\right) \ \wedge \ f\left(\left\langle \mathcal{G}^{t-1} \right\rangle_1\right) = \bot. \end{aligned} \tag{5.117}$$

Scheme as Nassi-Shneiderman diagram

The complete parallelization scheme is summarized by the Nassi-Shneiderman diagram in figure 5.4.

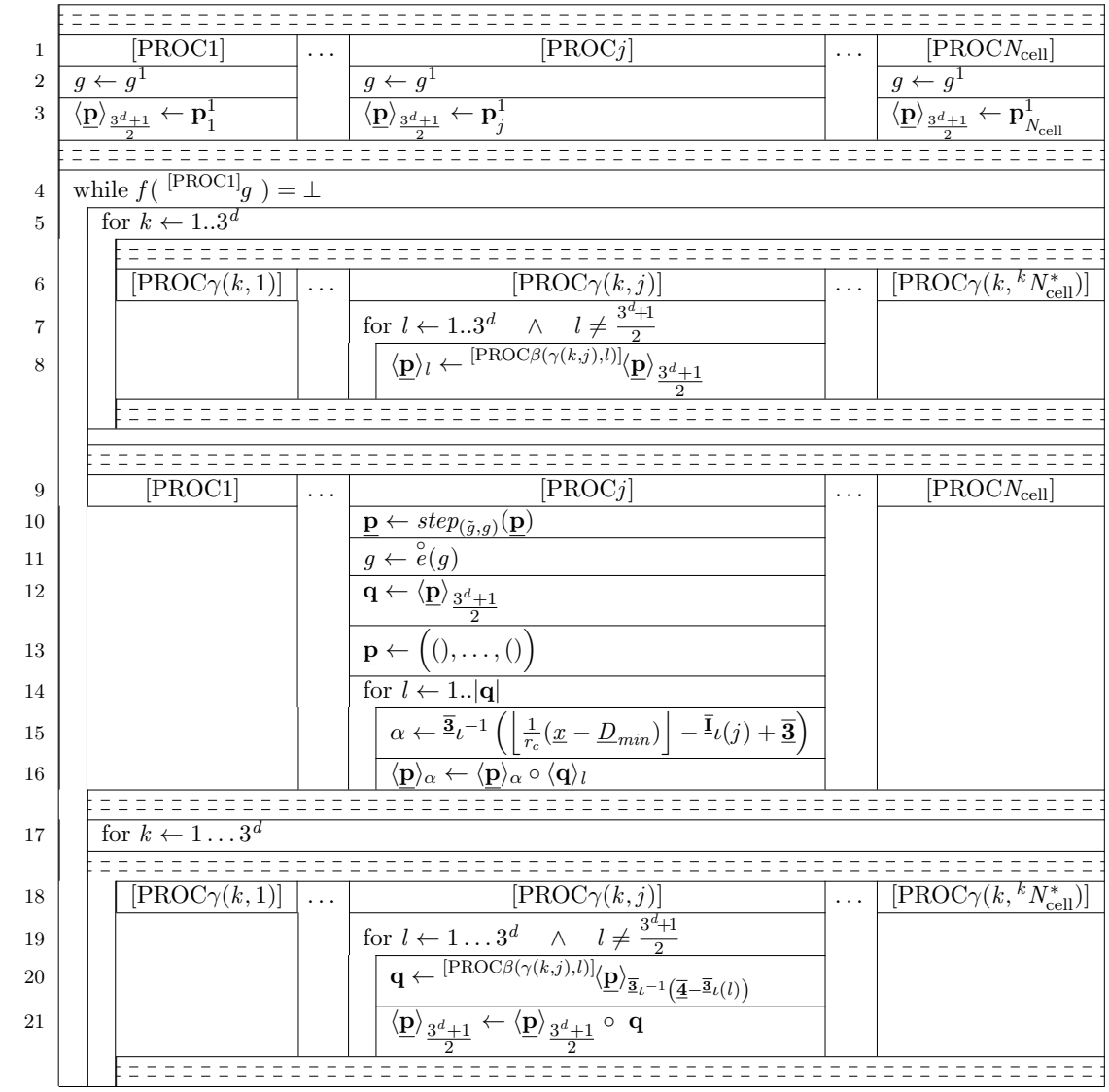


Figure 5.4: Nassi-Shneiderman diagram of the distributed-memory parallelization of the outer loop of a particle method with pull interactions and without global operations. The dashed double lines mark parallel sections of the algorithm.

The algorithm starts by initializing the global variable storage (line 2) 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, 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 other 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' 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).

5.3.2 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 lemmata.

Lemma 6. *$\underline{\bar{\mathbf{I}}}_\iota$ and $\underline{\bar{\mathbf{I}}}_\iota^{-1}$ are bijections and mutual functional inverses, i.e., $(\underline{\bar{\mathbf{I}}}_\iota^{-1})^{-1} = \underline{\bar{\mathbf{I}}}_\iota$.*

Proof.
We need to prove that $\underline{\bar{\mathbf{I}}}_\iota$ and $\underline{\bar{\mathbf{I}}}_\iota^{-1}$ are bijective and their respective inverse function.

$\underline{\bar{\mathbf{I}}}_\iota$ and $\underline{\bar{\mathbf{I}}}_\iota^{-1}$ are bijective and their respective inverse function if and only if

$$\forall j \in \left\{1, ..., \prod_{\delta=1}^{d} I_\delta\right\} : \underline{\bar{\mathbf{I}}}_\iota^{-1}\left(\underline{\bar{\mathbf{I}}}_\iota(j)\right) = j \tag{5.118}$$

$$\wedge \quad \forall \underline{j} \in \mathbb{N}^d \cap \left[\underline{\bar{\mathbf{1}}}, \underline{\bar{\mathbf{I}}}\right] : \underline{\bar{\mathbf{I}}}_\iota\left(\underline{\bar{\mathbf{I}}}_\iota^{-1}(\underline{j})\right) = \underline{j} \tag{5.119}$$

First, we proof (5.118):

$$\underline{\bar{\mathbf{I}}}_\iota^{-1}\left(\underline{\bar{\mathbf{I}}}_\iota(j)\right) = 1 + \left(\left((j-1) - \left\lfloor\frac{j-1}{I_1}\right\rfloor I_1 + 1\right) - 1\right) \tag{5.120}$$

$$+ \left(\left(\left\lfloor\frac{j-1}{I_1}\right\rfloor - \left\lfloor\frac{j-1}{I_1 I_2}\right\rfloor I_2 + 1\right) - 1\right) I_1 \tag{5.121}$$

$$+ \left(\left(\left\lfloor\frac{j-1}{I_1 I_2}\right\rfloor - \left\lfloor\frac{j-1}{I_1 I_2 I_3}\right\rfloor I_3 + 1\right) - 1\right) I_1 I_2 \tag{5.122}$$

$$+ ... + \left(\left(\left\lfloor\frac{j-1}{\prod_{t=1}^{l-1} I_t}\right\rfloor - \left\lfloor\frac{j-1}{\prod_{t=1}^{l} I_t}\right\rfloor I_l + 1\right) - 1\right) \prod_{t=1}^{l-1} I_t \tag{5.123}$$

$$+ \left(\left(\left\lfloor\frac{j-1}{\prod_{t=1}^{l} I_t}\right\rfloor - \left\lfloor\frac{j-1}{\prod_{t=1}^{l+1} I_t}\right\rfloor I_{l+1} + 1\right) - 1\right) \prod_{t=1}^{l} I_t \tag{5.124}$$

$$+ ... + \left(\left(\left\lfloor\frac{j-1}{\prod_{t=1}^{d-2} I_t}\right\rfloor - \left\lfloor\frac{j-1}{\prod_{t=1}^{d-1} I_t}\right\rfloor I_{d-1} + 1\right) - 1\right) \prod_{t=1}^{d-2} I_t \tag{5.125}$$

$$+ \left(\left(\left\lfloor\frac{j-1}{\prod_{t=1}^{d-1} I_t}\right\rfloor + 1\right) - 1\right) \prod_{t=0}^{d-1} I_t \tag{5.126}$$

$$= 1 + (j-1) \quad \underbrace{-\left\lfloor \frac{j-1}{I_1} \right\rfloor I_1 + \left\lfloor \frac{j-1}{I_1} \right\rfloor I_1}_{=0} \tag{5.127}$$

$$\underbrace{-\left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_2 I_1 + \left\lfloor \frac{j-1}{I_1 I_2} \right\rfloor I_1 I_2}_{=0} \tag{5.128}$$

$$-\left\lfloor \frac{j-1}{I_1 I_2 I_3} \right\rfloor I_3 I_1 I_2 + ... + \left\lfloor \frac{j-1}{\prod_{t=1}^{l-1} I_t} \right\rfloor \prod_{t=1}^{l-1} I_t \tag{5.129}$$

$$\underbrace{-\left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor \prod_{t=1}^{l} I_t + \left\lfloor \frac{j-1}{\prod_{t=1}^{l} I_t} \right\rfloor \prod_{t=1}^{l} I_t}_{=0} \tag{5.130}$$

$$-\left\lfloor \frac{j-1}{\prod_{t=1}^{l+1} I_t} \right\rfloor \prod_{t=1}^{l+1} I_t + ... + \left\lfloor \frac{j-1}{\prod_{t=1}^{d-2} I_t} \right\rfloor \prod_{t=1}^{d-2} I_t \tag{5.131}$$

$$\underbrace{-\left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor I_{d-1} \prod_{t=1}^{d-2} I_t + \left\lfloor \frac{j-1}{\prod_{t=1}^{d-1} I_t} \right\rfloor \prod_{t=1}^{d-1} I_t}_{=0} \tag{5.132}$$

$$\underline{= j} \tag{5.133}$$

Second, we prove (5.119) by subdividing the resulting vector in its entries. Starting with the first entry

$$\left\langle \underline{\bar{\mathbf{I}}}_\iota \left(\underline{\bar{\mathbf{I}}}_\iota^{-1}(\ \underline{j}\ ) \right) \right\rangle_1 = \left(\left(1 + (j_1 - 1) + (j_2 - 1) I_1 + ... + (j_d - 1) \prod_{t=1}^{d-1} I_t \right) - 1 \right) - \left\lfloor \frac{\left(1 + (j_1 - 1) + ... + (j_d - 1) \prod_{t=1}^{d-1} I_t \right) - 1}{I_1} \right\rfloor I_1 + 1 \tag{5.134}$$

$$(\textit{auxiliary calculations}:) \left[\begin{array}{ll} = & \left\lfloor \frac{\left(1+(j_1-1)+...+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{I_1} \right\rfloor I_1 \\ = & \left\lfloor \frac{j_1-1}{I_1} + \frac{(j_2-1)I_1+...+(j_d-1)\prod_{t=1}^{d-1} I_t}{I_1} \right\rfloor I_1 \\ = & \left\lfloor \underbrace{\frac{j_1-1}{I_1}}_{1 \le j_1 \le I_1} + \underbrace{(j_2-1) + ... + (j_d-1) \prod_{t=2}^{d-1} I_t}_{\in \mathbb{N}_0} \right\rfloor I_1 \\ = & \left((j_2-1) + ... + (j_d-1) \prod_{t=2}^{d-1} I_t \right) I_1 \\ = & (j_2-1) I_1 + ... + (j_d-1) \prod_{t=1}^{d-1} I_t \end{array} \right] \tag{5.135}$$

$$=(j_1-1)+(j_2-1)I_1+...+(j_d-1)\prod_{t=1}^{d-1} I_t \tag{5.136}$$

$$-\left((j_2-1)I_1+...+(j_d-1)\prod_{t=1}^{d-1} I_t\right)+1 \tag{5.137}$$

$$=j_1. \tag{5.138}$$

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

$\forall l \in \{2,...,d-1\}$:

$$\left\langle \bar{\mathbf{I}}_\iota \left(\bar{\mathbf{I}}_\iota^{-1}(\ \underline{j})\ \right)\right\rangle_l = \left\lfloor \frac{\left(1+(j_1-1)+...+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l-1} I_t}\right\rfloor - \left\lfloor \frac{\left(1+(j_1-1)+...+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l} I_t}\right\rfloor I_l+1 \tag{5.139}$$

auxiliary calculations :

$$\left[\begin{array}{l}
= \left\lfloor \frac{\left(1+(j_1-1)+...+(j_d-1)\prod_{t=1}^{d-1} I_t\right)-1}{\prod_{t=1}^{l-1} I_t}\right\rfloor \\
= \left\lfloor \frac{(j_1-1)+...+(j_{l-1}-1)\prod_{t=1}^{l-2} I_t}{\prod_{t=1}^{l-1} I_t} + \frac{(j_l-1)\prod_{t=1}^{l-1} I_t+...+(j_d-1)\prod_{t=1}^{d-1} I_t}{\prod_{t=1}^{l-1} I_t}\right\rfloor \\
\quad \textit{auxiliary calculations} : \\
\quad \left[\begin{array}{rl}
0 \leq & \underbrace{(j_1-1)}_{1\leq j_1\leq I_1}+...+\underbrace{(j_{l-1}-1)}_{1\leq j_{l-1}\leq I_{l-1}}\prod_{t=1}^{l-2} I_t \\
\leq & I_1-1+(I_2-1)I_1+...+(I_{l-1}-1)\prod_{t=1}^{l-2} I_t \\
= & I_1-1+I_2I_1-I_1+...+\prod_{t=1}^{l-1} I_t-\prod_{t=1}^{l-2} I_t \\
= & \prod_{t=0}^{l-1} I_t-1 \\
\prod_{t=0}^{l-1} I_t-1 < & \prod_{t=0}^{l-1} I_t
\end{array}\right] \\
= \left\lfloor \underbrace{\frac{(j_1-1)+...+(j_{l-1}-1)\prod_{t=1}^{l-2} I_t}{\prod_{t=1}^{l-1} I_t}}_{\geq 0,\ <1} + \underbrace{(j_l-1)+...+(j_d-1)\prod_{t=l}^{d-1} I_t}_{\in \mathbb{N}_0}\right\rfloor \\
= (j_l-1)+...+(j_d-1)\prod_{t=l}^{d-1} I_t
\end{array}\right] \tag{5.140}$$

$$=\left((j_l-1)+...+(j_d-1)\prod_{t=l}^{d-1} I_t\right)-\left((j_{l+1}-1)+...+(j_d-1)\prod_{t=l+1}^{d-1} I_t\right)I_l+1 \tag{5.141}$$

$$=(j_l-1)+(j_{l+1}-1)I_l+...+(j_d-1)\prod_{t=l}^{d-1} I_t-(j_{l+1}-1)I_l-...-(j_d-1)\prod_{t=l}^{d-1} I_t+1 \tag{5.142}$$

$$=j_l \tag{5.143}$$

We finish with the last entry:

$$\left\langle \underline{\bar{\mathbf{I}}}_\iota \left(\underline{\bar{\mathbf{I}}}_\iota^{-1}(\underline{j}) \right) \right\rangle_d = \left\lfloor \frac{\left(1 + (j_1 - 1) + ... + (j_d - 1) \prod_{t=1}^{d-1} I_t\right) - 1}{\prod_{t=1}^{d-1} I_t} \right\rfloor + 1 \tag{5.144}$$

$$= \left\lfloor \underbrace{\frac{1 + (j_1 - 1) + ... + (j_{d-1} - 1) \prod_{t=1}^{d-2} I_t}{\prod_{t=1}^{d-1} I_t}}_{<1} + \underbrace{(j_d - 1)}_{\in \mathbb{N}_0} \right\rfloor + 1 \tag{5.145}$$

$$= j_d. \tag{5.146}$$

Taking these three results together, we can conclude

$$\underline{\bar{\mathbf{I}}}_\iota \left(\underline{\bar{\mathbf{I}}}_\iota^{-1}(\underline{j}) \right) = \begin{bmatrix} j_1 \\ j_2 \\ \vdots \\ j_d \end{bmatrix} \tag{5.147}$$

$$= \underline{j}\ . \tag{5.148}$$

Hence, $\underline{\bar{\mathbf{I}}}_\iota$ and $\underline{\bar{\mathbf{I}}}_\iota^{-1}$ are bijective and their respective inverse. □

Lemma 7. *Order independence of a series of interactions follows from the order independence of the interact function:*

$$_1 i_g(_1 i_g(p_j, p_k), p_{k'}) =_1 i_g(_1 i_g(p_j, p_{k'}), p_{k''})$$
$$\rightarrow \tilde{p} \ *_{_1 i_g} \ \sigma(p_1, \ldots, p_n) = \tilde{p} \ *_{_1 i_g} \ (p_1, \ldots, p_n). \tag{5.149}$$

Proof.

The proof idea is to use the "bubble sort" strategy [31] 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.

$$_1 i_g(_1 i_g(p_j, p_k), p_{k'}) = _1 i_g(_1 i_g(p_j, p_{k'}), p_{k''}) \tag{5.150}$$

$$\rightarrow \tilde{p} \ *_{_1 i_g} \ \sigma(p_1, ..., p_n) \tag{5.151}$$

$$= \underbrace{\tilde{p} \ *_{_1 i_g} \ (\sigma(p_1), ..., \sigma(p_{j-1})}_{=:p'}, \sigma(p_j), \sigma(p_{j+1}), ..., \sigma(p_n)) \tag{5.152}$$

$$= p' \ *_{_1 i_g} \ (\sigma(p_j), \sigma(p_{j+1}), ..., \sigma(p_n)) \tag{5.153}$$

$$= _1 i_g(_1 i_g(p', \sigma(p_j)), \sigma(p_{j+1})) \ *_{_1 i_g} \ (\sigma(p_{j+2}), ..., \sigma(p_n)) \tag{5.154}$$

$$= _1 i_g(_1 i_g(p', \sigma(p_{j+1})), \sigma(p_j)) \ *_{_1 i_g} \ (\sigma(p_{j+2}), ..., \sigma(p_n)) \tag{5.155}$$

$$= \tilde{p} \ *_{_1 i_g} \ (\sigma(p_1), ..., \sigma(p_{j-1}), \sigma(p_{j+1}), \sigma(p_j), \sigma(p_{j+2}), ..., \sigma(p_n)) \tag{5.156}$$

$$\text{using bubble sort} \tag{5.157}$$

$$= \tilde{p} \ *_{_1 i_g} \ (p_1, ..., p_n) \tag{5.158}$$

□

Lemma 8. *The function $copy_{\bar{g}}^{ALL}$ does not cause overlapping communications. This implies that two processes can only communicate to the same third process or each other.*

Proof.

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 can not be overlapping communication. To prove that, we need to prove

$$\begin{aligned}&\forall k, l', l'' \in \left\{1, ..., 3^d\right\} \ \forall j', j'' \in \left\{1, ..., {}^k N_{cell}^*\right\} : \\ &j' \neq j'' \rightarrow \ \beta(\gamma(k, j'), l') \neq \beta(\gamma(k, j''), l'') \ \vee \ \beta(\gamma(k, j'), l') = undef. \ ,\end{aligned} \tag{5.159}$$

where k is the number of the checkerboard-like pattern, $\gamma(k, j)$ (5.93) is an index of a reading process and $\beta(\gamma(k, j), l)$ (5.94) its l-th neighbor with 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

$$\beta(\gamma(k, j), l) = \underline{\bar{\mathbf{I}}}\iota^{-1}\left(\underline{\bar{\mathbf{I}}}\iota\left(\underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j) \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) - \underline{\bar{\mathbf{3}}}\right)\right) + \underline{\bar{\mathbf{3}}}\iota(l) - \underline{\bar{\mathbf{2}}}\right) \tag{5.160}$$

$$= \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j) \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) \ + \underline{\bar{\mathbf{3}}}\iota(l) - \underline{\bar{\mathbf{5}}}\right) \tag{5.161}$$

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

Assuming

$$j' \neq j'' \ \wedge \ \beta(\gamma(k, j'), l') = \beta(\gamma(k, j''), l'') \ \wedge \ \beta(\gamma(k, j'), l') \neq undef. \tag{5.162}$$

$\xleftrightarrow{(5.160)}$

$$\begin{gathered} j' \neq j'' \\ \wedge \ \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l') - \underline{\bar{\mathbf{5}}}\right) = \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j'') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l'') - \underline{\bar{\mathbf{5}}}\right) \\ \wedge \ \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l') - \underline{\bar{\mathbf{5}}}\right) \neq undef. \end{gathered} \tag{5.163}$$

$\xleftrightarrow{lemma\ 6}$

$$\begin{gathered} j' \neq j'' \\ \wedge \ \Bigg({}^k\underline{\bar{\mathbf{I}}}^*\iota(j') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l') - \underline{\bar{\mathbf{5}}} = {}^k\underline{\bar{\mathbf{I}}}^*\iota(j'') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l'') - \underline{\bar{\mathbf{5}}} \\ \vee \ \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l') - \underline{\bar{\mathbf{5}}}\right) = undef.\Bigg) \\ \wedge \ \underline{\bar{\mathbf{I}}}\iota^{-1}\left({}^k\underline{\bar{\mathbf{I}}}^*\iota(j') \cdot 3 + \underline{\bar{\mathbf{3}}}\iota(k) + \underline{\bar{\mathbf{3}}}\iota(l') - \underline{\bar{\mathbf{5}}}\right) \neq undef. \end{gathered} \tag{5.164}$$

$\longleftrightarrow$

$$\begin{gathered} j' \neq j'' \\ \wedge\ {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}} = {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j'') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l'') - \overline{\underline{\mathbf{5}}} \\ \wedge\ \overline{\underline{\mathbf{I}}}\iota^{-1}\left({}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}}\right) \neq \mathit{undef}. \end{gathered} \tag{5.165}$$

$\overset{\mathit{def.}\ 14}{\longleftrightarrow}$

$$\begin{gathered} j' \neq j'' \\ \wedge\ \underbrace{{}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j')}_{\in \mathbb{N}_1^d} \cdot 3 + \underbrace{\overline{\underline{\mathbf{3}}}\iota(l')}_{\in [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{3}}}]} = \underbrace{{}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j'')}_{\in \mathbb{N}_1^d} \cdot 3 + \underbrace{\overline{\underline{\mathbf{3}}}\iota(l'')}_{\in [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{3}}}]} \\ \wedge\ {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}} \in \mathbb{N}_1^d \cap [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}}] \end{gathered} \tag{5.166}$$

$\longleftrightarrow$

$$\begin{gathered} j' \neq j'' \\ \wedge\ \underbrace{{}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') - {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j'')}_{\in \mathbb{Z}^d} \cdot 3 = \underbrace{\overline{\underline{\mathbf{3}}}\iota(l'') - \overline{\underline{\mathbf{3}}}\iota(l')}_{\in [-\overline{\underline{\mathbf{2}}}, \overline{\underline{\mathbf{2}}}]} \\ \wedge\ {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}} \in \mathbb{N}_1^d \cap [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}}] \end{gathered} \tag{5.167}$$

$\overset{\mathit{def.}\ 14}{\longleftrightarrow}$

$$\begin{gathered} j' \neq j'' \\ \wedge\ \underbrace{{}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') - {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j'')}_{\in \mathbb{Z}^d} \cdot 3 = \underbrace{\overline{\underline{\mathbf{3}}}\iota(l'') - \overline{\underline{\mathbf{3}}}\iota(l')}_{\in [-\overline{\underline{\mathbf{2}}}, \overline{\underline{\mathbf{2}}}]} = \overline{\underline{\mathbf{0}}} \\ \wedge\ {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}} \in \mathbb{N}_1^d \cap [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}}] \end{gathered} \tag{5.168}$$

$\overset{\mathit{lemma}\ 6}{\longleftrightarrow}$

$$\begin{gathered} j' \neq j'' \\ \wedge\ j' = j'' \ \wedge\ l' = l'' \\ \wedge\ {}^{k}\overline{\underline{\mathbf{I}}}^{*}\iota(j') \cdot 3 + \overline{\underline{\mathbf{3}}}\iota(k) + \overline{\underline{\mathbf{3}}}\iota(l') - \overline{\underline{\mathbf{5}}} \in \mathbb{N}_1^d \cap [\overline{\underline{\mathbf{1}}}, \overline{\underline{\mathbf{I}}}] \\ \lightning\ \text{(contradiction)} \end{gathered} \tag{5.169}$$

$\longrightarrow$

$$\begin{gathered} \forall k, l', l'' \in \{1, ..., 3^d\}\ \ \forall j', j'' \in \{1, ..., {}^{k}N^{*}_{cell}\} : \\ j' \neq j'' \rightarrow\ \beta(\gamma(k, j'), l') \neq \beta(\gamma(k, j''), l'')\ \vee\ \beta(\gamma(k, j'), l') = \mathit{undef}.\ , \end{gathered} \tag{5.170}$$

$\square$

Lemma 9. *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.*

Proof.
We need to prove, after the $copy_{\tilde{g}}^{ALL}$ function, 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 $copy_{\tilde{g}}^{ALL}$ function and second, that on a process after the $copy_{\tilde{g}}^{ALL}$ function all neighbor particles of all particles of the center storage compartment are in the storage.

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

$$\gamma : \left\{1,...,3^d\right\} \times \left\{1,...,{}^k N_{cell}^*\right\} \to \{1,...,N_{cell}\} \text{ is bijective.} \tag{5.171}$$

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

$$\underline{\gamma}^* : \{1,...,3\}^d \times \mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}},\ {}^k\underline{\overline{\mathbf{I}}}^*\right] \to \mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \tag{5.172}$$

and define

$$\underline{\gamma}^*(\underline{k},\underline{j}) := \underline{k} + (\underline{j} - \underline{\overline{\mathbf{1}}}) \cdot 3. \tag{5.173}$$

One element of $\underline{\gamma}^*$ is $\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w$ and we can rewrite it by

$$\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w = 1 + (k_w - 1) + (j_w - 1)\cdot 3 = \underbrace{{}^k\underline{\overline{\mathbf{J}}}_w \iota^{-1}\left(\begin{pmatrix} k_w \\ j_w \end{pmatrix}\right)}_{\text{is bijective } (\iota \text{ is bijective})} \tag{5.174}$$

where ${}^k\underline{\overline{\mathbf{J}}}_w := \begin{pmatrix} 3 \\ {}^k I_w^* \end{pmatrix}$. 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 $\mathbb{N}^d \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right]$. $\underline{\gamma}^*$ 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

$$min\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) = 1 + (1-1)\cdot 3 = \underline{1}. \tag{5.175}$$

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

$$max\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) = k_w^{max} + \left({}^{k^{max}} I_w^* - 1\right)\cdot 3. \tag{5.176}$$

Using the definition of ${}^{k_{max}}\underline{\overline{\mathbf{I}}}^*$ (5.91) leads to

$$max\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) = k_w^{max} + \left(\left\lfloor \tfrac{1}{3}\left(I_w - k_w^{max} + 3\right)\right\rfloor - 1\right)\cdot 3 \tag{5.177}$$

We substituted

$$I_w - k_w^{max} + 3 =: \underbrace{T'}_{\in\mathbb{N}} \cdot 3 + \underbrace{T''}_{\in\{0,1,2\}} \tag{5.178}$$

and get

$$max\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) = k_w^{max} + \left(\left\lfloor \tfrac{3T'}{3} + \tfrac{T''}{3}\right\rfloor - 1\right)\cdot 3 \tag{5.179}$$

$$= k_w^{max} + (T' - 1)\cdot 3. \tag{5.180}$$

We rearrange the substitution to

$$3T' = I_w - k_w^{max} + 3 - T'' \tag{5.181}$$

and plug it in

$$max\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) = k_w^{max} + I_w - k_w^{max} + 3 - T'' - 3 \tag{5.182}$$

$$= I_w - \underbrace{T''}_{\in\{0,1,2\}} . \tag{5.183}$$

This means

$$max\left(\left\langle \underline{\gamma}^*(\underline{k},\underline{j})\right\rangle_w\right) \leq I_w. \tag{5.184}$$

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

$$\exists k_w \in \{1,2,3\} : T'' = 0. \tag{5.185}$$

From $T'' = 0$ follows that

$$\frac{3T' + T''}{3} = \frac{I_w + 3 - k_w^{max}}{3} = n \in \mathbb{N}. \tag{5.186}$$

$\longrightarrow$

$$n + \frac{k_w^{max} - 1}{3} = \frac{I_w + 2}{3} \tag{5.187}$$

Taking the *floor* on both sides lead to

$$\left\lfloor n + \underbrace{\frac{k_w^{max} - 1}{3}}_{\in\{0,1,2\}} \right\rfloor = \left\lfloor \frac{I_w + 2}{3} \right\rfloor \tag{5.188}$$

$\longrightarrow$

$$n = \left\lfloor \frac{I_w + 2}{3} \right\rfloor . \tag{5.189}$$

Inserting this into (5.187) lead to

$$k_w = ((I_w + 3) - 1) - \left\lfloor \tfrac{(I_w+3)-1}{3} \right\rfloor \cdot 3 + 1 = \left\langle \underline{\overline{\mathbf{J}}}\iota(I_w + 3)\right\rangle_1 \in \{1,2,3\} \tag{5.190}$$

where $\underline{\overline{\mathbf{J}}} = (3, ...)^{\mathbf{T}}$. We can follow,

$$\forall w \in \{1, ..., d\} : \left\langle \underline{\gamma}^*\right\rangle_w \text{ is bijective} \tag{5.191}$$

$\longrightarrow$

$$\underline{\gamma}^* \text{ is bijective} \tag{5.192}$$

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

$$\underline{k} = {}^{\overline{\mathbf{3}}}\iota(k), \qquad \underline{j} = {}^{k\overline{\mathbf{I}}^*}\iota(j). \tag{5.193}$$

We know that ι is bijective (lemma 6, def. 14) and therefore

$$k \in \{1, ..., 3^d\}, \qquad j \in \ N^d \cap \left[\underline{\overline{\mathbf{1}}}, {}^k\underline{\overline{\mathbf{I}}}^*\right]. \tag{5.194}$$

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

$$\underline{\overline{\mathbf{I}}}\iota^{-1}\left(\underline{\gamma}^*\left(\underline{\overline{\mathbf{3}}}\iota(k), {}^k\underline{\overline{\mathbf{I}}}^*\iota(j)\right)\right) = \underline{\overline{\mathbf{I}}}\iota^{-1}\left(\underline{\overline{\mathbf{3}}}\iota(k) + \left({}^k\underline{\overline{\mathbf{I}}}^*\iota(j) - \underline{\overline{\mathbf{1}}}\right)\cdot 3\right) \tag{5.195}$$

$$= \gamma(k, j) \tag{5.196}$$

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

To second, it remains to be proven that the neighbor particles of all particles of the center storage compartment are in the storage. The function $copy_{(\tilde{g},\mathcal{P},w)}(\underline{\mathbf{p}}, l)$ (5.103) copies the central $\left(\frac{3^d+1}{2}\right)$ storage compartment form the $\beta(w, l)$-th process to the l-th storage compartment on the w-th process. We start from the vectorial index view to find the corresponding cell/process where the particles are for the l-th storage compartment on the w-th process. The vectorial index of the w-th process is $\underline{\overline{\mathbf{I}}}\iota(w)$ and the vectorial index of the l-th storage compartment is $\underline{\overline{\mathbf{3}}}\iota(l) \in \{1, 2, 3\}$. The storage center compartment $l = \left(\frac{3^d+1}{2}\right)$ corresponds to the cell belonging to the process. The rest of the storage compartments should represent the surrounding cells. Therefore, to account for this, we need to shift the storage compartment index by -2 in all dimensions. Hence, the vectorial index of the corresponding process for the l-th storage compartment of the w-th process is

$$\underline{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{3}}}\iota(l) - \underline{\overline{\mathbf{2}}}. \tag{5.197}$$

Transforming the vectorial index to a scalar index results in

$$\underline{\overline{\mathbf{I}}}\iota^{-1}\left(\underline{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{3}}}\iota(l) - \underline{\overline{\mathbf{2}}}\right) = \beta(w, l). \tag{5.198}$$

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

$$\left\{\underline{\overline{\mathbf{I}}}\iota(\beta(w, l)) : l \in \{1, ..., 3^d\}\right\} = \left[\underline{\overline{\mathbf{I}}}\iota(w) - \underline{\overline{\mathbf{1}}},\ \underline{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{1}}}\right] \cap \left[\underline{\overline{\mathbf{1}}}, \underline{\overline{\mathbf{I}}}\right] \cap \mathbb{N}^d. \tag{5.199}$$

Per definition (5.96) the particle that belong to the w-th cell are

$$p_j = (..., x_j, ...) \in \mathbf{p}_w^1 : \ \underline{\overline{\mathbf{I}}}\iota(w) = \left\lfloor \frac{1}{r_c}(\underline{x}_j - \underline{D}_{min})\right\rfloor + \underline{\overline{\mathbf{1}}}. \tag{5.200}$$

$\longrightarrow$

$$p_j \in \mathbf{p}_w^1 : \ \underline{\overline{\mathbf{I}}}\iota(w) - \underline{\overline{\mathbf{1}}} = \left\lfloor \frac{1}{r_c}(\underline{x}_j - \underline{D}_{min})\right\rfloor. \tag{5.201}$$

$\longrightarrow$

$$p_j \in \mathbf{p}_w^1 : \ \frac{1}{r_c}(\underline{x}_j - \underline{D}_{min}) \in \left[\underline{\overline{\mathbf{I}}}\iota(w) - \underline{\overline{\mathbf{1}}},\ \underline{\overline{\mathbf{I}}}\iota(w) - \underline{\overline{\mathbf{1}}} + \underline{\overline{\mathbf{1}}}\right). \tag{5.202}$$

$\longrightarrow$

$$p_j \in \mathbf{p}^1_w : \ \underline{x}_j \in \left[\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min},\ \overline{\mathbf{I}}\iota(w) \cdot r_c + \underline{D}_{min}\right). \tag{5.203}$$

Taking all cells/compartments (5.199) of the storage of the w-process leads to

$$p_j \in \bigcirc_{l=1}^{3^d} \langle^{[\mathrm{PROC}w]}\underline{\mathbf{p}}^1\rangle_l : \\ \underline{x}_j \in \left[\left(\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{1}}\right) - \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min},\ \left(\overline{\mathbf{I}}\iota(w) + \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min}\right) \\ \cap [\underline{D}_{min}, \underline{D}_{max}), \tag{5.204}$$

where $\left[\overline{\mathbf{1}}, \overline{\mathbf{I}}\right]$ translates to $[\underline{D}_{min}, \underline{D}_{max})$ (5.89).

$\longrightarrow$

$$p_j \in \bigcirc_{l=1}^{3^d} \langle^{[\mathrm{PROC}w]}\underline{\mathbf{p}}^1\rangle_l : \\ \underline{x}_j \in \left[\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{2}}\right) \cdot r_c + \underline{D}_{min},\ \left(\overline{\mathbf{I}}\iota(w) + \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min}\right) \cap [\underline{D}_{min}, \underline{D}_{max}). \tag{5.205}$$

It remains to be proven that the neighbor particles of all particles of the central storage compartment of the w-th process are in the storage. Hence, the domain area covered by the storage includes the area covered by the neighborhood function u. Be $p_k \in \mathbf{p}^1$ and $p_j \in \left\langle \mathbf{p}^1 \right\rangle_{u(g,\mathbf{p}^1,k)}$ (5.80) then

$$|\underline{x}_k - \underline{x}_j| \leq r_c \tag{5.206}$$

$\longrightarrow$

$$\underline{x}_j \in \left[\underline{x}_k - \overline{\mathbf{1}} \cdot r_c,\ \underline{x}_k + \overline{\mathbf{1}} \cdot r_c\right] \tag{5.207}$$

Be now $p_k \in \mathbf{p}^1_w$ then we know (5.203)

$$\underline{x}_k \in \left[\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min},\ \overline{\mathbf{I}}\iota(w) \cdot r_c + \underline{D}_{min}\right). \tag{5.208}$$

$\longrightarrow$

$$\underline{x}_j \in \left[\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min} - \overline{\mathbf{1}} \cdot r_c,\ \overline{\mathbf{I}}\iota(w) \cdot r_c + \underline{D}_{min} + \overline{\mathbf{1}} \cdot r_c\right) \\ \cap [\underline{D}_{min}, \underline{D}_{max}). \tag{5.209}$$

$\longrightarrow$

$$\underline{x}_j \in \left[\left(\overline{\mathbf{I}}\iota(w) - \overline{\mathbf{2}}\right) \cdot r_c + \underline{D}_{min},\ \left(\overline{\mathbf{I}}\iota(w) + \overline{\mathbf{1}}\right) \cdot r_c + \underline{D}_{min}\right) \cap [\underline{D}_{min}, \underline{D}_{max}). \tag{5.210}$$

This is the same as (5.205), which means that the neighbor particles of all particles of the center storage compartment of the w-th process are all in the storage of the w-th process, and this is true for all processes. $\square$

Lemma 10. *The functions $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ 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,*

$$\forall w \in \{1,\dots,N_{cell}\} \forall p_j^1 \in \left\langle \left\langle \mathcal{P}^1 \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} : p_j^2 := \left\langle \left\langle \left\langle step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right)\right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j$$
$$\longrightarrow \alpha = {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{3}}}\right) \in \{1,\dots,3^d\}. \quad (5.211)$$

Proof.
We need to prove that the functions $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ places for each process the particles from the center storage compartments only to the other storage compartments of the same processes.

All particles have a position $\underline{x} \in \mathbb{R}^d$, hence, $p_j^1 = (..., \underline{x}_j^1, ...)$ and $p_j^2 = (..., \underline{x}_j^2, ...)$. The condition (5.82) restricts the movement of the particles to be smaller than r_c. Meaning for $\underline{\Delta x} := \underline{x}_j^2 - \underline{x}_j^1$

$$|\underline{\Delta x}| \leq r_c. \quad (5.212)$$

$\xrightarrow{def.\ 5}$

$$|\underline{\Delta x}| = \left| \begin{pmatrix} \Delta x_1 \\ \vdots \\ \Delta x_d \end{pmatrix} \right| = \sqrt{\Delta x_1^2 + ... + \Delta x_d^2} \leq r_c \quad (5.213)$$

$\longrightarrow$

$$\forall k \in \{1, ..., d\} : r_c \leq \Delta x_k \leq r_c \quad (5.214)$$

$\longrightarrow$

$$\forall k \in \{1, ..., d\} : \Delta x_k \in [-r_c, r_c] \quad (5.215)$$

Using this, we can rewrite α from

$$\alpha = {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{3}}}\right) \quad (5.216)$$

to

$$\alpha = {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^1 + \underline{\Delta x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(w) + \underline{\overline{\mathbf{3}}}\right). \quad (5.217)$$

We put the index for the dimension as pre-sub-script.

$$\alpha = {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\begin{pmatrix} \left\lfloor \frac{1}{r_c}({}_1x_j^1 + {}_1\Delta x - {}_1D_{min})\right\rfloor - \left\langle {}^{\overline{\mathbf{I}}}\iota(w)\right\rangle_1 + 3 \\ \vdots \\ \left\lfloor \frac{1}{r_c}({}_dx_j^1 + {}_d\Delta x - {}_dD_{min})\right\rfloor - \left\langle {}^{\overline{\mathbf{I}}}\iota(w)\right\rangle_d + 3 \end{pmatrix}\right) \quad (5.218)$$

We know from the condition (5.96) that

$$\forall p_j^1 \in \mathbf{p}^1 : p_j^1 \in \mathbf{p}_w^1 \text{ with } \quad w = {}^{\overline{\mathbf{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^1 - \underline{D}_{min})\right\rfloor + \underline{\overline{\mathbf{1}}}\right). \quad (5.219)$$

Taking this into α we get

$$\alpha = \overline{\mathbf{3}}\iota^{-1}\left(\underbrace{\begin{pmatrix} \underbrace{\left\lfloor \frac{{}_1x_j^1 - {}_1D_{min}}{r_c} + \underbrace{\frac{{}_1\Delta x}{r_c}}_{\in[-1,1]} \right\rfloor}_{\in\left\{\left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_1 - 2,\ \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_1 - 1,\ \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_1\right\}} & - \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_1 + 3 \\ \vdots & \\ \underbrace{\left\lfloor \frac{{}_dx_j^1 - {}_dD_{min}}{r_c} + \underbrace{\frac{{}_d\Delta x}{r_c}}_{\in[-1,1]} \right\rfloor}_{\in\left\{\left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_d - 2,\ \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_d - 1,\ \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_d\right\}} & - \left\langle \overline{\mathbf{I}}\iota(w)\right\rangle_d + 3 \end{pmatrix}}_{\in\mathbb{N}\cap\left[\overline{\mathbf{1}},\overline{\mathbf{3}}\right]}\right) \tag{5.220}$$

$\longrightarrow$

$$\underline{\alpha \in \{1, ..., 3^d\}} \tag{5.221}$$

Hence, under the condition (5.96), no particle leaves the storage compartments of its process through the function $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$. □

Lemma 11. *The function $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ 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. Be*

$$\underline{w} = (w_1, \ldots, w_d)^\top := \overline{\mathbf{I}}\iota(w) \tag{5.222}$$

$$\underline{\alpha} = (\alpha_1, \ldots, \alpha_d)^\top := \left\lfloor \frac{1}{r_c}(\underline{x}_j^2 - \underline{D}_{\min}) \right\rfloor - \underline{w} + \overline{\mathbf{3}}, \tag{5.223}$$

then

$$\begin{aligned} \forall w \in \{1, \ldots, N_{\text{cell}}\} \quad \forall p_j^2 := \left\langle \left\langle \left\langle step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right)\right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right\rangle_j : \\ (k \in \{1, \ldots, d\} \wedge w_k = 1) \ \rightarrow \ \alpha_k \in \{2, 3\} \\ \wedge (k \in \{1, \ldots, d\} \wedge w_k = I_k) \ \rightarrow \ \alpha_k \in \{1, 2\}. \end{aligned} \tag{5.224}$$

Proof.

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

$$\exists k \in \{1, ..., d\} : w_k = 1 \ \wedge \alpha_k = 1 \tag{5.225}$$

then

$$\alpha_k = 1 = \left\lfloor \frac{1}{r_c}({}_kx_j^2 - {}_kD_{min}) \right\rfloor - 1 + 3. \tag{5.226}$$

$\longrightarrow$

$$-1 = \left\lfloor \frac{1}{r_c}({}_kx_j^2 - {}_kD_{min}) \right\rfloor \tag{5.227}$$

$\longrightarrow$

$$\tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{min}) = [-1, 0) \tag{5.228}$$

$\longrightarrow$

$${}_k x_j^2 = [-r_c + {}_k D_{min}, 0) \quad \lightning \tag{5.229}$$

This contradicts the condition that no particle leaves the domain (5.81). Hence, in this case $\alpha_k \in \{2, 3\}$. Now we prove the second statement with $w_k = I_k$. Amusing

$$\exists k \in \{1, ..., d\} : w_k = I_k \ \wedge \alpha_k = 3 \tag{5.230}$$

then

$$\alpha_k = 3 = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{min}) \right\rfloor - I_k + 3. \tag{5.231}$$

$\longrightarrow$

$$I_k = \left\lfloor \tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{min}) \right\rfloor \tag{5.232}$$

$\longrightarrow$

$$\tfrac{1}{r_c}({}_k x_j^2 - {}_k D_{min}) = [I_k, I_k + 1) \tag{5.233}$$

$\longrightarrow$

$${}_k x_j^2 = [I_k r_c + {}_k D_{min}, I_k r_c + {}_k D_{min} + r_c) \quad \lightning \tag{5.234}$$

$\xrightarrow{def.\ \underline{\bar{\mathbf{I}}}\ (5.89)}$

$${}_k x_j^2 > {}_k D_{max}) \quad \lightning \tag{5.235}$$

This contradicts the condition that no particle leaves the domain (5.81). 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 which represent cells inside the domain. □

Lemma 12. $\overline{\mathcal{P}}^1 := step^{ALL}_{\mathcal{G}^1}\left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^1)\right)$ *does not necessarily fulfill the condition in (5.96) to reiterate* $step^{ALL}_{\mathcal{G}^2}\left(copy^{ALL}_{\tilde{g}}\left(\overline{\mathcal{P}}^1\right)\right)$. *To avoid that* $interact_g$ *misses interactions due to incomplete neighborhoods, the particles in* $\overline{\mathcal{P}}^1$ *therefore need to be redistributed such that (5.96) is fulfilled. The decomposition of the initial permuted particle tuple is stored in the center storage compartment of the processes. Hence, (5.96) can be rewritten for all state transition steps as*

$$\forall p_j^t \in \bigcirc_{v=1}^{N_{\text{cell}}} \left\langle \left\langle \mathcal{P}^t \right\rangle_v \right\rangle_{\frac{3^d+1}{2}} : p_j^t \in \left\langle \left\langle \mathcal{P}^t \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}, \tag{5.236}$$

where

$$w = \underline{\bar{\mathbf{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}_j^t - \underline{D}_{\min}) \right\rfloor + \underline{\bar{\mathbf{1}}}\right). \tag{5.237}$$

This is achieved by the two functions $dist^{ALL}_{(\tilde{g},\mathcal{G}^1)}$ *and* $collect^{ALL}_{\tilde{g}}$.

Proof.
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 (5.236), (5.237) is true for $\mathcal{P}^1$ by definition (5.96).

Regarding the induction step, we need to prove that if $\mathcal{P}^t$ fulfills the condition (5.236), (5.237) then $\mathcal{P}^{t+1} = collect_{\tilde{g}}^{ALL}\left(dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}\left(step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^t)\right)\right)\right)$ fulfills also the condition.

$dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ (5.112) 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.

$collect_{\tilde{g}}^{ALL}$ (5.115) copies for each process its particles from the other processes corresponding storage compartments. It uses the same strategy as the function $copy_{\tilde{g}}^{ALL}$ to avoid overlapping conditions, hence, lemma 8 is also valid for the $collect_{\tilde{g}}^{ALL}$ function. The $collect_{\tilde{g}}^{ALL}$ function takes the particles from the ${}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right)$-th storage compartment of the $\beta(w,l)$-th process and stores it in the center storage compartment of the w-th 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 (5.236), (5.237). We choose without restricting generality a particle p from the ${}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right)$-th storage compartment of the $\beta(w,l)$-th process. Hence,

$$p = (...,x,...) \in \left\langle \left\langle step_{\mathcal{G}^1}^{ALL}\left(copy_{\tilde{g}}^{ALL}(\mathcal{P}^t)\right)\right\rangle_{\beta(w,l)}\right\rangle_{{}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right)}. \tag{5.238}$$

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

$$\underbrace{\alpha}_{\stackrel{!}{=}{}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right)} := {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(\underbrace{j}_{\stackrel{!}{=}\beta(w,l)}) + \underline{\overline{\mathbf{3}}}\right), \tag{5.239}$$

where α is set to ${}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right)$ the index of the storage compartment from that the particles are collected, and j is set to $\beta(w,l)$ the index of the process from which is collected. Hence, our particle is on that storage compartment of that process and gets distributed by the mechanism of the $dist_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ function. We put the indices in and transform

$${}^{\overline{\mathbf{3}}}\iota^{-1}\left(\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l)\right) = {}^{\overline{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(\beta(w,l)) + \underline{\overline{\mathbf{3}}}\right), \tag{5.240}$$

$$\xrightarrow{lemma\ 6}$$

$$\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l) = \left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(\beta(w,l)) + \underline{\overline{\mathbf{3}}}, \tag{5.241}$$

$$\xrightarrow{def.\ of\ \beta\ (5.94)}$$

$$\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l) = \left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota\left({}^{\overline{\mathbf{I}}}\iota^{-1}\left({}^{\overline{\mathbf{I}}}\iota(w) + {}^{\overline{\mathbf{3}}}\iota(l) - \underline{\overline{\mathbf{2}}}\right)\right) + \underline{\overline{\mathbf{3}}}, \tag{5.242}$$

$$\xrightarrow{lemma\ 6}$$

$$\underline{\overline{\mathbf{4}}} - {}^{\overline{\mathbf{3}}}\iota(l) = \left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min})\right\rfloor - {}^{\overline{\mathbf{I}}}\iota(w) - {}^{\overline{\mathbf{3}}}\iota(l) + \underline{\overline{\mathbf{2}}} + \underline{\overline{\mathbf{3}}}, \tag{5.243}$$

$\longrightarrow$

$$^{\overline{\mathbf{I}}}\iota(w) = \left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min}) \right\rfloor + \overline{\mathbf{1}}, \tag{5.244}$$

$\xrightarrow{lemma\ 6}$

$$w = {}^{\overline{\mathbf{I}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x} - \underline{D}_{min}) \right\rfloor + \overline{\mathbf{1}} \right), \tag{5.245}$$

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 w-th process

$$p \in \left\langle \left\langle collect^{ALL}_{\tilde{g}} \left(dist^{ALL}_{(\tilde{g},\mathcal{G}^1)} \left(step^{ALL}_{\mathcal{G}^1} \left(copy^{ALL}_{\tilde{g}}(\mathcal{P}^t) \right) \right) \right) \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}. \tag{5.246}$$

Hence,

$$p \in \left\langle \left\langle \mathcal{P}^{t+1} \right\rangle_w \right\rangle_{\frac{3^d+1}{2}}. \tag{5.247}$$

fulfills the condition (5.236), (5.237). □

5.3.3 Parallelizability

Theorem 2 (Parallelizability of particle methods on distributed-memory computers)**.** *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 5.4 computes the same results, except for particle ordering, as the underlying sequential particle method in figure 3.1. Hence,*

$$S\left([g^1, \mathbf{p}^1]\right) = \left[\left\langle \mathcal{G}^T \right\rangle_1 \ , \sigma^* \left(\bigcirc_{w=1}^{N_{cell}} \left\langle \left\langle \mathcal{P}^T \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right) \right] \tag{5.248}$$

where

$$\left[\mathcal{G}^T, \mathcal{P}^T\right] = \tilde{S}\left(\left[\mathcal{G}^1, \mathcal{P}^1\right]\right). \tag{5.249}$$

Proof.
We prove that each sequential state transition step s (3.15) is equivalent to each parallel state transition step $\tilde{s}$ (5.116) and that both algorithms terminate after the same number of state transitions. The sequential state transition step was defined as follows:

$$s\left([g, \mathbf{p}]\right) := \left[\overset{\circ}{e}(\overline{g}), \overline{\mathbf{p}} \right] \quad \text{with } [\overline{g}, \overline{\mathbf{p}}] = \epsilon^{\mathrm{N}}\left(g, \iota^{\mathrm{N}\times\mathrm{U}}\left([g, \mathbf{p}]\right)\right). \tag{3.15}$$

Under the condition the interact function (5.84) and the neighborhood function (5.87) are independent of previous interactions, we can rewrite the third interact subfunction $\iota^{\mathrm{N}\times\mathrm{U}}$ for pull interaction particle methods (5.83):

$$\iota^{\mathrm{N}\times\mathrm{U}}\left([g, \mathbf{p}]\right) = \left(p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ldots, \ p_{|\mathbf{p}|} *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)} \right) \tag{5.33}$$

We also know that if the evolve function does not change the global variable (5.88), we can write the state transition step as:

$$s\left([g, \mathbf{p}]\right) := \left[\overset{\circ}{e}(g), \ {}_2\epsilon^{\mathrm{N}} \left(\begin{pmatrix} p_1 *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)} \\ \vdots \\ p_{|\mathbf{p}|} *_{1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)} \end{pmatrix}^{\top} \right) \right]. \tag{5.250}$$

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

$$\begin{pmatrix} g^2 \\ \vdots \\ g^2 \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(g^1\right) \\ \vdots \\ \overset{\circ}{e}\left(g^1\right) \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(\left\langle \mathcal{G}^1 \right\rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\left\langle \mathcal{G}^1 \right\rangle_{N_{\text{cell}}}\right) \end{pmatrix}^\top = \begin{pmatrix} \left\langle \mathcal{G}^2 \right\rangle_1 \\ \vdots \\ \left\langle \mathcal{G}^2 \right\rangle_{N_{\text{cell}}} \end{pmatrix}^\top = \mathcal{G}^2. \tag{5.251}$$

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

$$\begin{pmatrix} g^t \\ \vdots \\ g^t \end{pmatrix}^\top = \mathcal{G}^t \tag{5.252}$$

and get

$$\begin{pmatrix} g^{t+1} \\ \vdots \\ g^{t+1} \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(g^t\right) \\ \vdots \\ \overset{\circ}{e}\left(g^t\right) \end{pmatrix}^\top = \begin{pmatrix} \overset{\circ}{e}\left(\left\langle \mathcal{G}^t \right\rangle_1\right) \\ \vdots \\ \overset{\circ}{e}\left(\left\langle \mathcal{G}^t \right\rangle_{N_{\text{cell}}}\right) \end{pmatrix}^\top = \begin{pmatrix} \left\langle \mathcal{G}^{t+1} \right\rangle_1 \\ \vdots \\ \left\langle \mathcal{G}^{t+1} \right\rangle_{N_{\text{cell}}} \end{pmatrix}^\top = \mathcal{G}^{t+1}. \tag{5.253}$$

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 $\left\langle \mathcal{G}^t \right\rangle_1$. From this and $g^t = \left\langle \mathcal{G}^t \right\rangle_1$ follows

$$\underline{{}_1S\left(\left[g^1, \mathbf{p}^1\right]\right) = g^T = \left\langle \mathcal{G}^T \right\rangle_1 = \left\langle {}_1\tilde{S}\left(\left[\mathcal{G}^1, \mathcal{P}^1\right]\right)\right\rangle_1}. \tag{5.254}$$

Since the stop function f is for both schemes identical, this proves the both schemes stop after the same number of states.

We now prove the equivalence of the particle tuple part of the state transitions

$${}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\begin{pmatrix} p_1^1 *_{1i_{g^1}} \left\langle \mathbf{p}^1 \right\rangle_{u(g^1, \mathbf{p}^1, 1)} \\ \vdots \\ p_{|\mathbf{p}^1|}^1 *_{1i_{g^1}} \left\langle \mathbf{p}^1 \right\rangle_{u(g^1, \mathbf{p}^1, |\mathbf{p}^1|)} \end{pmatrix}^\top\right). \tag{5.255}$$

The interact function is order-independent (5.86), and the permutation $\pi(\mathbf{p}^1)$ (5.95) of the initial particle tuple sorts the particles into cells. The neighborhood function is restricted to not using indices (5.80). Hence, for a permuted particle tuple, the neighborhood function returns the same result as for the un-permuted particle tuple, with the same permutation also applied to the result. This leads to

$${}_2s\left(\left[g^1, \mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\begin{pmatrix} p_{\pi(1)}^1 *_{1i_{g^1}} \left\langle \pi(\mathbf{p}^1) \right\rangle_{u(g^1, \pi(\mathbf{p}^1), \pi(1))} \\ \vdots \\ p_{\pi(|\mathbf{p}^1|)}^1 *_{1i_{g^1}} \left\langle \pi(\mathbf{p}^1) \right\rangle_{u(g^1, \pi(\mathbf{p}^1), \pi(|\mathbf{p}^1|))} \end{pmatrix}^\top\right)\right). \tag{5.256}$$

From the construction of the initial particle storages of all processes $\mathcal{P}^1$ (5.96)-(5.100), as well as Lemmata 8 and 9, we derive

$$
{}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\begin{pmatrix} {}^{w}p_1^1 *_{1 i_{g^1}} {}^{w}\mathbf{q}^1{}_{u(g^1,{}^{w}\mathbf{q}^1,{}^{w}z+1)} \\ \vdots \\ {}^{w}p_{{}^{w}n}^1 *_{1 i_{g^1}} {}^{w}\mathbf{q}^1{}_{u(g^1,{}^{w}\mathbf{q}^1,{}^{w}z+{}^{w}n)} \end{pmatrix}^{\top}\right)\right), \tag{5.257}
$$

where the tuple of all particles in the storage of the w-th process is

$$
{}^{w}\mathbf{q}^1 := \bigcirc_{l=1}^{3^d}\left\langle\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right\rangle_l, \tag{5.258}
$$

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

$$
{}^{w}z := \sum_{l=1}^{\frac{3^d+1}{2}-1}\left|\left\langle\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right\rangle_l\right|, \tag{5.259}
$$

the number of particles in the central storage compartment is

$$
{}^{w}n = \left|\left\langle\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right|, \tag{5.260}
$$

and the particles in the central storage compartment are

$$
\left({}^{w}p_1^1,\ldots,{}^{w}p_{{}^{w}n}^1\right) = \left\langle\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}. \tag{5.261}
$$

The interactions of the particles with their neighbors in (5.257) resembles the $interaction_g$ function (5.3.1)

$$
{}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = {}_2\epsilon^{\mathrm{N}}\left(\pi^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}} interaction_{g^1}\left(\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{5.262}
$$

The evolution function cannot change the global variable (condition (5.88)). Hence, the second evolution subfunction ϵ^{N} can also not change the global variable and is, therefore, independent of the ordering of the particles. Since the evolve function e can create or destroy particles, the permutation π^{-1} can not 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

$$
{}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left({}_2\epsilon^{\mathrm{N}}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}} interaction_{g^1}\left(\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{5.263}
$$

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

$$
{}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}} {}_2\epsilon^{\mathrm{N}}\left(interaction_{g^1}\left(\left\langle copy_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right)\right). \tag{5.264}
$$

The combination of ${}_2\epsilon^{\mathrm{N}}$ and $\mathit{interaction}_g$ is the same as the $\mathit{step}_{(\tilde{g},g)}$ function (5.107) at the center storage compartment

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle \mathit{step}_{(\tilde{g},g^1)}\left(\left\langle \mathit{copy}_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right\rangle_w\right)\right\rangle_{\frac{3^d+1}{2}}\right). \tag{5.265} $$

The function $\mathit{step}_{\mathcal{G}^1}^{ALL}$ (5.108) calculates the step for all processes, leading to

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle \mathit{step}_{\mathcal{G}^1}^{ALL}\left(\mathit{copy}_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{5.266} $$

$\overline{\mathcal{P}}^1 := \mathit{step}_{\mathcal{G}^1}^{ALL}\left(\mathit{copy}_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)$ does not necessarily fulfill the condition in (5.96) to iterate $\mathit{step}_{\mathcal{G}^2}^{ALL}\left(\mathit{copy}_{\tilde{g}}^{ALL}\left(\overline{\mathcal{P}}^1\right)\right)$. For $\mathit{interact}_g$ to not miss interactions due to incomplete neighborhoods, the particles in $\overline{\mathcal{P}}^1$ need to be redistributed such that the condition in (5.96) is fulfilled. This is achieved by the functions $\mathit{dist}_{(\tilde{g},\mathcal{G}^1)}^{ALL}$ (5.112) and $\mathit{collect}_{\tilde{g}}^{ALL}$ (5.115), as proven in the Lemmata 10, 11, and 12. Hence,

$$ \begin{aligned} &{}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) \\ &\quad= \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle \mathit{collect}_{\tilde{g}}^{ALL}\left(\mathit{dist}_{(\tilde{g},\mathcal{G}^1)}^{ALL}\left(\mathit{step}_{\mathcal{G}^1}^{ALL}\left(\mathit{copy}_{\tilde{g}}^{ALL}(\mathcal{P}^1)\right)\right)\right)\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \end{aligned} \tag{5.267} $$

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

$$ {}_2s\left(\left[g^1,\mathbf{p}^1\right]\right) = \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle\underbrace{{}_2\tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^1,\mathcal{P}^1\right]\right)}_{=:\mathcal{P}^2}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \tag{5.268} $$

$$ \left[g^2,\mathbf{p}^2\right] = \tilde{\pi}'^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle\mathcal{P}^2\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{5.269} $$

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

$$ \left[g^t,\mathbf{p}^t\right] = \tilde{\pi}''^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle\mathcal{P}^t\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{5.270} $$

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 (5.96) (or (5.257), (5.258)). 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 10, 11, and 12 we derive that

$$ {}_2s\left(\left[g^t,\mathbf{p}^t\right]\right) = \tilde{\pi}'''^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle\underbrace{{}_2\tilde{s}_{\tilde{g}}\left(\left[\mathcal{G}^t,\mathcal{P}^t\right]\right)}_{=:\mathcal{P}^{t+1}}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right), \tag{5.271} $$

hence,

$$ \left[g^{t+1},\mathbf{p}^{t+1}\right] = \tilde{\pi}'''^{-1}\left(\bigcirc_{w=1}^{N_{\mathrm{cell}}}\left\langle\left\langle\mathcal{P}^{t+1}\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right). \tag{5.272} $$

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

$$S\left(\left[g^1, \mathbf{p}^1\right]\right) = \left[\left\langle \mathcal{G}^T \right\rangle_1 \ , \sigma^* \left(\bigcirc_{w=1}^{N_{\text{cell}}} \left\langle \left\langle \mathcal{P}^T \right\rangle_w \right\rangle_{\frac{3^d+1}{2}} \right)\right], \tag{5.273}$$

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. □

5.3.4 Bounds on Time Complexity and Parallel Scalability

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 each particle. We further assume that the algorithm terminates in a finite time. Hence, an upper bound exists for all functions' time complexities. An upper bound on the time complexity of the interact function $\tau_{i(g,p',p'')}$ is

$$\forall g \in \{g^1, \ldots, g^T\}, p', p'' \in \bigcirc_{w=1}^{T} \mathbf{p}^w \ \exists \tilde{g} \in \{g^1, \ldots, g^T\}, \tilde{p}', \tilde{p}'' \in \bigcirc_{w=1}^{T} \mathbf{p}^w : \\ \tau_{i(g,p',p'')} \leq \tau_{i(\tilde{g},\tilde{p}',\tilde{p}'')} =: \tau_i. \tag{5.274}$$

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

$$\forall g \in \{g^1, \ldots, g^T\}, p \in \bigcirc_{w=1}^{T} \mathbf{p}^w \ \exists \tilde{g} \in \{g^1, \ldots, g^T\}, \tilde{p} \in \bigcirc_{w=1}^{T} \mathbf{p}^w : \\ \tau_{e(g,p)} \leq \tau_{e(\tilde{g},\tilde{p})} =: \tau_e. \tag{5.275}$$

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

$$\forall g \in \{g^1, \ldots, g^T\} \ \exists \tilde{g} \in \{g^1, \ldots, g^T\} : \quad \tau_{\overset{\circ}{e}(g)} \leq \tau_{\overset{\circ}{e}(\tilde{g})} =: \tau_{\overset{\circ}{e}}. \tag{5.276}$$

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

$$\forall g \in \{g^1, \ldots, g^T\} \ \exists \tilde{g} \in \{g^1, \ldots, g^T\} : \quad \tau_{f(g)} \leq \tau_{f(\tilde{g})} =: \tau_f. \tag{5.277}$$

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

$$\forall g \in \{g^1, \ldots, g^T\}, \mathbf{p} \in \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}, j \in \{1, \ldots, |\mathbf{p}|\} \\ \exists \tilde{g} \in \{g^1, \ldots, g^T\}, \tilde{\mathbf{p}} \in \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}, \tilde{j} \in \{1, \ldots, |\tilde{\mathbf{p}}|\} : \\ \tau_{u([g,\mathbf{p}],j)} \leq \tau_{u([\tilde{g},\tilde{\mathbf{p}}],\tilde{j})} =: \tau_u. \tag{5.278}$$

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

$$\forall g \in \{g^1,\dots,g^T\}, \mathbf{p} \in \{\mathbf{p}^1,\dots,\mathbf{p}^T\}, j \in \{1,\dots,|\mathbf{p}|\} \\ \exists \tilde{g} \in \{g^1,\dots,g^T\}, \tilde{\mathbf{p}} \in \{\mathbf{p}^1,\dots,\mathbf{p}^T\}, \tilde{j} \in \{1,\dots,|\tilde{\mathbf{p}}|\} : \\ |u([g,\mathbf{p}],j)| \leq |u([\tilde{g},\tilde{\mathbf{p}}],\tilde{j})| =: \varsigma_u. \quad (5.279)$$

The time complexity of calculating $\underline{\bar{\mathbf{I}}}\iota$ and $\underline{\bar{\mathbf{I}}}\iota^{-1}$ is in $\mathcal{O}(d)$. 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

$$\forall\ \underline{\bar{\mathbf{I}}} \in \mathbb{N}^d_{>0}, \forall t \in \{1,\dots,N_{\text{cell}}\},\ l \in \{1,\dots,3^d\}\ \exists C_\beta \in \mathbb{R} :\ \tau_{\beta(t,l)} \leq C_\beta d. \quad (5.280)$$

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

$$\forall\ \underline{\bar{\mathbf{I}}} \in \mathbb{N}^d_{>0}, \forall k \in \{1,\dots,3^d\},\ j \in \{1,\dots,{}^kN^*_{\text{cell}}\}\ \exists C_\gamma \in \mathbb{R} :\ \tau_{\gamma(t,l)} \leq C_\gamma d. \quad (5.281)$$

The time complexity of computing α is bound by

$$\forall\ \underline{D}_{\min}, \underline{D}_{\max} \in \mathbb{R}^d, \underline{x} \in [\underline{D}_{\min}, \underline{D}_{\max}]\,,\ \forall j \in \{1,\dots,N_{\text{cell}}\}\ \exists C_\alpha \in \mathbb{R} : \\ \tau_{\underline{\bar{\mathbf{3}}}\iota^{-1}\left(\left\lfloor \frac{1}{r_c}(\underline{x}-\underline{D}_{\min})\right\rfloor - \underline{\bar{\mathbf{I}}}\iota(j) + \underline{\bar{\mathbf{3}}}\right)} \leq C_\alpha d. \quad (5.282)$$

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

$$\forall l \in \{1,\dots,3^d\}\ \exists C_c \in \mathbb{R} : \tau_{\underline{\bar{\mathbf{3}}}\iota^{-1}\left(\underline{\bar{\mathbf{4}}} - \underline{\bar{\mathbf{3}}}\iota(l)\right)} \leq C_c d. \quad (5.283)$$

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

$$\forall t \in \{1,\dots,T\}\ \exists N^{\max}_{\mathbf{p}} \in \mathbb{N} : \left|\mathbf{p}^t\right| \leq N^{\max}_{\mathbf{p}}. \quad (5.284)$$

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

$$\tau_{S([g^1,\mathbf{p}^1])} \leq T\left(N^{\max}_{\mathbf{p}}\left(\varsigma_u\left(N^{\max}_{\mathbf{p}}\right)\ \tau_i + \tau_u\left(N^{\max}_{\mathbf{p}}\right) + \tau_e\right) + \tau_f + \tau_{\overset{\circ}{e}}\right), \quad (5.285)$$

where the neighborhood-related terms $\varsigma_u\left(N^{\max}_{\mathbf{p}}\right)$ and $\tau_u\left(N^{\max}_{\mathbf{p}}\right)$ potentially depend on $N^{\max}_{\mathbf{p}}$. 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

$$\forall t \in \{1,\dots,T\},\ w \in \{1,\dots,N_{\text{cell}}\}\ \exists n_{\max} \in \mathbb{N} :\ \left|\left\langle\left\langle\mathcal{P}^t\right\rangle_w\right\rangle_{\frac{3^d+1}{2}}\right| \leq n_{\max}. \quad (5.286)$$

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

$$\forall t \in \{1,\dots,T\},\ w \in \{1,\dots,N_{\text{cell}}\} :\ \left|\bigcirc_{l=1}^{3^d}\left\langle\left\langle\mathcal{P}^t\right\rangle_w\right\rangle_l\right| \leq 3^d n_{\max}. \quad (5.287)$$

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

$$\forall g \in \{g^1, \ldots, g^T\}, \mathbf{p} \in \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}, j, k \in \{1, \ldots, |\mathbf{p}|\}\ \exists C_u \in \mathbb{R}: \\ \tau_{|\underline{x}_k - \underline{x}_j| \leq r_c} + \tau_{\Omega(g, p_k, p_j)} \leq dC_u, \quad (5.288)$$

then the time complexity and the size of the neighborhood function are bound by

$$\tau_u \leq 3^d n_{\max} d C_u, \qquad \varsigma_u \leq 3^d n_{\max}. \quad (5.289)$$

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 $collect_{\tilde{g}}^{ALL}$, $dist_{(\tilde{g}, \mathcal{G})}^{ALL}$, $step_{\mathcal{G}}^{ALL}$, $copy_{\tilde{g}}^{ALL}$, the evolve function of the global variable for each cell, and the stop function. Then,

$$\begin{aligned} \tau_{\tilde{S}([g^1, \mathbf{p}^1])}(1) \quad \leq T\Bigg(\quad & \tau_f + \underbrace{N_{\text{cell}} \tau_{\overset{\circ}{e}}}_{(global\ variable\ evolve)} \\ & + \underbrace{\prod_{k=1}^{3^d}{}^{k} N_{\text{cell}}^* \left(C_\gamma d + 3^d (C_\beta d + C_c d + n_{\max}) \right)}_{(collect)} \\ & + \underbrace{N_{\text{cell}} n_{\max} (C_\alpha d + 1)}_{(dist)} \\ & + \underbrace{N_{\text{cell}} n_{\max} \left(\tau_e + 3^d n_{\max} C_u d + 3^d n_{\max} \tau_i \right)}_{(step)} \\ & + \underbrace{\prod_{k=1}^{3^d}{}^{k} N_{\text{cell}}^* \left(C_\gamma d + 3^d (C_\beta d + n_{\max}) \right)}_{(copy)} \Bigg). \end{aligned} \quad (5.290)$$

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

$$\tau_{\tilde{S}([g^1, \mathbf{p}^1])}(1) \leq T\Bigg(C_f + N_{\text{cell}} \left(C_{\overset{\circ}{e}} + C_{dist} + C_{step} \right) \\ + \prod_{k=1}^{3^d}{}^{k} N_{\text{cell}}^* \left(C_{collect} + C_{copy} \right) \Bigg). \quad (5.291)$$

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

$$\tau_{\tilde{S}([g^1, \mathbf{p}^1])}(1) \leq T \left(C_f + N_{\text{cell}} (C_{\overset{\circ}{e}} + C_{collect} + C_{dist} + C_{step} + C_{copy}) \right). \quad (5.292)$$

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) here 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 k-th 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

$$\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(n_{CPU}) \leq T\Bigg(C_f + \Xi_{calc}(N_{\text{cell}}, n_{CPU}, d)(C_{\overset{\circ}{e}} + C_{dist} + C_{step}) \\ + \Xi_{com}(N_{\text{cell}}, n_{CPU}, d)(C_{collect} + C_{copy})\Bigg), \quad (5.293)$$

where

$$\Xi_{calc}(N_{\text{cell}}, n_{CPU}, d) := \begin{cases} \left\lceil \frac{3^d}{n_{CPU}} \right\rceil \frac{N_{\text{cell}}}{3^d} & \text{if} \quad n_{CPU} \in \{1, \ldots, 3^d\} \\ M_2 & \text{if} \quad n_{CPU} \in \{3^d, \ldots, N_{\text{cell}}\} \\ 1 & \text{if} \quad n_{CPU} > N_{\text{cell}}, \end{cases} \quad (5.294)$$

and

$$\Xi_{com}(N_{\text{cell}}, n_{CPU}, d) := \begin{cases} N_{\text{cell}} & \text{if} \quad n_{CPU} \in \{1, \ldots, 3^d\} \\ n_1 M_1 + n_2 M_2 & \text{if} \quad n_{CPU} \in \{3^d, \ldots, N_{\text{cell}}\} \\ 3^d & \text{if} \quad n_{CPU} > N_{\text{cell}} \end{cases}, \quad (5.295)$$

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

$$n_1 := n_{CPU} \bmod 3^d, \quad (5.296)$$

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

$$n_2 := 3^d - n_1, \quad (5.297)$$

the maximum number of processes per processor is

$$M_1 := \left\lceil \frac{N_{\text{cell}}}{3^d \left\lceil \frac{n_{CPU}}{3^d} \right\rceil} \right\rceil, \quad (5.298)$$

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

$$M_2 := \left\lceil \frac{N_{\text{cell}}}{3^d \left\lfloor \frac{n_{CPU}}{3^d} \right\rfloor} \right\rceil. \quad (5.299)$$

For $n_{CPU} \in \{1, \ldots, 3^d\}$, 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 $\left\lceil \frac{3^d}{n_{CPU}} \right\rceil \frac{N_{\text{cell}}}{3^d}$, where $\left\lceil \frac{3^d}{n_{CPU}} \right\rceil$ is the maximum number of checkerboard pattern 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 $n_{CPU} \in \left\{3^d, \ldots, N_{\text{cell}}\right\}$, 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 $\left\lfloor \frac{n_{CPU}}{3^d} \right\rfloor$ 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 ($\left\lceil \frac{n_{CPU}}{3^d} \right\rceil$) to them and a number ($n_2$) of checkerboard pattern with less ($\left\lfloor \frac{n_{CPU}}{3^d} \right\rfloor$). The processors for one checkerboard pattern can communicate in parallel but sequential for the checkerboard pattern. Hence, to sum up the maximum number of communication 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 speed-ups for both strong scaling according to Amdahl's law [3] and weak scaling according to Gustafson's law [37].

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

$$speedup_{\text{cell}}\left(N_{\mathbf{p}}^{\max}\right) := \frac{\tau_{S([g^1,\mathbf{p}^1])}}{\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1)} \tag{5.300}$$

$$\approx \frac{N_{\mathbf{p}}^{\max 2} C_u d + N_{\mathbf{p}}^{\max}\left(3^d n_{\max}\tau_i + \tau_e\right) + \tau_f + \tau_{\overset{\circ}{e}}}{N_{\mathbf{p}}^{\max}\left(3^d n_{\max} C_u d + 3^d n_{\max}\tau_i + \tau_e + \frac{\tau_{\overset{\circ}{e}} + C_{collect} + C_{dist} + C_{copy}}{n_{\max}}\right) + \tau_f} \tag{5.301}$$

$$\in \frac{\mathcal{O}\left(N_{\mathbf{p}}^{\max 2}\right)}{\mathcal{O}\left(N_{\mathbf{p}}^{\max}\right)} = \mathcal{O}\left(N_{\mathbf{p}}^{\max}\right). \tag{5.302}$$

Since particles can move, the neighborhood search function checks each pair of particles to see if they are neighbors. This has a complexity in $\mathcal{O}(n^2)$, where n is the number of particles. Fast neighbor list algorithms like cell-lists [41] reduce this to $\mathcal{O}(n)$ under the assumptions made here. We can confirm this by comparing the simplified time complexity of the cell-list-based scheme (5.292) with the time complexity of the sequential state transition (5.285), 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 speed-up is $\mathcal{O}\left(N_{\mathbf{p}}^{\max}\right)$ as derived in (5.302) and visualized in figure 5.5a.

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

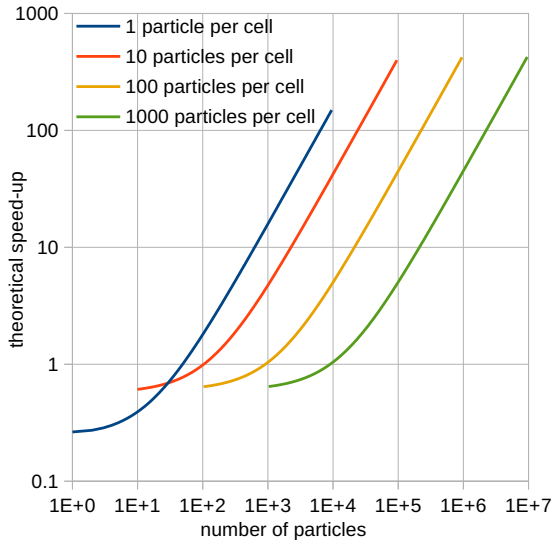
$$\begin{aligned} speedup_{\text{Amdal}}(n_{CPU}) &= \frac{\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(1)}{\tau_{\tilde{S}([g^1,\mathbf{p}^1])}(n_{CPU})} \\ &\approx \frac{C_f + N_{\text{cell}}\ (C_{\underset{e}{\circ}} + C_{collect} + C_{dist} + C_{step} + C_{copy})}{C_f + \Xi_{calc}(N_{\text{cell}}, n_{CPU}, d)(C_{\underset{e}{\circ}} + C_{dist} + C_{step}) + \Xi_{com}(N_{\text{cell}}, n_{CPU}, d)(C_{collect} + C_{copy})}. \end{aligned} \tag{5.303}$$

In this case, we can increase n_{CPU} until we reach the number of cells N_{cell}. After that, there will be no further speed-up. 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 5.5b.

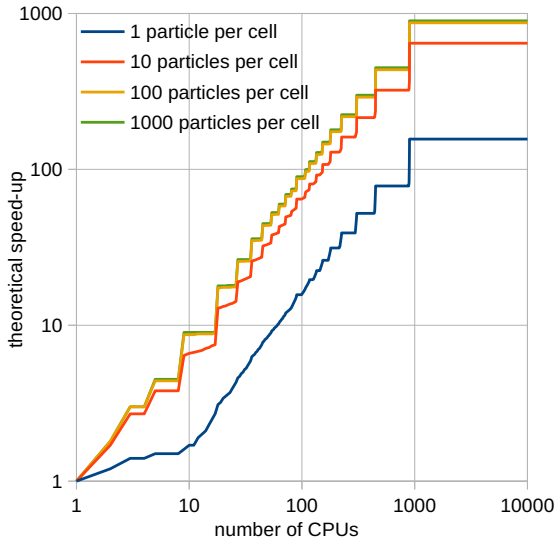
Third, Gustafson's law [37] provides an upper bound on the speed-up of the cell-list scheme on multiple processors when the ratio of problem size to process number is constant while increasing the number of processors (weak scaling); $\frac{N_{\text{cell}}}{n_{\text{CPU}}} = \text{const}$. We achieve 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 speed-up on average with steps as visualized in figure 5.5c.

$$\begin{aligned} speedup_{\text{Gustafson}}(n_{CPU}) &= \frac{\tau_{\tilde{S}([g^1,\mathbf{p}^1(n_{CPU})])}(1)}{\tau_{\tilde{S}([g^1,\mathbf{p}^1(n_{CPU})])}(n_{CPU})} \\ &\approx \frac{C_f + n_{CPU} \cdot N_{\text{cell}}\ (C_{\underset{e}{\circ}} + C_{collect} + C_{dist} + C_{step} + C_{copy})}{C_f + \Xi_{calc}(n_{CPU} \cdot N_{\text{cell}}, n_{CPU}, d)(C_{\underset{e}{\circ}} + C_{dist} + C_{step}) + \Xi_{com}(n_{CPU} \cdot N_{\text{cell}}, n_{CPU}, d)(C_{collect} + C_{copy})} \end{aligned} \tag{5.304}$$

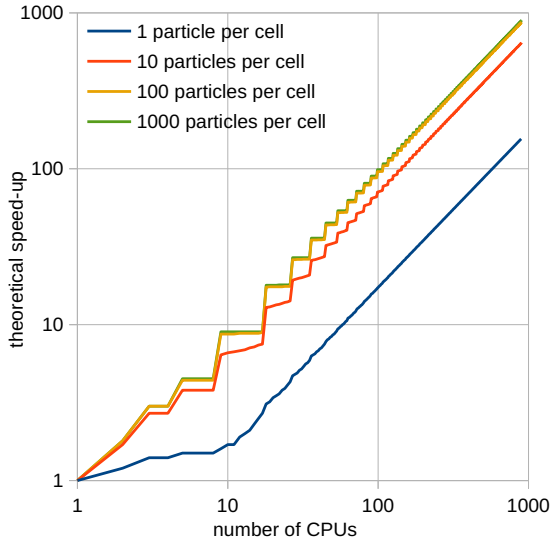
Overall the scheme behaves as expected for cell-list algorithms.



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



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



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

Figure 5.5: Theoretical speed-up bounds. The constants are chosen to be $d = 2$, $C_u = C_\alpha = C_\beta = C_\gamma = 1$, $\tau_i = \tau_e = 3$, and $\tau_f = \tau_{\overset{\circ}{e}} = 1$. For Amdahl's and Gustafson's laws, $N_{\text{cell}} = 900$.

5.4 Conclusion

In high-performance computing, the increasing computational power is made more accessible by generic software frameworks. They bridge the gap between accessible programming and heterogeneous-hardware parallelization [76]. Even though the frameworks are generic, the conditions under which an algorithm is parallelizable, its complexity, and how it scales

are usually derived application-wise.

Here, we provide a shared- and a distributed-memory parallelization scheme for particle methods independent of specific applications. We proved the correctness of the presented parallelization schemes 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 schemes executed on both sequential and parallel computers, and we discussed the parallel scalability limits.

The presented parallelization schemes are not novel, and they are, in fact, similar to 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 proofs guarantee correctness. The presented schemes are only a tiny fraction of possible schemes. Less and more restrictive parallelization schemes with better time complexity or broader applicability are possible. A limitation of the proposed schemes is that they assume only pull interactions between the particles. This neglects the potential runtime benefits and versatility of symmetric interaction evaluations. However, pull interactions are suitable for more computer architectures, especially in a shared memory setting, which could allow for combining the distributed scheme with the shared memory scheme. In the distributed-memory scheme, pull interactions also reduce communication since only particles in the center cell of a process are changed. They do not need to be communicated back, as would be the case for push or symmetric interactions, which also change copies of particles from other processes [77]. We also restricted the neighborhood function such that the cell-list strategy became applicable. This limits the expressiveness of the particle method but allows the efficient distribution of moving particles. Further, the constraint that particles are not allowed to leave the domain keeps domain handling simple. Otherwise, cells would dynamically need to be added or removed as necessary, resulting in a much more complex and dynamic mapping of cells to processes. We also restricted particles to not moving 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, requiring particles to never move further than one sub-domain per state iteration. Additionally, we restricted the global variable only to be changed by the evolve function of the global variable and not by the evolve function of any particle. Therefore, no global operations are allowed where global variable changes require additional synchronization. Still, global variable changes can be independently computed locally, keeping them in sync 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 sub-system, providing at least a bound on the scalability and permitting correctness proofs.

Notwithstanding these limitations, with their proof of equivalence to the sequential particle methods definition, the present parallelization schemes stand in contrast with the mainly algorithm-specific or empirically tested parallelization schemes used so far. The rigorousness of our analysis paves the way for future research into the theory of parallel scientific simulation algorithms and the engineering of provably correct parallel software implementations independent of specific applications.

Future theoretical work could optimize the presented schemes for computer architectures with parallel or synchronously clocked communication and computation. Global operations could also be allowed, and the checkerboard-like communication pattern of the shared-memory scheme could be relaxed, leading to an improved scalability pre-factor (up to 27-fold). Also, the network topology of the machine's interconnect could be explicitly incorporated into the parallelization scheme. Furthermore, proofs for push, symmetric, and no 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 schemes 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 [76].

Overall, the presented proven parallelization schemes provide a way to parallelize classes of particle methods on shared- and distributed-memory systems with full knowledge of their validity, performance, and assumptions. We proved that they compute the same result as the underlying sequential particle method. Therefore, using them in a general framework for particle methods is suitable, even for critical computations, since the proofs guarantee that the parallelizations do not change the results. Therefore it marks the starting point of a well-founded discussion about the parallelization of particle methods independent of specific applications.

Chapter 6

Turing Powerfulness and Halting Decidability

6.1 Introduction

Our definition of particle methods is practically universal. It can be applied to various fields, including discrete element methods (sec. 4.2), diffusion simulations (sec. 4.3), fluid dynamics (sec. 4.4), molecular dynamics (sec. 4.5), triangulation refinement (sec. 4.6), Conway's game of life (sec. 4.7), and solving linear equation systems (sec. 4.8). Despite this broad applicability, the theoretical limits of our particle methods definition are unclear.

Automata theory provides the means to investigate these theoretical limits. Automata theory is the study of abstract machines (automata) and the problems that these machines can solve. Two of the challenges in automata theory are, first, to classify a formalism or automaton in terms of its powerfulness and second, the ability to decide whether it halts or runs forever.

The powerfulness is measured in the formal languages an automaton can accept. This relates the automata theory to formal languages and grammars, hence, to the Chomsky hierarchy [16]. According to this hierarchy, the most powerful automaton is the Turing machine [84]. Turing machines resemble the concept on which most modern computers are built. Therefore, it is often said a Turing machine can compute everything a modern computer can and vice versa. From a practical perspective, this might be almost always true. Still, a Turing machine is, in theory, more powerful since it has an infinitely long tape (memory) in contrast to modern computers, which have huge but finite memory.

The ability to decide whether an algorithm on an automaton halts or runs forever is called halting decidability. For Turing machines, it is generally not decidable if an algorithm halts [84]. Therefore, halting decidable automata are less powerful than Turing machines.

Particle methods can be interpreted as automata. Therefore, we address these two problems, powerfulness and halting decidability, in this chapter. Particle methods are Turing powerful. This can be seen in the application of Conway's game of life (sec. 4.7), which is proven to be Turing powerful [74, 73] or in the simple reduction, where a whole Turing machine is implemented in the evolve function of the global variable. More interesting is the question of how much we can restrict particle methods such that they are

still Turing powerful, which we address with two sets of restrictions, where one is not the subset of the other. For halting decidability, the question is the opposite. Here we ask: How less can we restrict particle methods such that they are still halting decidable? We address this with one set of restrictions and prove that the halting problem is decidable under these restrictions.

6.2 Turing Machine

Turing machines [84] are the most general and powerful automata so far.

There exist many equivalently powerful formulations of Turing machines. We chose to follow the formulation of Kozen [52]. A Turing machine consists of a semi-finite tape and a read-write head, and it is in a state. The tape consists of cells. Each cell carries a symbol from the tape alphabet Γ. The most left cell contains the left end-marker $\vdash$. At the start, a finite string of input symbols from Σ stands on the right of the end-marker. The tape's infinite rest is filled with blank symbols $\_$. The read-write head points exactly on one cell. At the start, it points at the most left cell (first cell). Depending on the symbol of the pointed-to cell and the current state of the Turing machine, the head changes the state of the Turing machine, the symbol of the cell, and moves right (1) or left (-1). The transition function δ determines this.

In formal terms, the here used notion of Turing machines is defined as follows.

Definition 18. This definition is taken from Kozen [52].
A **deterministic one tape Turing machine** is a tuple

$$(Q, \Sigma, \Gamma, \vdash, \_, \delta, \textbf{start}, \textbf{accept}, \textbf{reject}) \tag{6.1}$$

such that:

$$Q \text{ is a finite set of states,} \tag{6.2}$$
$$\Sigma \text{ is a finite input alphabet,} \tag{6.3}$$
$$\Gamma \text{ is a finite tape alphabet with } \Sigma \subseteq \Gamma, \tag{6.4}$$
$$\vdash \text{ is the left end-marker with } \vdash \in \Gamma \backslash \Sigma, \tag{6.5}$$
$$\_ \text{ is the blank symbol with } \_ \in \Gamma \backslash \Sigma, \tag{6.6}$$
$$\delta \text{ is the transition function with } \delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 1\}, \tag{6.7}$$
$$\textbf{start} \text{ is the start state,} \tag{6.8}$$
$$\textbf{accept} \text{ is the accept state,} \tag{6.9}$$
$$\textbf{reject} \text{ is the reject state with } \textbf{reject} \neq \textbf{accept}. \tag{6.10}$$

To prevent the head from overwriting the left end marker and leaving the tape on the left side, the following is required for the transition function δ.

$$\forall q \in Q \exists \overline{q} \in Q : \ \delta(q, \vdash) = (\overline{q}, \vdash, 1) \tag{6.11}$$

Additionally, if the Turing machine enters the accept or reject state, it must stay in that state.

$$\forall b \in \Gamma \ \exists c, \overline{c} \in \Gamma \wedge d, \overline{d} \in \{-1, 1\} : \qquad \delta(\textbf{accept}, b) = (\textbf{accept}, c, d) \tag{6.12}$$
$$\wedge \delta(\textbf{reject}, b) = (\textbf{reject}, \overline{c}, \overline{d}) \tag{6.13}$$

Definition 19. This definition is taken from Kozen [52].
A **configuration** of a Turing machine is a tuple:

$$(q, \vdash \mathbf{x}\_^{\omega}, n) \in C := Q \times \{\vdash \mathbf{x}\_^{\omega} : \mathbf{x} \in \Gamma^*\} \times \mathbb{N}, \tag{6.14}$$

where

$$q \text{ is the state,} \tag{6.15}$$
$$\vdash \mathbf{x}\_^{\omega} \text{ is the semi finite string on the tape, and} \tag{6.16}$$
$$n \text{ is the head's position on the tape.} \tag{6.17}$$

A configuration contains all relevant information about a Turing machine's state at a time point.

The semi-finite string $\vdash \mathbf{x}\_^{\omega}$ contains the left end-marker $\vdash$, a finite string $\mathbf{x} \in \Gamma^*$ and the semi-finite string $\_^{\omega}$ where ω is the smallest ordinal number.

The **start configuration** of a Turing machine is:

$$(\mathbf{start}, \mathbf{x}^1\_^{\omega}, 1) \text{ with } \mathbf{x}^1 = \vdash \mathbf{y} \text{ and } \mathbf{y} \in \Sigma^*, \tag{6.18}$$

where $\mathbf{y}$ is the finite input string of the Turing machine and 1 means that the head is scanning $\vdash$ and is most left.

Definition 20. This definition is taken from [52].
Be x_n the n-th symbol of the string $\mathbf{x}$, where x_1 is the most left symbol, and be $s_b^n(\mathbf{x})$ the string where the n-th symbol of $\mathbf{x}$ is replaced by b. Then the **next configuration relation** $\xrightarrow[M]{1}$ of a Turing machine is defined by:

$$\xrightarrow[M]{1} \subseteq C \times C \tag{6.19}$$

$$\begin{aligned} &\Big((q, \mathbf{x}, n), (q', \mathbf{x}', n')\Big) \in \xrightarrow[M]{1} \\ &\qquad \longleftrightarrow \ (q', \mathbf{x}', n') = (\overline{q}, s_b^n(\mathbf{x}), n + d) \quad \text{for } \delta(q, x_n) = (\overline{q}, b, d) \end{aligned} \tag{6.20}$$

Since this is a deterministic Turing machine, $\xrightarrow[M]{1}$ is a function.

6.3 Turing Powerfulness of Particle Methods Under a First Set of Constraints

Theorem 3 (Turing Powerfulness of particle methods under a first set of constraints)**.** *Particle methods are Turing powerful under the constraints:*

It has an empty neighborhood

$$u([g, \mathbf{p}], j) = (), \tag{6.21}$$

the interact function is the identity

$$i\left(g, p_j, p_k\right) = \left(p_j, p_k\right), \tag{6.22}$$

the evolve function is order-independent regarding g

$$_1e\left(_1e\left(g, p'\right), p''\right) = {}_1e\left(_1e\left(g, p''\right), p'\right), \tag{6.23}$$

and independent of previous evolutions regarding p

$$_2e\left(g, p'\right) = {}_2e\left(_1e\left(g, p''\right), p'\right), \tag{6.24}$$

and the size of the global variable ς_g, the size of each particle ς_p, the time complexity of the stop function τ_f, of the evolve function τ_e and of the evolve function of the global variable $\tau_{\overset{\circ}{e}}$ are in $\mathcal{O}(\log(|\mathbf{p}^t|))$.

$$\varsigma_{g^t}, \varsigma_{p_j^t}, \tau_f, \tau_e, \tau_{\overset{\circ}{e}} \in \mathcal{O}\left(\log(|\mathbf{p}^t|)\right). \tag{6.25}$$

Proof.
For the proof, we use the overbar notation

$$(\overline{\mathfrak{q}}, \overline{\mathfrak{z}}, \overline{\mathfrak{d}}) := \delta(\mathfrak{q}, z) \tag{6.26}$$

for the results of the transition function. Without loss of generality, we assume a strict total order $<$ on the set of states Q of the Turing machine, where

$$\forall q \in Q \backslash \{\textbf{start}\} : \textbf{start} < q \tag{6.27}$$

We construct a particle method that fulfills the constraints and emulates an arbitrary Turing machine as defined in section 6.2. Then we prove that this is the case.

Constructed Particle Method for a Turing Machine

The constructed particle method that resembles a Turing machine and fulfills the constraints (6.21) to (6.25) is:

$$p := (\mathfrak{k}, \mathfrak{z}) \quad \text{for } p \in P := \mathbb{N}_{>0} \times \Gamma \tag{6.28}$$

$$g := (\mathfrak{q}, \Delta\mathfrak{q}, \mathfrak{d}, \mathfrak{m}, \mathfrak{M}) \quad \text{for } p \in P := Q^2 \times \{-1, 1\} \times {\mathbb{N}_{>0}}^2 \tag{6.29}$$

$$u\left([g, \mathbf{p}], j\right) := () \tag{6.30}$$

$$f(g) := (\mathfrak{q} = \textbf{reject} \ \vee \mathfrak{q} = \textbf{accept}\}) \tag{6.31}$$

$$i(g, p_j, p_k) := (p_j, p_k) \tag{6.32}$$

$$e(g, p_j) := \begin{cases} \left(\left(\mathfrak{q}, \max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}), \max(\mathfrak{d}, \overline{\mathfrak{d}}), \mathfrak{m}, \mathfrak{M}\right), ((\mathfrak{k}_j, \overline{\mathfrak{z}}))\right) & \text{if } \mathfrak{k}_j = \mathfrak{m} \wedge \mathfrak{m} + \overline{\mathfrak{d}} \leq \mathfrak{M} \\ \Big(\left(\mathfrak{q}, \max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}), \max(\mathfrak{d}, \overline{\mathfrak{d}}), \mathfrak{m}, \mathfrak{M}\right), & \text{if } \mathfrak{k}_j = \mathfrak{m} \wedge \mathfrak{m} + \overline{\mathfrak{d}} > \mathfrak{M} \\ \ ((\mathfrak{k}_j, \overline{\mathfrak{z}}), (\mathfrak{k}_j + 1, \_\ )) \Big) & \\ (g, (p_j)) & \text{else} \end{cases} \tag{6.33}$$

$$\overset{\circ}{e}(g) := \begin{cases} (\Delta\mathfrak{q}, \textbf{start}, -1, \mathfrak{m} + \mathfrak{d}, \mathfrak{M} + 1) & \text{if } \mathfrak{m} + \mathfrak{d} > \mathfrak{M} \\ (\Delta\mathfrak{q}, \textbf{start}, -1, \max(1, \mathfrak{m} + \mathfrak{d}), \mathfrak{M}) & \text{else} \end{cases} \tag{6.34}$$

This particle method resembles an arbitrary Turing machine regarding the definition of Kozen (sec. 6.2). Each particle mimics one cell of the tape. It contains the cell index $\mathfrak{k}$ and symbol $\mathfrak{z}$. The global variable represents the Turing machine's read-write head and contains the properties for orchestrating the creation of new particles (tape cell). Therefore, it contains the state of the Turing machine $\mathfrak{q}$, an accumulator for the state $\Delta\mathfrak{q}$, the movement direction of the head $\mathfrak{d}$, the position (index) of the head $\mathfrak{m}$, and the number of particles $\mathfrak{M}$. The accumulator $\Delta\mathfrak{q}$ is necessary for the order-independence of the evolve function. The neighborhood is empty. Hence, there are no interactions. The evolve function is only different from the identity for the particle to which the head points ($p_{\mathfrak{m}}$). In this case, the transition function δ is calculated, and the results are stored in the global variable and the particle accordingly. $\mathfrak{q}$ is not directly overwritten to make the evolve function, in general, independent of previous evolutions. The max function for overwriting the accumulator $\Delta\mathfrak{q}$ and the direction $\mathfrak{d}$ generally makes the evolve function order-independent. For a Turing-machine-related particle methods instance, both do not have an influence. If the head points, in the following particle methods state, to a non-existing particle/ empty cell $\mathfrak{m} + \overline{\mathfrak{d}} > \mathfrak{M}$ this particle is created with the blank symbol $\_$ by the evolve function. The evolve function of the global variable $\overset{\circ}{e}$ changes the position of the head $\mathfrak{m}$ according to the head movement $\mathfrak{d}$, adds 1 to the number of particles if a new particle was created, and resets the accumulator $\Delta\mathfrak{q}$ and the direction $\mathfrak{d}$ to the smallest values, i.e., **start** and -1.

Particle Method for Turing Machine Fulfills Constrains

We prove that the constraints are fulfilled by the particle method. The fulfillment of the constraints (6.21) and (6.22) are directly visible in the definition of the particle method algorithm (6.30) and (6.32).

To prove that the evolve function is order-independent regarding g (6.23), we have to distinguish between four cases.

First, $\mathfrak{k}' = \mathfrak{k}'' = \mathfrak{m}$

$$
\begin{aligned}
&{}_1e\left({}_1e\left(g, p'\right), p''\right) && (6.35)\\
&= {}_1e\left(\left(\mathfrak{q}, \max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}'), \max(\mathfrak{d}, \overline{\mathfrak{d}}'), \mathfrak{m}, \mathfrak{M}\right), p''\right) && (6.36)\\
&= \left(\mathfrak{q}, \max\left(\max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}'), \overline{\mathfrak{q}}''\right), \max\left(\max(\mathfrak{d}, \overline{\mathfrak{d}}'), \overline{\mathfrak{d}}''\right), \mathfrak{m}, \mathfrak{M}\right) && (6.37)\\
&= \left(\mathfrak{q}, \max\left(\Delta\mathfrak{q}, \overline{\mathfrak{q}}', \overline{\mathfrak{q}}''\right), \max\left(\mathfrak{d}, \overline{\mathfrak{d}}', \overline{\mathfrak{d}}''\right), \mathfrak{m}, \mathfrak{M}\right) && (6.38)\\
&= \left(\mathfrak{q}, \max\left(\max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}''), \overline{\mathfrak{q}}'\right), \max\left(\max(\mathfrak{d}, \overline{\mathfrak{d}}''), \overline{\mathfrak{d}}'\right), \mathfrak{m}, \mathfrak{M}\right) && (6.39)\\
&= {}_1e\left(\left(\mathfrak{q}, \max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}''), \max(\mathfrak{d}, \overline{\mathfrak{d}}''), \mathfrak{m}, \mathfrak{M}\right), p'\right) && (6.40)\\
&= \underline{{}_1e\left({}_1e\left(g, p''\right), p'\right)}. && (6.41)
\end{aligned}
$$

Second, $\mathfrak{k}' = \mathfrak{m}$ and $\mathfrak{k}'' \neq \mathfrak{m}$, then we set $\tilde{g} := {}_1e\left(g, p'\right)$ and know $g = {}_1e\left(g, p''\right)$. From

this follows

$$
\begin{aligned}
&{}_1e\left({}_1e\left(g,p'\right),p''\right) && (6.42)\\
&= {}_1e\left(\tilde{g},p''\right) && (6.43)\\
&= \tilde{g} && (6.44)\\
&= {}_1e\left(g,p'\right) && (6.45)\\
&= \underline{{}_1e\left({}_1e\left(g,p''\right),p'\right)}. && (6.46)
\end{aligned}
$$

Third, $\mathfrak{k}' \neq \mathfrak{m}$ and $\mathfrak{k}'' = \mathfrak{m}$, then we know $g = {}_1e\left(g,p'\right)$ and set $\tilde{g} := {}_1e\left(g,p''\right)$. From this follows

$$
\begin{aligned}
&{}_1e\left({}_1e\left(g,p'\right),p''\right) && (6.47)\\
&= {}_1e\left(g,p''\right) && (6.48)\\
&= \tilde{g} && (6.49)\\
&= {}_1e\left(\tilde{g},p'\right) && (6.50)\\
&= \underline{{}_1e\left({}_1e\left(g,p''\right),p'\right)}. && (6.51)
\end{aligned}
$$

Fourth, $\mathfrak{k}' \neq \mathfrak{m}$ and $\mathfrak{k}'' \neq \mathfrak{m}$, then we know $g = {}_1e\left(g,p'\right) = {}_1e\left(g,p''\right)$. Hence,

$$
\begin{aligned}
&{}_1e\left({}_1e\left(g,p'\right),p''\right) && (6.52)\\
&= {}_1e\left(g,p''\right) && (6.53)\\
&= g && (6.54)\\
&= {}_1e\left(g,p'\right) && (6.55)\\
&= \underline{{}_1e\left({}_1e\left(g,p''\right),p'\right)}. && (6.56)
\end{aligned}
$$

Next, we prove the condition that the evolve function is independent of previous evolutions regarding p (6.24). We have to distinguish between two cases.

First, $\mathfrak{k}'' = \mathfrak{m}$, for the evolve function, only the global variable's properties $\mathfrak{q}$, $\mathfrak{m}$, and $\mathfrak{M}$ are relevant. The evolve function does not change these properties. Hence,

$$
\begin{aligned}
&{}_2e\left({}_1e\left(g,p''\right),p'\right) && (6.57)\\
&= {}_2e\left(\left(\mathfrak{q},\max\left(\Delta\mathfrak{q},\overline{\mathfrak{q}}''\right),\max\left(\mathfrak{d},\overline{\mathfrak{d}}''\right),\mathfrak{m},\mathfrak{M}\right),p'\right) && (6.58)\\
&= {}_2e\left(\left(\mathfrak{q},\Delta\mathfrak{q},\mathfrak{d},\mathfrak{m},\mathfrak{M}\right),p'\right) && (6.59)\\
&= \underline{{}_2e\left(g,p'\right)}. && (6.60)
\end{aligned}
$$

Second, $\mathfrak{k}'' \neq \mathfrak{m}$, then we know $g = {}_1e\left(g,p''\right)$. Hence,

$$
\begin{aligned}
&{}_2e\left({}_1e\left(g,p''\right),p'\right) && (6.61)\\
&= \underline{{}_2e\left(g,p'\right)}. && (6.62)
\end{aligned}
$$

The last constraint we prove is that the size of the global variable ς_g and each particle ς_p, the time complexity of the stop function τ_f, of the evolve function τ_e and of the evolve function of the global variable $\tau_{\overset{\circ}{e}}$ are in $\mathcal{O}(\log(|\mathbf{p}^t|))$. The only potentially growing properties are the cell index $\mathfrak{k}$ of the particles, the head position $\mathfrak{m}$, and the number of

particles $\mathfrak{M}$. All three numbers only increase if a new particle is created, hence when the number of particles increases. This happens for $\mathfrak{m} + \overline{\mathfrak{d}} > \mathfrak{M}$ in the evolve function and in the evolve function e of the global variable $\overset{\circ}{e}$. In theory, more than one particle could be created per step, but the properties would still be only advanced by one. Therefore, $\varsigma_g, \varsigma_p \in \mathcal{O}\left(\log(|\mathbf{p}^t|)\right)$. For a particle methods instance for a Turing machine, as defined in section 6.2, the creation of multiple particles per step would not happen. The operations in $f, e, \overset{\circ}{e}$ are only comparisons, additions, and replacements. If we assume these operations are done digit-vise and we know that the numbers are in $\mathcal{O}\left(\log(|\mathbf{p}^t|)\right)$, we can follow the time complexity of the functions $f, e, \overset{\circ}{e}$ are also in $\mathcal{O}\left(\log(|\mathbf{p}^t|)\right)$. Hence,

$$\varsigma_{g^t}, \varsigma_{p_j^t}, \tau_f, \tau_e, \tau_{\overset{\circ}{e}} \in \mathcal{O}\left(\log(|\mathbf{p}^t|)\right). \tag{6.63}$$

Therefore the particle method fulfills all constraints from (6.21) to (6.25).

Particle Method Emulates Arbitrary Turing Machine

To prove that the particle method emulates an arbitrary Turing machine, we need to translate the start configuration of a Turing machine (6.18) into a particle methods instance. Then we prove for all states of the particle method that the back translation is the corresponding configuration of the Turing machine. The last step is to show that the particle method stops when the Turing machine reaches an accept or reject state.

We define for this particle method the translation function as

$$\psi\left((q, \mathbf{x}\_^{\omega}, n)\right) := [(q, \mathbf{start}, -1, n, |\mathbf{x}|), ((1, x_1), (2, x_2), ..., (|\mathbf{x}|, x_{|\mathbf{x}|}))], \tag{6.64}$$

and the back translation as

$$\psi^{-1}\left(\left[(\mathfrak{q}, \Delta\mathfrak{q}, \mathfrak{d}, \mathfrak{m}, \mathfrak{M}), ((\mathfrak{k}_1, \mathfrak{z}_1), (\mathfrak{k}_2, \mathfrak{z}_2), ..., (\mathfrak{k}_{|\mathbf{p}|}, \mathfrak{z}_{|\mathbf{p}|}))\right]\right) := \left(\mathfrak{q}, \mathfrak{z}_1\mathfrak{z}_2 \cdots \mathfrak{z}_{|\mathbf{p}|}\_^{\omega}, \mathfrak{m}\right). \tag{6.65}$$

We see these translation functions are only copying values. The only part which might be a calculation is the length of the finite input string $|\mathbf{x}|$. Therefore, the translations do not carry any calculation of the Turing machine.

The instance of the particle method is defined by translating the start configuration $\alpha^1 = (\mathbf{start}, \mathbf{x}^1\_^{\omega}, 1)$. Hence,

$$[g^1, \mathbf{p}^1] := \psi(\alpha^1). \tag{6.66}$$

We need additional criteria to be fulfilled by each particle methods step so that the particle method works. In total, the following criteria are needed:

$$\begin{aligned} &\psi^{-1}\left(\left[g^t, \mathbf{p}^t\right]\right) = \alpha^t \\ &\wedge \mathfrak{M}^t = |\mathbf{p}^t| \\ &\wedge \mathfrak{m}^t \leq \mathfrak{M}^t \\ &\wedge \mathfrak{d}^t = -1 \\ &\wedge \Delta\mathfrak{q}^t = \mathbf{start} \\ &\wedge \forall p_j^t \in \mathbf{p}^t : \mathfrak{k}_j^t = j. \end{aligned} \tag{6.67}$$

We prove through induction that the back translation of all particle method states is the Turing machine's corresponding configuration and that the criteria (6.67) are fulfilled.

The base case regards the instance and the start configuration.

$$\psi^{-1}\left([g^1, \mathbf{p}^1]\right) \tag{6.68}$$
$$= \psi^{-1}\left(\psi\left((\mathbf{start}, \mathbf{x}^1 \_^{\omega}, 1)\right)\right) \tag{6.69}$$
$$= \psi^{-1}([(\mathbf{start}, \mathbf{start}, -1, 1, |\mathbf{x}^1|), ((1, \vdash), (2, x_2^1), ..., (|\mathbf{x}^1|, x^1_{|\mathbf{x}^1|}))]) \tag{6.70}$$
$$= \left(\mathbf{start}, \vdash x_2^1 \cdots x^1_{|\mathbf{x}^1|} \_^{\omega}, 1\right) \tag{6.71}$$
$$= \left(\mathbf{start}, \mathbf{x}^1 \_^{\omega}, 1\right) \tag{6.72}$$
$$= \underline{\alpha^1}. \tag{6.73}$$

The rest of the criteria (6.67) follow directly from (6.70).

For the induction step, we need to prove that under the condition that the criteria (6.67) are fulfilled by $\left[g^t, \mathbf{p}^t\right]$ follows that the criteria (6.67) are fulfilled by $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

We start by calculation $\left[g^{t+1}, \mathbf{p}^{t+1}\right] = s\left(\left[g^t, \mathbf{p}^t\right]\right)$. Since the neighborhood is empty, there is no interaction. Hence, the state transition step is the evolve function e and the evolve function of the global variable $\overset{\circ}{e}$. Therefore, we can rewrite s to

$$s\left(\left[g^t, \mathbf{p}^t\right]\right) = \left[\overset{\circ}{e}\left(g *_{1e} \left(p_1^t, ..., p^t_{|\mathbf{p}^t|}\right)\right), {}_2e\left(p_1^t\right) \circ ... \circ {}_2e\left(p^t_{|\mathbf{p}^t|}\right)\right]. \tag{6.74}$$

We calculate the part of the global variable separately from that of the particle tuple. The evolve function is different from the identity just for $n^t = \mathfrak{k}_j$. This is only the case for the particle p_{n^t}. Hence,

$$g *_{1e} \left(p_1^t, ..., p^t_{|\mathbf{p}^t|}\right) \tag{6.75}$$
$$= g *_{1e} \left(p_1^t, ..., p^t_{n^t-1}, p^t_{n^t}, p^t_{n^t+1}, ..., p^t_{|\mathbf{p}^t|}\right) \tag{6.76}$$
$$= g *_{1e} \left(p^t_{n^t}, p^t_{n^t+1}, ..., p^t_{|\mathbf{p}^t|}\right) \tag{6.77}$$
$$= \left(q^t, \max(\mathbf{start}, \overline{\mathfrak{q}}^t), \max(-1, \overline{\mathfrak{d}}^t), n^t, |\mathbf{p}^t|\right) *_{1e} \left(, p^t_{n^t+1}, ..., p^t_{|\mathbf{p}^t|}\right) \tag{6.78}$$
$$= \left(q^t, \overline{\mathfrak{q}}^t, \overline{\mathfrak{d}}^t, n^t, |\mathbf{p}^t|\right) *_{1e} \left(, p^t_{n^t+1}, ..., p^t_{|\mathbf{p}^t|}\right) \tag{6.79}$$
$$= \left(q^t, \overline{\mathfrak{q}}^t, \overline{\mathfrak{d}}^t, n^t, |\mathbf{p}^t|\right) \tag{6.80}$$
$$= \left(q^t, q^{t+1}, \overline{\mathfrak{d}}^t, n^t, |\mathbf{p}^t|\right) \tag{6.81}$$

The evolve function of the global variable $\overset{\circ}{e}$ has two cases. First, $\mathfrak{m} + \mathfrak{d} > \mathfrak{M}$ in this case this means $n^t + \mathfrak{d}^t > |\mathbf{p}^t|$. Hence,

$$\overset{\circ}{e}\left(g *_{1e} \left(p_1^t, ..., p^t_{|\mathbf{p}^t|}\right)\right) = \overset{\circ}{e}\left(\left(q^t, q^{t+1}, \mathfrak{d}^t, n^t, |\mathbf{p}^t|\right)\right) \tag{6.82}$$
$$= \left(q^{t+1}, \mathbf{start}, -1, n^t + \mathfrak{d}^t, |\mathbf{p}^t| + 1\right) \tag{6.83}$$
$$= \underline{\left(q^{t+1}, \mathbf{start}, -1, n^{t+1}, |\mathbf{p}^t| + 1\right)}. \tag{6.84}$$

Second, else. Hence,

$$\overset{\circ}{e}\left(g *_{1e} \left(p_1^t, ..., p_{|\mathbf{p}^t|}^t\right)\right) \tag{6.85}$$
$$= \overset{\circ}{e}\left(\left(q^t, q^{t+1}, \mathfrak{d}^t, n^t, |\mathbf{p}^t|\right)\right) \tag{6.86}$$
$$= \left(q^{t+1}, \mathbf{start}, -1, \max\left(1, n^t + \mathfrak{d}^t\right), |\mathbf{p}^t|\right) \tag{6.87}$$

We know $n^t \geq 1$, $p_1^t = (1, \vdash)$, and $\delta(q, \vdash) = (\overline{q}, \vdash, 1)$. Therefore, if $n^t = 1$, then $\mathfrak{d}^t = 1$. Hence,

$$= \left(q^{t+1}, \mathbf{start}, -1, n^t + \mathfrak{d}^t, |\mathbf{p}^t|\right) \tag{6.88}$$
$$= \underline{\left(q^{t+1}, \mathbf{start}, -1, n^{t+1}, |\mathbf{p}^t|\right)}. \tag{6.89}$$

The global variable is finished. Next is the particle tuple. We know the evolve function differs from the identity for $n^t = \mathfrak{k}_j$. This is the case only for the particle p_{n^t}. We need to distinguish two cases.
First, when $\mathfrak{m} + \overline{\mathfrak{d}} > \mathfrak{M}$ in this case $n^t + \overline{\mathfrak{d}}^t > |\mathbf{p}^t|$. Additionally, we know that $n^t \leq |\mathbf{p}^t|$ and $\overline{\mathfrak{d}}^t \in \{-1, 1\}$. This implies that $n^t = |\mathbf{p}^t|$ and $\overline{\mathfrak{d}}^t = 1$. Hence,

$${}_2e\left(p_1^t\right) \circ ... \circ {}_2e\left(p_{|\mathbf{p}^t|}^t\right) \tag{6.90}$$
$$= \left(p_1^t\right) \circ ... \circ \left(p_{|\mathbf{p}^t|-1}^t\right) \circ \left(\left(|\mathbf{p}^t|, \overline{\mathfrak{z}}^t\right), \left(|\mathbf{p}^t| + 1, \_\right)\right) \tag{6.91}$$
$$= \underline{\left(p_1^t, ..., p_{|\mathbf{p}^t|-1}^t, \left(|\mathbf{p}^t|, \overline{\mathfrak{z}}^t\right), \left(|\mathbf{p}^t| + 1, \_\right)\right)} \tag{6.92}$$

Second, in the case $n^t + \overline{\mathfrak{d}}^t \leq |\mathbf{p}^t|$, we get

$${}_2e\left(p_1^t\right) \circ ... \circ {}_2e\left(p_{|\mathbf{p}^t|}^t\right) \tag{6.93}$$
$$= \left(p_1^t\right) \circ ... \circ \left(p_{n^t-1}^t\right) \circ \left(\left(n^t, \overline{\mathfrak{z}}^t\right)\right) \circ \left(p_{n^t+1}^t\right) \circ ... \circ \left(p_{|\mathbf{p}^t|}^t\right) \tag{6.94}$$
$$= \underline{\left(p_1^t, ..., p_{n^t-1}^t, \left(n^t, \overline{\mathfrak{z}}^t\right), p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\right)}. \tag{6.95}$$

Using these results, we prove that the criteria (6.67) hold. First, for $n^t + \overline{\mathfrak{d}}^t \geq |\mathbf{p}^t|$.

$$\psi^{-1}\left(\left[\left(q^{t+1}, \mathbf{start}, -1, n^{t+1}, |\mathbf{p}^t| + 1\right), \left(p_1^t, ..., p_{|\mathbf{p}^t|-1}^t, \left(|\mathbf{p}^t|, \overline{\mathfrak{z}}^t\right), \left(|\mathbf{p}^t| + 1, \_\right)\right)\right]\right) \tag{6.96}$$
$$= \left(q^{t+1}, x_1^t x_2^t \cdots x_{|\mathbf{p}^t|-1}^t \overline{\mathfrak{z}}^t \_\_^{\omega}, n^{t+1}\right) \tag{6.97}$$
$$= \left(q^{t+1}, x_1^t x_2^t \cdots x_{|\mathbf{p}^t|-1}^t \overline{\mathfrak{z}}^t \_^{\omega+1}, n^{t+1}\right) \tag{6.98}$$
$$= \left(q^{t+1}, x_1^t x_2^t \cdots x_{|\mathbf{p}^t|-1}^t \overline{\mathfrak{z}}^t \_^{\omega}, n^{t+1}\right) \tag{6.99}$$
$$= \underline{\alpha^{t+1}} \tag{6.100}$$

There was a particle added to $\mathbf{p}^t$. The length increased by one, and so did

$$\underline{\mathfrak{M}^{t+1}} = |\mathbf{p}^t| + 1 \underline{= |\mathbf{p}^{t+1}|}. \tag{6.101}$$

From $\overline{\mathfrak{d}}^t = 1$, $\mathfrak{m}^t \leq \mathfrak{M}^t$ we can follow

$$\mathfrak{m}^t \leq \mathfrak{M}^t \tag{6.102}$$
$$\rightarrow \; n^t \leq |\mathbf{p}^t| \tag{6.103}$$
$$\rightarrow \; n^t + 1 \leq |\mathbf{p}^t| + 1 \tag{6.104}$$
$$\rightarrow \; n^{t+1} \leq |\mathbf{p}^{t+1}| \tag{6.105}$$
$$\rightarrow \; \underline{\mathfrak{m}^{t+1}} \underline{\leq \mathfrak{M}^{t+1}}. \tag{6.106}$$

We directly see that

$$\mathfrak{d}^{t+1} = -1, \quad \Delta\mathfrak{q}^{t+1} = \textbf{start}. \tag{6.107}$$

The particles are not change except for the particle $p_{|\mathbf{p}^t|}$, where the symbol $\mathfrak{z}_{|\mathbf{p}^t|}$ changed and the index $\mathfrak{k}_{|\mathbf{p}^t|}$stayed the same. But a new particle was added at the end, i.e., at the index $|\mathbf{p}^t| + 1$, and it got the index $\mathfrak{k}_{|\mathbf{p}^t|+1} = |\mathbf{p}^t| + 1$. Hence, we can follow

$$\forall p_j^{t+1} \in \mathbf{p}^{t+1} : \mathfrak{k}_j^{t+1} = j. \tag{6.108}$$

Second for $n^t + \overline{\mathfrak{d}}^t \leq |\mathbf{p}^t|$:

$$\psi^{-1}\left(\left[\left(q^{t+1}, \textbf{start}, -1, n^{t+1}, |\mathbf{p}^t|\right), \left(p_1^t, ..., p_{n^t-1}^t, \left(n^t, \overline{\mathfrak{z}}^t\right), p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\right)\right]\right) \tag{6.109}$$
$$= \left(q^{t+1}, x_1^t x_2^t \cdots x_{n^t-1}^t \overline{\mathfrak{z}}^t x_{n^t+1}^t \cdots x_{|\mathbf{p}^t|}^t \; {}_-{}^{\omega}, n^{t+1}\right) \tag{6.110}$$
$$= \left(q^{t+1}, \mathbf{x}^{t+1} \; {}_-{}^{\omega}, n^{t+1}\right) \tag{6.111}$$
$$= \underline{\alpha^{t+1}}. \tag{6.112}$$

Since there was no particle added to $\mathbf{p}^t$, the length stays the same

$$\underline{\mathfrak{M}^{t+1}} = \mathfrak{M}^t = |\mathbf{p}^t| \underline{= |\mathbf{p}^{t+1}|}. \tag{6.113}$$

We can also follow

$$\underline{\mathfrak{m}^{t+1}} = n^{t+1} = n^t + \overline{\mathfrak{d}}^t \;\; \underline{\leq} \;\; |\mathbf{p}^t| = |\mathbf{p}^{t+1}| = \underline{\mathfrak{M}^{t+1}}, \tag{6.114}$$

and directly see that

$$\mathfrak{d}^{t+1} = -1, \quad \Delta\mathfrak{q}^{t+1} = \textbf{start}. \tag{6.115}$$

The particles are not changed except for the particle p_{n^t}, and the index $\mathfrak{k}_{n^t}$ was not changed. Hence,

$$\forall p_j^t \in \mathbf{p}^t : \mathfrak{k}_j^t = j \;\; \rightarrow \;\; \forall p_j^{t+1} \in \mathbf{p}^{t+1} : \mathfrak{k}_j^{t+1} = j. \tag{6.116}$$

This completes the induction step.

The last part to prove is that the particle method stops if and only if the Turing machine holds.

$$f(g^t) = \top \;\; \leftrightarrow \;\; q^t \in \{\textbf{accept}, \textbf{reject}\} \tag{6.117}$$

This is proven by

$$f\left(g^t\right) = \top \tag{6.118}$$
$$\leftrightarrow \mathfrak{q}^t \in \{\textbf{accept}, \; \textbf{reject}\} \tag{6.119}$$
$$\xleftrightarrow{\psi^{-1}\left(\left[g^t, \mathbf{p}^t\right]\right) = \alpha^t} q^t \in \{\textbf{accept}, \; \textbf{reject}\}. \tag{6.120}$$

□

6.4 Turing Powerfulness of Particle Methods Under a Second Set of Constraints

Theorem 4 (Turing powerfulness of particle methods under a second set of constraints)**.** *Particle methods are Turing powerful under the constraints:*

1. *P and G are finite*
$$|P| < \infty, \quad |G| < \infty, \tag{6.121}$$

2. *the interact function i is a pull interaction (just the first particle p_j is changed)*
$$i\left(g, p_j, p_k\right) = \left(\overline{p}_j, p_k\right), \tag{6.122}$$

3. *i is independent of previous interactions*
$${}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) = {}_1i_g(p_j, p_k), \tag{6.123}$$

4. *i is order independent*
$${}_1i_g({}_1i_g(p_j, p_k), p_{k'}) = {}_1i_g({}_1i_g(p_j, p_{k'}), p_k), \tag{6.124}$$

5. *the evolve function e is order-independent regarding g*
$${}_1e\left({}_1e\left(g, p'\right), p''\right) = {}_1e\left({}_1e\left(g, p''\right), p'\right), \tag{6.125}$$

6. *e is independent of previous evolutions regarding p*
$${}_2e\left(g, p'\right) = {}_2e\left({}_1e\left(g, p''\right), p'\right), \tag{6.126}$$

7. *u is independent of g and the values of $p_j \in \mathbf{p}$*
$$u([g, \mathbf{p}], j) = (k \in \{1, ..., |\mathbf{p}|\} : \Omega(j, k) = \top), \tag{6.127}$$

8. *u is independent of previous interactions*
$$u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^I_{(g,k')}} (k'')], j), \tag{6.128}$$

9. *$e, i, f, \overset{\circ}{e}$ have a time complexity bound by a constant,*
$$\tau_e, \tau_i, \tau_u, \tau_f, \tau_{\overset{\circ}{e}} \in \mathcal{O}(1), \tag{6.129}$$

   and

10. *$e, i, u, f, \overset{\circ}{e}$ have a space complexity bound by a constant,*
$$\varsigma_e, \varsigma_i, \varsigma_u, \varsigma_f, \varsigma_{\overset{\circ}{e}} \in \mathcal{O}(1). \tag{6.130}$$

Proof.
For the proof, we use the overbar notation

$$(\overline{\mathfrak{q}}_j, \overline{\mathfrak{z}}_j, \overline{\mathfrak{d}}_j) := \delta(\mathfrak{q}, \mathfrak{z}_j) \tag{6.131}$$

for the results of the transition function. Without loss of generality, we assume a strict total order $<$ on the set of states Q of the Turing machine, where

$$\forall q \in Q \backslash \{\textbf{start}\} : \textbf{start} < q \tag{6.132}$$

We construct a particle method that fulfills the constraints and emulates an arbitrary Turing machine as defined in section 6.2. Then we prove that this is the case.

Constructed Particle Method for a Turing Machine

The constructed particle method that resembles a Turing machine and fulfills the constraints (6.121) to (6.130) is:

$$p := (\mathfrak{z}, \mathfrak{h}, \Delta\mathfrak{h}, \mathfrak{o}, \Delta\mathfrak{o}, \mathfrak{a}) \quad \text{for } p \in P := \underbrace{\Gamma}_{\mathfrak{z}\ \in} \times \underbrace{\{-1,0,1\}}_{\mathfrak{h}\ \in} \times \underbrace{\{0,1\}}_{\Delta\mathfrak{h}\ \in} \times \underbrace{\{-1,1\}}_{\mathfrak{o},\ \Delta\mathfrak{o}\ \in}{}^2 \times \underbrace{\{0,1\}}_{\mathfrak{a}\ \in} \tag{6.133}$$

$$g := (\mathfrak{q}, \Delta\mathfrak{q}) \quad \text{for} \quad g \in G := Q \times Q \tag{6.134}$$

$$u([g, \mathbf{p}], j) := (k \in (1, ..., |\mathbf{p}|) : k = j - 1 \vee k = j + 1) \tag{6.135}$$

$$f(g) := (\mathfrak{q} = \textbf{reject} \ \vee \mathfrak{q} = \textbf{accept}\}) \tag{6.136}$$

$$i(g, p_j, p_k) := \tag{6.137}$$

$$\begin{cases} \left(\left(\mathfrak{z}_j, \mathfrak{h}_j, \underbrace{1}_{\Delta\mathfrak{h}_j}, \mathfrak{o}_j, \underbrace{\max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k)}_{\Delta\mathfrak{o}_j}, \mathfrak{a}_j\right),\ p_k\right) & \text{if} \quad \mathfrak{h}_k = 1 \wedge ((\mathfrak{h}_j = -1 \wedge \overline{\mathfrak{d}}_k \neq \mathfrak{o}_k) \\ & \qquad \vee (\mathfrak{h}_j = 0 \wedge \overline{\mathfrak{d}}_k = \mathfrak{o}_k)) \\ \left(\left(\mathfrak{z}_j, \mathfrak{h}_j, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \Delta\mathfrak{o}_j, \underbrace{1}_{\mathfrak{a}_j}\right),\ p_k\right) & \text{if} \quad \mathfrak{h}_j = 1 \wedge ((\mathfrak{h}_k = -1 \wedge \overline{\mathfrak{d}}_j \neq \mathfrak{o}_j) \\ & \qquad \vee (\mathfrak{h}_k = 0 \wedge \overline{\mathfrak{d}}_j = \mathfrak{o}_j)) \\ (p_j,\ p_k) & \text{else} \end{cases} \tag{6.138}$$

$$
e(g, p_j) :=
\begin{cases}
\Big(g, \Big(\big(\mathfrak{z}_j, \underbrace{1}_{\mathfrak{h}_j}, \underbrace{0}_{\Delta\mathfrak{h}_j}, \underbrace{\Delta\mathfrak{o}_j}_{\mathfrak{o}_j}, \underbrace{-1}_{\Delta\mathfrak{o}_j}, \underbrace{0}_{\mathfrak{a}_j}\big)\Big)\Big) & \text{if } \Delta\mathfrak{h}_j = 1 \\
\Big(g, \Big(\big(\mathfrak{z}_j, \underbrace{0}_{\mathfrak{h}_j}, \underbrace{0}_{\Delta\mathfrak{h}_j}, \underbrace{-1}_{\mathfrak{o}_j}, \underbrace{-1}_{\Delta\mathfrak{o}_j}, \underbrace{0}_{\mathfrak{a}_j}\big)\Big)\Big) & \text{if } \Delta\mathfrak{h}_j = 0 \wedge \mathfrak{h}_j = -1 \\
\Big(\Big(\mathfrak{q}, \underbrace{\max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}_j)}_{\Delta\mathfrak{q}}\Big), \Big(\big(\underbrace{\overline{\mathfrak{z}}_j}_{\mathfrak{z}_j}, \underbrace{-1}_{\mathfrak{h}_j}, \underbrace{0}_{\Delta\mathfrak{h}_j}, \underbrace{-1}_{\mathfrak{o}_j}, \underbrace{-1}_{\Delta\mathfrak{o}_j}, \underbrace{0}_{\mathfrak{a}_j}\big)\Big)\Big) & \text{if } \Delta\mathfrak{h}_j = 0 \wedge \mathfrak{h}_j = 1 \wedge \mathfrak{a}_j = 1 \\
\Big(\Big(\mathfrak{q}, \underbrace{\max(\Delta\mathfrak{q}, \overline{\mathfrak{q}}_j)}_{\Delta\mathfrak{q}}\Big), \Big(\big(\underbrace{\overline{\mathfrak{z}}_j}_{\mathfrak{z}_j}, \underbrace{-1}_{\mathfrak{h}_j}, \underbrace{0}_{\Delta\mathfrak{h}_j}, \underbrace{-1}_{\mathfrak{o}_j}, \underbrace{-1}_{\Delta\mathfrak{o}_j}, \underbrace{0}_{\mathfrak{a}_j}\big), & \text{if } \Delta\mathfrak{h}_j = 0 \wedge \mathfrak{h}_j = 1 \wedge \mathfrak{a}_j = 0 \\
\big(\underbrace{-}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{\overline{\mathfrak{d}}_j}_{\mathfrak{o}}, \underbrace{0}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\big)\Big)\Big) & \\
\Big(g, (p_j)\Big) & \text{else}
\end{cases}
\tag{6.139}
$$

$$
\overset{\circ}{e}(g) := \Big(\underbrace{\Delta\mathfrak{q}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.140}
$$

This particle method resembles an arbitrary Turing machine regarding the definition of Kozen (sec. 6.2). Each particle mimics one cell of the tape. One particle carries the information where the read-write head of the Turing machine is, and the global variable holds the state of the Turing machine. A constant time complexity limits all functions except the neighborhood function. Hence, it is not possible to do calculations with indices. Indices are growing with the number of particles. To circumvent indices, we use the information where the read-write head was in the particle methods state before and in which direction it went. The information where the head was and is, is stored in the particle property $\mathfrak{h}$. If $\mathfrak{h} = 1$, then there is the head. If $\mathfrak{h} = -1$, then there was the head in the particle methods state before, and $\mathfrak{h} = 0$ else. The property $\Delta\mathfrak{h}$ stores new information about the position of the head to render the interact function i independent of previous interactions. This information is only on one particle. The direction where the head went is stored in $\mathfrak{o}$ of the particle with $\mathfrak{h} = -1$. If $\mathfrak{o} = -1$, the head went left. If $\mathfrak{o} = 1$, it went right. The property $\Delta\mathfrak{o}$ stores the information where the head goes. This information is only on the particle where the head will be next. The property $\Delta\mathfrak{o}$ helps to render the interact function independent of previous interactions.

The interact function i is responsible for exchanging information about the read-write head. Hence, the interaction is not the identity only if the particle with the head $\mathfrak{h} = 1$ interacts with the particle where the head goes. The information about where the head was and is and where it went helps to determine if it goes back where it came from or if it moves on without knowing the indices. Suppose the head tries to move on beyond the particle tuple, then the flag $\mathfrak{a}$ stays 0 for the particle with the head $\mathfrak{h} = 1$. This information is then used in the evolve function to create a new particle. The max function

for determining the property $\Delta\mathfrak{o}$ keeps the interact-function order-independent in general. For a Turing-machine-related particle methods instance, the max function has no influence.

The evolve function e is responsible for computing all properties for the new state. Hence, it sets the particle that has as next the head $\Delta\mathfrak{h} = 1$ to having the head $\mathfrak{h} = 1$, a particle that had the head before and did not get it back $\Delta\mathfrak{h} = 0, \mathfrak{h} = -1$ to a standard particle without head information, a particle that has the head and was able to hand it to the next particle $\Delta\mathfrak{h} = 0, \mathfrak{h} = 1, \mathfrak{a} = 1$ to a particle that had the head, a particle that has the head and was not able to hand it to the next particle $\Delta\mathfrak{h} = 0, \mathfrak{h} = 1, \mathfrak{a} = 0$ to a particle that had the head and creates a new particle that has the head, and the rest of the particles stay the same. The evolve function also changes the global variable state storage $\Delta\mathfrak{q}$ to the new state of the Turing machine $\overline{\mathfrak{h}}$ and the symbol of the particle $\mathfrak{z}$, both happen just if the particle has the head. The storage $\Delta\mathfrak{q}$ renders the evolve function independent of previous evolutions regarding p, and the max function guarantees the order independence regarding g. For a Turing-machine-related particle methods instance, both do not have an influence.

Finally, the evolve function of the global variable tests the state of the Turing machine $\mathfrak{q}$ to the value of the storage $\Delta\mathfrak{q}$ and resets the storage $\Delta\mathfrak{q}$ to the minimal value **start**.

Particle Method of Turing Machine Fulfills Constrains

We prove that the particle method fulfills all constraints from (6.121) to (6.130).

The condition that P and G are finite (6.121) is fulfilled since the Cartesian product of two finite sets is a finite set. All sets are finite, inducing Γ and Q. Hence, $|P| < \infty$ and $|G| < \infty$.

The interact function i is defined so that the second particle stays the same for each case. Therefore it is a pull interaction (6.122).

The interact function i can only change $\Delta\mathfrak{h}, \Delta\mathfrak{o}, \mathfrak{a}$ and only depends on $\mathfrak{z}, \mathfrak{h}, \mathfrak{o}$ from the second particle, i.e., it depends only on properties that are not changed during the interactions. Hence,

$$
\begin{aligned}
{}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) &= {}_1i_g(p_j, (\mathfrak{z}_k, \mathfrak{h}_k, \Delta\mathfrak{h}'_k, \mathfrak{o}_k, \Delta\mathfrak{o}'_k, \mathfrak{a}'_k)) && (6.141)\\
&= {}_1i_g(p_j, (\mathfrak{z}_k, \mathfrak{h}_k, \Delta\mathfrak{h}_k, \mathfrak{o}_k, \Delta\mathfrak{o}_k, \mathfrak{a}_k)) && (6.142)\\
&= \underline{{}_1i_g(p_j, p_k)}. && (6.143)
\end{aligned}
$$

Therefore, i is independent of previous interactions (6.123).

We know that the interact function i does not change any property of p_j and p_k that is used to change properties in the interact function i. Therefore, it is sufficient to focus on how the properties are changed to prove the order independence of i (6.124). For readability, we leaf out the conditions they can be handled similarly as in (6.151) to

(6.157).

$$
\begin{aligned}
&{}_1i({}_1i(g, p_j, p_k), p_{k'}) && (6.144)\\
&= {}_1i\left(\left(\mathfrak{z}_j, \mathfrak{h}_j, \begin{cases}\Delta\mathfrak{h}_j\\ 1\end{cases}, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k)\end{cases}, \begin{cases}\mathfrak{o}_j\\ 1\end{cases}\right), p_{k'}\right) && (6.145)\\
&= \left(\mathfrak{z}_j, \mathfrak{h}_j, \begin{cases}\begin{cases}\Delta\mathfrak{h}_j\\ 1\end{cases}\\ 1\end{cases}, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \begin{cases}\begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k)\end{cases}\\ \max\left(\begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k)\end{cases}, \overline{\mathfrak{d}}_{k'}\right)\end{cases}, \begin{cases}\begin{cases}\mathfrak{o}_j\\ 1\end{cases}\\ 1\end{cases}\right) && (6.146)\\
&= \left(\mathfrak{z}_j, \mathfrak{h}_j, \begin{cases}\begin{cases}\Delta\mathfrak{h}_j\\ 1\end{cases}\\ 1\end{cases}, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k)\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_{k'})\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_k, \overline{\mathfrak{d}}_{k'})\end{cases}, \begin{cases}\begin{cases}\mathfrak{o}_j\\ 1\end{cases}\\ 1\end{cases}\right) && (6.147)\\
&= \left(\mathfrak{z}_j, \mathfrak{h}_j, \begin{cases}\begin{cases}\Delta\mathfrak{h}_j\\ 1\end{cases}\\ 1\end{cases}, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \begin{cases}\begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_{k'})\end{cases}\\ \max\left(\begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_{k'})\end{cases}, \overline{\mathfrak{d}}_k\right)\end{cases}, \begin{cases}\begin{cases}\mathfrak{o}_j\\ 1\end{cases}\\ 1\end{cases}\right) && (6.148)\\
&= {}_1i\left(\left(\mathfrak{z}_j, \mathfrak{h}_j, \begin{cases}\Delta\mathfrak{h}_j\\ 1\end{cases}, \Delta\mathfrak{h}_j, \mathfrak{o}_j, \begin{cases}\Delta\mathfrak{o}_j\\ \max(\Delta\mathfrak{o}_j, \overline{\mathfrak{d}}_{k'})\end{cases}, \begin{cases}\mathfrak{o}_j\\ 1\end{cases}\right), p_k\right) && (6.149)\\
&= \underline{{}_1i({}_1i(g, p_j, p_{k'}), p_k)} && (6.150)
\end{aligned}
$$

We prove that the evolve function e is order-independent regarding g (6.125) by

$$
\begin{aligned}
&{}_1e({}_1e(g,p'),p'') && (6.151)\\
&= {}_1e\left(\begin{cases}(\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}')) & \text{if } \mathfrak{h}'=1\\ g & \text{else}\end{cases},p''\right) && (6.152)\\
&= \begin{cases}\begin{cases}(\mathfrak{q},\max(\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'),\overline{\mathfrak{q}}'') & \text{if } \mathfrak{h}'=1\\ (\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'')) & \text{else}\end{cases} & \text{if } \mathfrak{h}''=1\\ \begin{cases}(\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}')) & \text{if } \mathfrak{h}'=1\\ g & \text{else}\end{cases} & \text{else}\end{cases} && (6.153)\\
&= \begin{cases}(\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}',\overline{\mathfrak{q}}'') & \text{if } \mathfrak{h}'=1 \ \wedge\ \mathfrak{h}''=1\\ (\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'')) & \text{if } \mathfrak{h}'\neq 1 \wedge\ \mathfrak{h}''=1\\ (\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}')) & \text{if } \mathfrak{h}'=1 \ \wedge\ \mathfrak{h}''\neq 1\\ g & \text{else}\end{cases} && (6.154)\\
&= \begin{cases}\begin{cases}(\mathfrak{q},\max(\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'),\overline{\mathfrak{q}}'') & \text{if } \mathfrak{h}''=1\\ (\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}')) & \text{else}\end{cases} & \text{if } \mathfrak{h}'=1\\ \begin{cases}(\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'')) & \text{if } \mathfrak{h}''=1\\ g & \text{else}\end{cases} & \text{else}\end{cases} && (6.155)\\
&= {}_1e\left(\begin{cases}(\mathfrak{q},\max(\Delta\mathfrak{q},\overline{\mathfrak{q}}'')) & \text{if } \mathfrak{h}''=1\\ g & \text{else}\end{cases},p'\right) && (6.156)\\
&\underline{= {}_1e({}_1e(g,p''),p')}. && (6.157)
\end{aligned}
$$

The evolve function changes only $\Delta\mathfrak{q}$ in the global variable, but changes by the evolve function only depend on the property $\mathfrak{q}$ of the global variable, i.e., in the series of evolutions only properties are changed that do not influence the change of following evolutions, hence the evolve function is independent of previous evolutions (6.126)

$$
\begin{aligned}
{}_2e({}_1e(g,p''),p') &= {}_2e((\mathfrak{q},\Delta\mathfrak{q}''),p') && (6.158)\\
&= {}_2e((\mathfrak{q},\Delta\mathfrak{q}),p') && (6.159)\\
&= \underline{{}_2e(g,p')}. && (6.160)
\end{aligned}
$$

The neighborhood function u only uses $|\mathbf{p}|$ and j to determine the neighbor indices k. Hence,

$$u([g,\mathbf{p}],j) = (k \in \{1,...,|\mathbf{p}|\} : \Omega(j,k) = \top) \tag{6.161}$$

with

$$\Omega(j,k) := (k = j-1 \vee k = j+1). \tag{6.162}$$

Therefore, u is independent of g and the values of $p_j \in \mathbf{p}$ (6.127).

The interact function can not add or destroy particles. Therefore it can not change the number of particles in a tuple $|\mathbf{p} *_{\iota^I_{(g,k')}} (k'')| = |\mathbf{p}|$. Hence,

$$u([g,\mathbf{p} *_{\iota^I_{(g,k')}} (k'')],j) = u([g,\mathbf{p}],j). \tag{6.163}$$

Therefore, u is independent of previous interactions (6.128).

The functions $e, i, f, \overset{\circ}{e}$ consist of evaluations of basic comparisons, max-functions and the transition function of the Turing machine $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 1\}$. There are no loops or equivalent. Since G, P, Q, and Γ are finite so is δ and with it the time complexities of $e, i, f, \overset{\circ}{e}$ (6.129)

$$\tau_e, \tau_i, \tau_f, \tau_{\overset{\circ}{e}} \in \mathcal{O}(1), \tag{6.164}$$

and the space complexities of $e, i, u, f, \overset{\circ}{e}$ (6.130)

$$\varsigma_e, \varsigma_i, \varsigma_u, \varsigma_f, \varsigma_{\overset{\circ}{e}} \in \mathcal{O}(1). \tag{6.165}$$

The time complexity of the neighborhood function τ_u is in $\mathcal{O}(log(|\mathbf{p}^t|))$ if one considers $|\mathbf{p}^t|$ to be known.

Particle Method Emulates Arbitrary Turing Machine

To prove that the particle method emulates an arbitrary Turing machine, we need to translate the start configuration of a Turing machine (6.18) into a particle methods instance. Then we prove for all states of the particle method that the back translation is the corresponding configuration of the Turing machine. The last step is to show that the particle method stops when the Turing machine reaches an accept or reject state.
We define for this particle method the translation function as

$$\psi\left((q, \mathbf{x}\_^{\omega}, n)\right) \tag{6.166}$$

$$:= \left[(\underbrace{q}_{\mathfrak{q}}, \underbrace{start}_{\Delta\mathfrak{q}}), \begin{pmatrix} (\underbrace{x_1}_{\mathfrak{z}}, & \underbrace{0}_{\mathfrak{h}}, & \underbrace{0}_{\Delta\mathfrak{h}}, & \underbrace{-1}_{\mathfrak{o}}, & \underbrace{-1}_{\Delta\mathfrak{o}}, & \underbrace{0}_{\mathfrak{a}}) \\ \vdots & & & & & \\ (x_{n-1}, & 0, & 0, & -1, & -1, & 0) \\ (x_n, & 1, & 0, & -1, & -1, & 0) \\ (x_{n+1}, & -1, & 0, & -1, & -1, & 0) \\ \vdots & & & & & \\ (x_{|\mathbf{x}|} & 0, & 0, & -1, & -1, & 0) \end{pmatrix}^{\mathbf{T}} \right], \tag{6.167}$$

and the back translation as

$$\psi^{-1}([(\mathfrak{q}, \Delta\mathfrak{q}), (p_1, ..., p_{|\mathbf{p}|})]) := \left(\mathfrak{q}, \mathfrak{z}_1\mathfrak{z}_2 \cdots \mathfrak{z}_{|\mathbf{p}|}\_^{\omega},\ j : \mathfrak{h}_j = 1\right) \tag{6.168}$$

We see these translation functions are only copying values. Therefore, the translations do not carry any calculation of the Turing machine.

The instance of the particle method is defined by translating the start configuration $\alpha^1 = (\mathbf{start}, \mathbf{x}^1\_^{\omega}, 1)$. Hence,

$$[g^1, \mathbf{p}^1] := \psi(\alpha^1). \tag{6.169}$$

We need additional criteria to be fulfilled by each particle methods step so that the particle method works. In total, the following criteria are needed:

$$\begin{aligned}
&\psi^{-1}\left(\left[g^t, \mathbf{p}^t\right]\right) = \alpha^t = \left(q^t, \mathbf{x}^t_^{\omega}, n^t\right)\\
&\wedge \mathfrak{h}^t_{n^t} = 1 \ \wedge \ \mathfrak{h}^t_{n^t - \mathfrak{o}^t_{n^t}} = -1\\
&\wedge \forall j \in \left\{1, ..., |\mathbf{p}^t|\right\} \setminus \left\{n^t, n^t - \mathfrak{o}^t_{n^t}\right\} : \ \mathfrak{h}^t_j = 0\\
&\wedge \forall j \in \left\{1, ..., |\mathbf{p}^t|\right\} : \Delta\mathfrak{h}^t_j = 0 \ \wedge \Delta\mathfrak{o}^t_j = -1 \wedge \mathfrak{a}^t_j = 0\\
&\wedge \Delta q^t = \mathbf{start}.
\end{aligned} \tag{6.170}$$

We prove through induction that the back translation of all particle method states is the Turing machine's corresponding configuration and that the criteria (6.170) are fulfilled.

The base case is for the particle method instance and the start configuration.

$$\psi^{-1}\left([g^1, \mathbf{p}^1]\right) \tag{6.171}$$

$$= \psi^{-1}\left(\psi\left(\left(\mathbf{start}, \mathbf{x}^1\_^{\omega}, 1\right)\right)\right) \tag{6.172}$$

$$= \psi^{-1}\left(\left[(\underbrace{\mathbf{start}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}), \begin{pmatrix} (\underbrace{\vdash}_{\mathfrak{z}}, & \underbrace{1}_{\mathfrak{h}}, & \underbrace{0}_{\Delta\mathfrak{h}}, & \underbrace{-1}_{\mathfrak{o}}, & \underbrace{-1}_{\Delta\mathfrak{o}}, & \underbrace{0}_{\mathfrak{a}}) \\ (x^1_2, & -1, & 0, & -1, & -1, & 0) \\ (x^1_3, & 0, & 0, & -1, & -1, & 0) \\ \vdots & & & & & \\ (x^1_{|\mathbf{x}^1|} & 0, & 0, & -1, & -1, & 0) \end{pmatrix}^{\mathbf{T}}\right]\right) \tag{6.173}$$

$$= \left(\mathbf{start}, \vdash x^1_2 \cdots x^1_{|\mathbf{x}^1|}\_^{\omega}, 1\right) \tag{6.174}$$

$$= \left(\mathbf{start}, \mathbf{x}^1\_^{\omega}, 1\right) \tag{6.175}$$

$$= \underline{\alpha^1}. \tag{6.176}$$

The rest of the criteria (6.170) follow directly from (6.173).

For the induction step, we need to prove that if the criteria (6.170) are valid for $[g^t, \mathbf{p}^t]$, it is also true for $[g^{t+1}, \mathbf{p}^{t+1}] = s([g^t, \mathbf{p}^t])$. We divide the induction step into six cases.

First, we prove it for the case that the head is on the first cell/ particle, $\mathfrak{h}^t_1 = 1$. We use the criteria (6.170) to set up $[g^t, \mathbf{p}^t]$ and calculate from it $[g^{t+1}, \mathbf{p}^{t+1}]$. Since p^t_0 does not exist, $\mathbf{h}^t_2 = -1$ and $\mathfrak{o}^t_1 = -1$. From this and the definition of the Turing machine especially (6.11) follows that

$$\forall t \in \{1, ..., T\} : \delta(\mathfrak{q}^t, \mathfrak{z}^t_1) = \delta(\mathfrak{q}^t, \vdash) = (\overline{\mathfrak{q}}, \vdash, 1) =: (\overline{\mathfrak{q}}^t, \overline{\mathfrak{z}}^t_1, \overline{\mathfrak{d}}^t_1). \tag{6.177}$$

We calculate the interaction of p^t_1 with its neighbors

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{I}\times\mathrm{U}}_g(\mathbf{p}^t, 1) \tag{6.178}$$

$$= \mathbf{p}^t *_{\iota^{\mathrm{I}}_{(g^t,1)}} u([g^t, \mathbf{p}^t], 1) \tag{6.179}$$

$$= \mathbf{p}^t *_{\iota^{\mathrm{I}}_{(g^t,1)}} (2) \tag{6.180}$$

$$= \left(\left(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\right), p^t_2, ..., p^t_{|\mathbf{p}^t|}\right). \tag{6.181}$$

and the interactions of p_2^t with its neighbors

$$\tilde{\tilde{\mathbf{p}}}^t := \iota_g^{\mathrm{I}\times\mathrm{U}}(\tilde{\mathbf{p}}^t, 2) \tag{6.182}$$

$$= \tilde{\mathbf{p}}^t *_{\iota^{\mathrm{I}}_{(g^t,2)}} u([g^t, \mathbf{p}^t], 2) \tag{6.183}$$

$$= \tilde{\mathbf{p}}^t *_{\iota^{\mathrm{I}}_{(g^t,2)}} (1, 3). \tag{6.184}$$

The particle p_1 is the only particle with $\mathfrak{h}_1^t = 1$ and p_2 with $\mathfrak{h}_2^t = -1$. Hence, $\mathfrak{h}_3^t = 0$ and the interact function is for p_2 with p_3 the identity ($i(g, p_2, p_3) = (p_2, p_3)$). Therefore,

$$\tilde{\tilde{\mathbf{p}}}^t = \Bigg(\Big(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big),$$
$$\Big(\underbrace{x_2^t}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \mathfrak{o}_2, \underbrace{\max\left(-1, \overline{\mathfrak{d}}_1^t\right)}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), p_3^t, ..., p_{|\mathbf{p}^t|}^t\Bigg) \tag{6.185}$$

$$= \Bigg(\Big(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big),$$
$$\Big(\underbrace{x_2^t}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \mathfrak{o}_2, \underbrace{1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), p_3^t, ..., p_{|\mathbf{p}^t|}^t\Bigg) \tag{6.186}$$

For all other particles, the interact function is the identity, only $\mathfrak{h}_1^t = 1$ and $\forall j \in \{3, ..., |\mathbf{p}^t|\} : 1 \notin u([g^t, \mathbf{p}^t], j)$. Hence,

$$\iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) = \tilde{\tilde{\mathbf{p}}}^t. \tag{6.187}$$

Next is the evolution step. The evolve function e is the identity for the particles except if $\mathfrak{h} = 1$ this is only true for p_1^t or if $\Delta\mathfrak{h} = 1$ this is only true for p_2^t or if $\mathfrak{h} = -1$ only true for p_2^t. e is also the identity for the global variable except if $\mathfrak{h} = 1$. Hence, we can reduce ϵ^{N} to

$$\epsilon^{\mathrm{N}}([g^t, \tilde{\tilde{\mathbf{p}}}^t]) = \left[{}_1e(g^t, p_1^t), {}_2e(g, p_1^t) \circ {}_2e(g, p_2^t) \circ (p_3^t, ..., p_{|\mathbf{p}^t|}^t)\right] \tag{6.188}$$

$$= \Bigg[\Big(\mathfrak{q}^t, \underbrace{\max\left(\mathbf{start}, \overline{\mathfrak{q}}_1^t\right)}_{\Delta\mathfrak{q}}\Big), \Bigg(\Big(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg)$$
$$\circ \Bigg(\Big(\underbrace{x_2^t}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg) \circ \left(p_3^t, ..., p_{|\mathbf{p}^t|}^t\right)\Bigg] \tag{6.189}$$

$$= \Bigg[\Big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}_1^t}_{\Delta\mathfrak{q}}\Big), \Bigg(\Big(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg)$$
$$\circ \Bigg(\Big(\underbrace{x_2^t}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg) \circ \left(p_3^t, ..., p_{|\mathbf{p}^t|}^t\right)\Bigg] \tag{6.190}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}_1^t}_{\Delta\mathfrak{q}}\big)\Big) := \Big(\underbrace{\overline{\mathfrak{q}}_1^t}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.191}$$

Hence,

$$\begin{aligned} s\left(\left[g^t, \mathbf{p}^t\right]\right) = \Bigg[& \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big), \Big(\Big(\underbrace{\vdash}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \\ & \circ \Big(\Big(\underbrace{x_2^t}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \circ \Big(p_3^t, ..., p_{|\mathbf{p}^t|}^t\Big) \Bigg] \qquad (6.192) \\ = & \left[g^{t+1}, \mathbf{p}^{t+1}\right] \qquad (6.193) \end{aligned}$$

We need to prove that the criteria (6.170) are true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$\begin{aligned} & \psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\ & = \Big(q^{t+1}, \vdash x_2^t \cdots x_{|\mathbf{p}^t|}^t \_^{\omega}, 2\Big) \\ & = \left(q^{t+1}, \mathbf{x}^{t+1} \_^{\omega}, n^{t+1}\right) \\ & = \underline{\alpha^{t+1}} \end{aligned} \tag{6.194}$$

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}_{n^{t+1}}^{t+1} = \mathfrak{h}_2^{t+1} = \underline{1}, \tag{6.195}$$

$$\mathfrak{h}_{n^{t+1}-\mathfrak{o}_{n^{t+1}}^{t+1}}^{t+1} = \mathfrak{h}_{2-\mathfrak{o}_2^{t+1}}^{t+1} = \mathfrak{h}_{2-1}^{t+1} = \underline{-1}, \tag{6.196}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \Big\{\underbrace{n^{t+1}}_{=2}, \underbrace{n^{t+1} - \mathfrak{o}_{n^{t+1}}^{t+1}}_{=1}\Big\} : \ \mathfrak{h}_j^{t+1} = 0, \tag{6.197}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}_j^{t+1} = 0 \ \wedge \Delta\mathfrak{o}_j^{t+1} = -1 \wedge \mathfrak{a}_j^{t+1} = 0, \tag{6.198}$$

$$\Delta q^{t+1} = \textbf{start}. \tag{6.199}$$

Second, we prove it for the case $\mathfrak{h}_{n^t}^t = 1$, $\mathfrak{h}_{n^t-1}^t = -1$, and $\overline{\mathfrak{d}}_{n^t}^t = 1$ where $n^t \in \{2, ..., |\mathbf{p}^t| - 1\}$. Using the criteria (6.170) lead to

$$\left[g^t, \mathbf{p}^t\right] = \left[(\underbrace{q^t}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}), \begin{pmatrix} (\underbrace{x_1^t}_{\mathfrak{z}}, & \underbrace{0}_{\mathfrak{h}}, & \underbrace{0}_{\Delta\mathfrak{h}}, & \underbrace{-1}_{\mathfrak{o}}, & \underbrace{-1}_{\Delta\mathfrak{o}}, & \underbrace{0}_{\mathfrak{a}}) \\ \vdots & & & & & \\ (x_{n^t-2}^t, & 0, & 0, & -1, & -1, & 0) \\ (x_{n^t-1}^t, & -1, & 0, & -1, & -1, & 0) \\ (x_{n^t}, & 1, & 0, & 1, & -1, & 0) \\ (x_{n^t+1}^t, & 0, & 0, & -1, & -1, & 0) \\ \vdots & & & & & \\ (x_{|\mathbf{x}^t|}^t & 0, & 0, & -1, & -1, & 0) \end{pmatrix}^{\mathbf{T}} \right] \tag{6.200}$$

We use again

$$\delta(\mathfrak{q}^t, \mathfrak{z}^t_{n^t}) =: (\overline{\mathfrak{q}}^t, \overline{\mathfrak{z}}^t_{n^t}, \overline{\mathfrak{d}}^t_{n^t}). \tag{6.201}$$

The particle p_{n^t} is the only particle with $\mathfrak{h}^t_{n^t} = 1$ and p_{n^t-1} with $\mathfrak{h}^t_{n^t-1} = -1$. Since $\overline{\mathfrak{d}}^t_{n^t} = 1$ and $\overline{\mathfrak{o}}^t_{n^t} = 1$ the interact function i is the identity except for $i(g, p_{n^t}, p_{n^t+1})$ and $i(g, p_{n^t+1}, p_{n^t})$. Therefore,

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) \tag{6.202}$$

$$= \Big(p^t_1, ..., p^t_{n^t-1}, \Big(\underbrace{x^t_{n^t}}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big),$$

$$\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{\max(-1,1)}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\Big) \tag{6.203}$$

$$= \Big(p^t_1, ..., p^t_{n^t-1}, \Big(\underbrace{x^t_{n^t}}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big),$$

$$\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\Big). \tag{6.204}$$

The next is the evolution step. The evolve function e is the identity for all particles except if $\mathfrak{h} = 1$ this is only true for $\tilde{p}^t_{n^t}$ or if $\Delta\mathfrak{h} = 1$, only true for $\tilde{p}^t_{n^t+1}$, or if $\mathfrak{h} = -1$ only true for $p^t_{n^t-1}$. The evolve function e is the identity for the global variable except if $\mathfrak{h} = 1$.

Hence, we can reduce ϵ^{N} to

$$\epsilon^{\mathrm{N}}([g^t, \tilde{\mathbf{p}}^t]) = [{}_1e(g^t, p^t_1), (p^t_1, ..., p^t_{n^t-2}) \circ {}_2e(g, p^t_{n^t-1})$$
$$\circ\, {}_2e(g, \tilde{p}^t_{n^t}) \circ {}_2e(g, \tilde{p}^t_{n^t+1}) \circ (p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|})] \tag{6.205}$$

$$= \Bigg[\Big(\mathfrak{q}^t, \underbrace{\max\big(\mathbf{start}, \overline{\mathfrak{q}}^t_{n^t}\big)}_{\Delta\mathfrak{q}}\Big),\ (p^t_1, ..., p^t_{n^t-2})$$

$$\circ \Big(\Big(\underbrace{x^t_{n^t-1}}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big)$$

$$\circ \Bigg(\Big(\underbrace{\overline{\mathfrak{z}}^t_{n^t}}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg)$$

$$\circ \Big(\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \circ \Big(p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\Big)\Bigg]. \tag{6.206}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\Delta\mathfrak{q}}\big)\Big) := \Big(\underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.207}$$

Hence,

$$
\begin{aligned}
s\left(\left[g^t, \mathbf{p}^t\right]\right) &= \left[g^{t+1}, \mathbf{p}^{t+1}\right] && (6.208)\\
&= \Bigg[\Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big),\ (p_1^t, ..., p_{n^t-2}^t) \\
&\quad \circ \Big(\Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}}, \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \\
&\quad \circ \left(\Big(\underbrace{\bar{\mathfrak{z}}_{n^t}^t}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\right) \\
&\quad \circ \Big(\Big(\underbrace{x_{n^t+1}^t}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \circ \left(p_{n^t+2}^t, ..., p_{|\mathbf{p}^t|}^t\right)\Bigg] && (6.209)
\end{aligned}
$$

We need to prove again that the criteria (6.170) is true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$
\begin{aligned}
&\psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\
&= \left(q^{t+1}, x_1^t \cdots x_{n^t-1}^t \bar{\mathfrak{z}}_{n^t}^t x_{n^t+1}^t \cdots x_{|\mathbf{p}^t|}^t \_^{\omega}, n^t+1\right) \\
&= \left(q^{t+1}, \mathbf{x}^{t+1} \_^{\omega}, n^{t+1}\right) \\
&= \underline{\alpha^{t+1}}
\end{aligned} \tag{6.210}
$$

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}_{n^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1}^{t+1} = 1, \tag{6.211}$$

$$\mathfrak{h}_{n^{t+1}-\mathfrak{o}_{n^{t+1}}^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1-\mathfrak{o}_{n^t+1}^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1-1}^{t+1} = -1, \tag{6.212}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \Big\{\underbrace{n^{t+1}}_{=n^t+1}, \underbrace{n^{t+1} - \mathfrak{o}_{n^{t+1}}^{t+1}}_{=n^t}\Big\} :\ \mathfrak{h}_j^{t+1} = 0, \tag{6.213}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}_j^{t+1} = 0\ \wedge \Delta\mathfrak{o}_j^{t+1} = -1 \wedge \mathfrak{a}_j^{t+1} = 0, \tag{6.214}$$

$$\Delta q^{t+1} = \textbf{start}. \tag{6.215}$$

Third, we prove it for the case $\mathfrak{h}_{n^t}^t = 1$, $\mathfrak{h}_{n^t-1}^t = -1$, and $\delta(\mathfrak{q}^t, \mathfrak{z}_{n^t}^t) =: (\bar{\mathfrak{q}}^t, \bar{\mathfrak{z}}_{n^t}^t, \bar{\mathfrak{d}}_{n^t}^t) = (q^{t+1}, \bar{\mathfrak{z}}_{n^t}^t, -1)$ where $n^t \in \{2, ..., |\mathbf{p}^t|\}$. This results in the same state $[g^t, \mathbf{p}^t]$ as in (6.200).

The particle p_{n^t} is the only particle with $\mathfrak{h}_{n^t}^t = 1$ and p_{n^t-1} with $\mathfrak{h}_{n^t-1}^t = -1$. Since $\bar{\mathfrak{d}}_{n^t}^t = -1$ and $\bar{\mathfrak{o}}_{n^t}^t = 1$, the interact function is the identity except for $i(g, p_{n^t-1}, p_{n^t})$ and

$i(g, p_{n^t}, p_{n^t-1})$. Therefore,

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) \tag{6.216}$$

$$= \Big(p_1^t, ..., p_{n^t-2}^t, \Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{\max(-1,-1)}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big),$$

$$\Big(\underbrace{x_{n^t}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big), p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\Big) \tag{6.217}$$

$$= \Big(p_1^t, ..., p_{n^t-2}^t, \Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big),$$

$$\Big(\underbrace{x_{n^t}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big), p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\Big). \tag{6.218}$$

The next is the evolution step. The evolve function e is the identity for all particles except if $\mathfrak{h} = 1$ this is only true for $\tilde{p}_{n^t}^t$ or if $\Delta\mathfrak{h} = 1$,only true for $\tilde{p}_{n^t-1}^t$, or if $\mathfrak{h} = -1$ only true for $\tilde{p}_{n^t-1}^t$, too. The evolve function e is the identity for the global variable except if $\mathfrak{h} = 1$. Hence, we can reduce ϵ^{N} to

$$\epsilon^{\mathrm{N}}([g^t, \tilde{\mathbf{p}}^t]) = [{}_1e(g^t, p_1^t), (p_1^t, ..., p_{n^t-2}^t) \circ {}_2e(g, \tilde{p}_{n^t-1}^t)$$
$$\circ\, {}_2e(g, \tilde{p}_{n^t}^t) \circ (p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t)] \tag{6.219}$$

$$= \Big[\Big(\mathfrak{q}^t, \underbrace{\max\big(\mathbf{start}, \overline{\mathfrak{q}}_{n^t}^t\big)}_{\Delta\mathfrak{q}}\Big),\ (p_1^t, ..., p_{n^t-2}^t)$$

$$\circ \Big(\Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big)$$

$$\circ \Bigg(\Big(\underbrace{\overline{\mathfrak{z}}_{n^t}^t}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Bigg) \circ \Big(p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\Big)\Big]. \tag{6.220}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}_{n^t}^t}_{\Delta\mathfrak{q}}\big)\Big) := \Big(\underbrace{\overline{\mathfrak{q}}_{n^t}^t}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.221}$$

Hence,

$$
\begin{aligned}
s\left(\left[g^t, \mathbf{p}^t\right]\right) &= \left[g^{t+1}, \mathbf{p}^{t+1}\right] && (6.222)\\
&= \Bigg[\Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big),\ (p_1^t, ..., p_{n^t-2}^t) \\
&\quad \circ \left(\left(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\right)\right) \\
&\quad \circ \left(\left(\underbrace{\bar{\mathfrak{z}}_{n^t}^t}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\right)\right) \circ \left(p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\right)\Bigg]. && (6.223)
\end{aligned}
$$

We need to prove again that the criteria (6.170) are true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$
\begin{aligned}
&\psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\
&= \left(q^{t+1}, x_1^t \cdots x_{n^t-1}^t \bar{\mathfrak{z}}_{n^t}^t x_{n^t+1}^t \cdots x_{|\mathbf{p}^t|}^t \_^{\omega}, n^t - 1\right) \\
&= \left(q^{t+1}, \mathbf{x}^{t+1} \_^{\omega}, n^{t+1}\right) \\
&= \underline{\alpha^{t+1}}
\end{aligned} \tag{6.224}
$$

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}_{n^{t+1}}^{t+1} = \mathfrak{h}_{n^t-1}^{t+1} = 1, \tag{6.225}$$

$$\mathfrak{h}_{n^{t+1}-\mathfrak{o}_{n^{t+1}}^{t+1}}^{t+1} = \mathfrak{h}_{n^t-1-\mathfrak{o}_{n^t-1}^{t+1}}^{t+1} = \mathfrak{h}_{n^t-1-(-1)}^{t+1} = -1, \tag{6.226}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \Big\{\underbrace{n^{t+1}}_{=n^t-1}, \underbrace{n^{t+1} - \mathfrak{o}_{n^{t+1}}^{t+1}}_{=n^t}\Big\} :\ \mathfrak{h}_j^{t+1} = 0, \tag{6.227}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}_j^{t+1} = 0\ \wedge \Delta\mathfrak{o}_j^{t+1} = -1 \wedge \mathfrak{a}_j^{t+1} = 0, \tag{6.228}$$

$$\Delta q^{t+1} = \textbf{start}. \tag{6.229}$$

Fourth, we prove it for the case $\mathfrak{h}_{n^t}^t = 1$, $\mathfrak{h}_{n^t+1}^t = -1$, and $\delta(\mathfrak{q}^t, \mathfrak{z}_{n^t}^t) =: (\bar{\mathfrak{q}}^t, \bar{\mathfrak{z}}_{n^t}^t, \bar{\mathfrak{d}}_{n^t}^t) = (q^{t+1}, \bar{\mathfrak{z}}_{n^t}^t, 1)$ where $n^t \in \{2, ..., |\mathbf{p}^t| - 1\}$. Using the criteria (6.170) lead to

$$
\left[g^t, \mathbf{p}^t\right] = \left[\Big(\underbrace{q^t}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big), \begin{pmatrix}
(\underbrace{x_1^t}_{\mathfrak{z}}, & \underbrace{0}_{\mathfrak{h}}, & \underbrace{0}_{\Delta\mathfrak{h}}, & \underbrace{-1}_{\mathfrak{o}}, & \underbrace{-1}_{\Delta\mathfrak{o}}, & \underbrace{0}_{\mathfrak{a}}) \\
\vdots & & & & & \\
(x_{n^t-1}^t, & 0, & 0, & -1, & -1, & 0) \\
(x_{n^t}, & 1, & 0, & -1, & -1, & 0) \\
(x_{n^t+1}^t, & -1, & 0, & -1, & -1, & 0) \\
(x_{n^t+2}^t, & 0, & 0, & -1, & -1, & 0) \\
\vdots & & & & & \\
(x_{|\mathbf{x}^t|}^t & 0, & 0, & -1, & -1, & 0)
\end{pmatrix}^{\mathbf{T}}\right] \tag{6.230}
$$

The particle p_{n^t} is the only particle with $\mathfrak{h}^t_{n^t} = 1$ and p_{n^t+1} with $\mathfrak{h}^t_{n^t+1} = -1$. Since $\overline{\mathfrak{d}}^t_{n^t} = 1$ and $\overline{\mathfrak{o}}^t_{n^t} = -1$ the interact function is the identity except for $i(g, p_{n^t}, p_{n^t+1})$ and $i(g, p_{n^t+1}, p_{n^t})$. Therefore,

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) \tag{6.231}$$

$$= \Bigg(p^t_1, ..., p^t_{n^t-1}, \Big(\underbrace{x^t_{n^t}}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Big),$$

$$\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{\max(-1,1)}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\Bigg). \tag{6.232}$$

The next is the evolution step. The evolve function e is the identity for all particles except if $\mathfrak{h} = 1$ this is only true for $\tilde{p}^t_{n^t}$ or if $\Delta\mathfrak{h} = 1$, only true for $\tilde{p}^t_{n^t+1}$, or if $\mathfrak{h} = -1$ only true for $\tilde{p}^t_{n^t+1}$, too. e is the identity for the global variable except if $\mathfrak{h} = 1$. Hence, we can reduce ϵ^{N} to

$$\epsilon^{\mathrm{N}}([g^t, \tilde{\mathbf{p}}^t]) = [{}_1e(g^t, p^t_1), (p^t_1, ..., p^t_{n^t-1}) \circ_2 e(g, \tilde{p}^t_{n^t})$$
$$\circ_2 e(g, \tilde{p}^t_{n^t+1}) \circ (p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|})] \tag{6.233}$$

$$= \Bigg[\Big(\mathfrak{q}^t, \underbrace{\max\left(\mathbf{start}, \overline{\mathfrak{q}}^t_{n^t}\right)}_{\Delta\mathfrak{q}}\Big),\ (p^t_1, ..., p^t_{n^t-1})$$

$$\circ \left(\Big(\underbrace{\overline{\mathfrak{z}}^t_{n^t}}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\right)$$

$$\circ \left(\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\right) \circ \left(p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\right)\Bigg]. \tag{6.234}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\Delta\mathfrak{q}}\big)\Big) := \Big(\underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.235}$$

Hence,

$$s\left([g^t, \mathbf{p}^t]\right) = \left[g^{t+1}, \mathbf{p}^{t+1}\right] \tag{6.236}$$

$$= \Bigg[\Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\Big),\ (p^t_1, ..., p^t_{n^t-1})$$

$$\circ \left(\Big(\underbrace{\overline{\mathfrak{z}}^t_{n^t}}_{\mathfrak{z}}, \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\right)$$

$$\circ \left(\Big(\underbrace{x^t_{n^t+1}}_{\mathfrak{z}}, \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\right) \circ \left(p^t_{n^t+2}, ..., p^t_{|\mathbf{p}^t|}\right)\Bigg]. \tag{6.237}$$

We need to prove again that the criteria (6.170) are true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$\begin{aligned}
&\psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\
&= \left(q^{t+1}, x_1^t \cdots x_{n^t-1}^t \bar{\mathfrak{z}}_{n^t}^t x_{n^t+1}^t \cdots x_{|\mathbf{p}^t|}^t _^{\omega}, n^t+1\right) \\
&= \left(q^{t+1}, \mathbf{x}^{t+1} _^{\omega}, n^{t+1}\right) \\
&= \underline{\alpha^{t+1}}
\end{aligned} \tag{6.238}$$

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}_{n^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1}^{t+1} = 1, \tag{6.239}$$

$$\mathfrak{h}_{n^{t+1}-\mathfrak{o}_{n^{t+1}}^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1-\mathfrak{o}_{n^t+1}^{t+1}}^{t+1} = \mathfrak{h}_{n^t+1-1}^{t+1} = -1, \tag{6.240}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \left\{\underbrace{n^{t+1}}_{=n^t+1}, \underbrace{n^{t+1} - \mathfrak{o}_{n^{t+1}}^{t+1}}_{=n^t}\right\} : \ \mathfrak{h}_j^{t+1} = 0, \tag{6.241}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}_j^{t+1} = 0 \ \wedge \Delta\mathfrak{o}_j^{t+1} = -1 \wedge \mathfrak{a}_j^{t+1} = 0, \tag{6.242}$$

$$\Delta q^{t+1} = \textbf{start}. \tag{6.243}$$

Fifth, we prove it for the case $\mathfrak{h}_{n^t}^t = 1$, $\mathfrak{h}_{n^t+1}^t = -1$, and $\delta(\mathfrak{q}^t, \mathfrak{z}_{n^t}^t) =: (\overline{\mathfrak{q}}^t, \bar{\mathfrak{z}}_{n^t}^t, \overline{\mathfrak{d}}_{n^t}^t) = (q^{t+1}, \bar{\mathfrak{z}}_{n^t}^t, -1)$ where $n^t \in \{2, ..., |\mathbf{p}^t| - 1\}$. This results in the same state $[g^t, \mathbf{p}^t]$ as in (6.230).

The particle p_{n^t} is the only particle with $\mathfrak{h}_{n^t}^t = 1$ and p_{n^t+1} with $\mathfrak{h}_{n^t+1}^t = -1$. Since $\overline{\mathfrak{d}}_{n^t}^t = -1$ and $\overline{\mathfrak{o}}_{n^t}^t = -1$ the interact function is the identity except for $i(g, p_{n^t-1}, p_{n^t})$ and $i(g, p_{n^t}, p_{n^t-1})$. Therefore,

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) \tag{6.244}$$

$$\begin{aligned}
&= \Bigg(p_1^t, ..., p_{n^t-2}^t, \Bigg(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{1}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{\max(-1,-1)}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Bigg), \\
&\qquad \Bigg(\underbrace{x_{n^t}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{1}_{\mathfrak{a}}\Bigg), p_{n^t+1}^t, ..., p_{|\mathbf{p}^t|}^t\Bigg)
\end{aligned} \tag{6.245}$$

The next is the evolution step. The evolve function e is the identity for all particles except if $\mathfrak{h} = 1$ this is only true for $\tilde{p}_{n^t}^t$ or if $\Delta\mathfrak{h} = 1$, only true for $\tilde{p}_{n^t-1}^t$, or if $\mathfrak{h} = -1$ only true for $\tilde{p}_{n^t+1}^t$. e is the identity for the global variable except if $\mathfrak{h} = 1$. Hence, we can

reduce ϵ^{N} to

$$\begin{aligned}\epsilon^{\mathrm{N}}([g^t, \tilde{\mathbf{p}}^t]) &= [_1 e(g^t, p_1^t), (p_1^t, ..., p_{n^t-2}^t) \circ_2 e(g, p_{n^t-1}^t) \\ &\qquad \circ_2 e(g, p_{n^t}^t) \circ_2 e(g, p_{n^t+1}^t) \circ (p_{n^t+2}^t, ..., p_{|\mathbf{p}^t|}^t)] \end{aligned} \tag{6.246}$$

$$\begin{aligned} &= \Bigg[\Big(\mathfrak{q}^t, \underbrace{\max\left(\mathbf{start}, \bar{\mathfrak{q}}_{n^t}^t\right)}_{\Delta\mathfrak{q}} \Big), \ (p_1^t, ..., p_{n^t-2}^t) \\ &\quad \circ \Bigg(\Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \\ &\quad \circ \Bigg(\Big(\underbrace{\bar{\mathfrak{z}}_{n^t}^t}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \\ &\quad \circ \Bigg(\Big(\underbrace{x_{n^t+1}^t}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \circ \Big(p_{n^t+2}^t, ..., p_{|\mathbf{p}^t|}^t \Big) \Bigg]. \end{aligned} \tag{6.247}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\bar{\mathfrak{q}}_{n^t}^t}_{\Delta\mathfrak{q}} \big) \Big) := \Big(\underbrace{\bar{\mathfrak{q}}_{n^t}^t}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}} \Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}} \Big) \tag{6.248}$$

Hence,

$$s\left([g^t, \mathbf{p}^t]\right) = [g^{t+1}, \mathbf{p}^{t+1}] \tag{6.249}$$

$$\begin{aligned} &= \Bigg[\Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}} \Big), \ (p_1^t, ..., p_{n^t-2}^t) \\ &\quad \circ \Bigg(\Big(\underbrace{x_{n^t-1}^t}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \\ &\quad \circ \Bigg(\Big(\underbrace{\bar{\mathfrak{z}}_{n^t}^t}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \\ &\quad \circ \Bigg(\Big(\underbrace{x_{n^t+1}^t}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}} \Big) \Bigg) \circ \Big(p_{n^t+2}^t, ..., p_{|\mathbf{p}^t|}^t \Big) \Bigg] \end{aligned} \tag{6.250}$$

We need to prove again that the criteria (6.170) are true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$\begin{aligned} &\psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\ &= \Big(q^{t+1}, x_1^t \cdots x_{n^t-1}^t \bar{\mathfrak{z}}_{n^t}^t x_{n^t+1}^t \cdots x_{|\mathbf{p}^t|}^t \_^{\omega}, n^t - 1 \Big) \\ &= \left(q^{t+1}, \mathbf{x}^{t+1} \_^{\omega}, n^{t+1} \right) \\ &= \underline{\alpha^{t+1}} \end{aligned} \tag{6.251}$$

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}^{t+1}_{n^{t+1}} = \mathfrak{h}^{t+1}_{n^t-1} = 1, \tag{6.252}$$

$$\mathfrak{h}^{t+1}_{n^{t+1}-\mathfrak{o}^{t+1}_{n^{t+1}}} = \mathfrak{h}^{t+1}_{n^t-1-\mathfrak{o}^{t+1}_{n^t-1}} = \mathfrak{h}^{t+1}_{n^t-1-(-1)} = -1, \tag{6.253}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \left\{\underbrace{n^{t+1}}_{=n^t-1}, \underbrace{n^{t+1} - \mathfrak{o}^{t+1}_{n^{t+1}}}_{=n^t}\right\} : \ \mathfrak{h}^{t+1}_j = 0, \tag{6.254}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}^{t+1}_j = 0 \ \wedge \Delta\mathfrak{o}^{t+1}_j = -1 \wedge \mathfrak{a}^{t+1}_j = 0, \tag{6.255}$$

$$\Delta q^{t+1} = \mathbf{start}. \tag{6.256}$$

Sixth and last, we prove it for the case $n^t = |\mathbf{p}^t|$ with $\mathfrak{h}^t_{|\mathbf{p}^t|} = 1$ and $\delta(\mathfrak{q}^t, \mathfrak{z}^t_{|\mathbf{p}^t|}) =: (\overline{\mathfrak{q}}^t, \overline{\mathfrak{z}}^t_{|\mathbf{p}^t|}, \overline{\mathfrak{d}}^t_{|\mathbf{p}^t|}) = (q^{t+1}, \overline{\mathfrak{z}}^t_{|\mathbf{p}^t|}, 1)$. From this follows that $\mathfrak{h}^t_{|\mathbf{p}^t|-1} = -1$ and $\mathfrak{o}^t_{|\mathbf{p}^t|} = 1$.

Using the criteria (6.170) lead to

$$\left[g^t, \mathbf{p}^t\right] = \left[\left(\underbrace{q^t}_{\mathfrak{q}}, \underbrace{\mathbf{start}}_{\Delta\mathfrak{q}}\right), \begin{pmatrix} (\underbrace{x^t_1}_{\mathfrak{z}}, & \underbrace{0}_{\mathfrak{h}}, & \underbrace{0}_{\Delta\mathfrak{h}}, & \underbrace{-1}_{\mathfrak{o}}, & \underbrace{-1}_{\Delta\mathfrak{o}}, & \underbrace{0}_{\mathfrak{a}}) \\ \vdots & & & & & \\ (x^t_{|\mathbf{p}^t|-2}, & 0, & 0, & -1, & -1, & 0) \\ (x^t_{|\mathbf{p}^t|-1}, & -1, & 0, & -1, & -1, & 0) \\ (x^t_{|\mathbf{p}^t|} & 1, & 0, & 1, & -1, & 0) \end{pmatrix}^{\mathbf{T}}\right] \tag{6.257}$$

The particle $p_{|\mathbf{p}^t|}$ is the only particle with $\mathfrak{h}^t_{|\mathbf{p}^t|} = 1$ and $p_{|\mathbf{p}^t|-1}$ with $\mathfrak{h}^t_{|\mathbf{p}^t|-1} = -1$. Since $\overline{\mathfrak{d}}^t_{n^t} = 1$ and $\overline{\mathfrak{o}}^t_{n^t} = 1$ the interact function is the identity for all particle. Therefore,

$$\tilde{\mathbf{p}}^t := \iota^{\mathrm{N}\times\mathrm{U}}([g^t, \mathbf{p}^t]) \tag{6.258}$$

$$= \mathbf{p}^t. \tag{6.259}$$

The next is the evolution step. The evolve function e is the identity for all particles except if $\mathfrak{h} = 1$ this is only true for $\tilde{p}^t_{|\mathbf{p}^t|}$ or if $\Delta\mathfrak{h} = 1$, true for no particle, or if $\mathfrak{h} = -1$ only true for $\tilde{p}^t_{|\mathbf{p}^t|-1}$. e is the identity for the global variable except if $\mathfrak{h} = 1$. The particle $p_{|\mathbf{p}^t|}$ did not interact, so $\mathfrak{a}_{|\mathbf{p}^t|} = 0$. The evolve function has a special case for this, where a new particle is generated. We can write ϵ^{N} as

$$\epsilon^{\mathrm{N}}([g^t, \tilde{\mathbf{p}}^t]) = \left[{}_1e(g^t, p^t_1), (p^t_1, ..., p^t_{|\mathbf{p}^t|-2}) \circ_2 e(g, p^t_{|\mathbf{p}^t|-1}) \circ_2 e(g, p^t_{|\mathbf{p}^t|})\right] \tag{6.260}$$

$$\begin{aligned} = \Bigg[& \Big(\mathfrak{q}^t, \underbrace{\max\left(\mathbf{start}, \overline{\mathfrak{q}}^t_{|\mathbf{p}^t|}\right)}_{\Delta\mathfrak{q}}\Big), \ (p^t_1, ..., p^t_{|\mathbf{p}^t|-2}) \\ & \circ \Big(\Big(\underbrace{x^t_{|\mathbf{p}^t|-1}}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \\ & \circ \Big(\Big(\underbrace{\overline{\mathfrak{z}}^t_{|\mathbf{p}^t|}}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), \Big(\underbrace{-}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big)\Bigg]. \end{aligned} \tag{6.261}$$

The last step for this case is the evolve function of the global variable.

$$\overset{\circ}{e}\Big(\big(\mathfrak{q}^t, \underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\Delta\mathfrak{q}}\big)\Big) := \Big(\underbrace{\overline{\mathfrak{q}}^t_{n^t}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big) = \Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big) \tag{6.262}$$

Hence,

$$s\left(\left[g^t, \mathbf{p}^t\right]\right) = \left[g^{t+1}, \mathbf{p}^{t+1}\right] \tag{6.263}$$

$$\begin{aligned} &= \Bigg[\Big(\underbrace{q^{t+1}}_{\mathfrak{q}}, \underbrace{\textbf{start}}_{\Delta\mathfrak{q}}\Big),\ (p^t_1, ..., p^t_{|\mathbf{p}^t|-2}) \\ &\quad \circ \Big(\Big(\underbrace{x^t_{|\mathbf{p}^t|-1}}_{\mathfrak{z}} \underbrace{0}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big) \\ &\quad \circ \Big(\Big(\underbrace{\overline{\mathfrak{z}}^t_{|\mathbf{p}^t|}}_{\mathfrak{z}} \underbrace{-1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{-1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big), \Big(\underbrace{\_}_{\mathfrak{z}} \underbrace{1}_{\mathfrak{h}}, \underbrace{0}_{\Delta\mathfrak{h}}, \underbrace{1}_{\mathfrak{o}}, \underbrace{-1}_{\Delta\mathfrak{o}}, \underbrace{0}_{\mathfrak{a}}\Big)\Big)\Bigg]. \end{aligned} \tag{6.264}$$

We need to prove again that the criteria (6.170) are true for $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$.

$$\begin{aligned} &\psi^{-1}\left(\left[g^{t+1}, \mathbf{p}^{t+1}\right]\right) \\ &= \Big(q^{t+1}, x^t_1 \cdots x^t_{|\mathbf{p}^t|-1}\overline{\mathfrak{z}}^t_{|\mathbf{p}^t|} - -^{\omega}, |\mathbf{p}^t| + 1\Big) \\ &= \left(q^{t+1}, \mathbf{x}^{t+1}\ \_^{\omega}, n^{t+1}\right) \\ &= \underline{\alpha^{t+1}} \end{aligned} \tag{6.265}$$

Note that $\_\_^{\omega} = \_^{\omega}$, because ω is the smallest ordinal number.

The rest of the criteria can be read from the calculated $\left[g^{t+1}, \mathbf{p}^{t+1}\right]$

$$\mathfrak{h}^{t+1}_{n^{t+1}} = \mathfrak{h}^{t+1}_{|\mathbf{p}^t|+1} = 1, \tag{6.266}$$

$$\mathfrak{h}^{t+1}_{n^{t+1}-\mathfrak{o}^{t+1}_{n^{t+1}}} = \mathfrak{h}^{t+1}_{|\mathbf{p}^t|+1-\mathfrak{o}^{t+1}_{|\mathbf{p}^t|+1}} = \mathfrak{h}^{t+1}_{|\mathbf{p}^t|+1-1} = -1, \tag{6.267}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} \setminus \Big\{\underbrace{n^{t+1}}_{=|\mathbf{p}^t|+1}, \underbrace{n^{t+1} - \mathfrak{o}^{t+1}_{n^{t+1}}}_{=|\mathbf{p}^t|}\Big\} :\ \mathfrak{h}^{t+1}_j = 0, \tag{6.268}$$

$$\forall j \in \left\{1, ..., |\mathbf{p}^{t+1}|\right\} : \Delta\mathfrak{h}^{t+1}_j = 0\ \wedge \Delta\mathfrak{o}^{t+1}_j = -1 \wedge \mathfrak{a}^{t+1}_j = 0, \tag{6.269}$$

$$\Delta q^{t+1} = \textbf{start}. \tag{6.270}$$

With this, we proved for all cases that the particle method can simulate each configuration transition of the Turing machine. The final part is to prove that the particle method stops if and only if the Turing machine halts.

$$f(g^t) = \top\ \leftrightarrow\ q^t \in \{accept, reject\} \tag{6.271}$$

This is proven by

$$f\left(g^t\right) = \top \tag{6.272}$$

$$\leftrightarrow \mathfrak{q}^t \in \{accept,\ reject\} \tag{6.273}$$

$$\xleftrightarrow{\psi^{-1}\left(\left[g^t, \mathbf{p}^t\right]\right)=\alpha^t} q^t \in \{accept,\ reject\}. \tag{6.274}$$

□

6.5 Halting Decidability of Particle Methods

The halting problem is the question if algorithms halt for a given input. This question is generally undecidable for Turing machines [84]. Since particle methods are also Turing powerful (thms. 3, 4), the halting problem is, in general, also undecidable for particle methods even if greatly restricted. Here we provide an answer to the question: Under which conditions is the halting problem for particle methods decidable?

Theorem 5 (Halting decidability of particle methods under certain constraints)**.** *The halting problem is decidable for particle methods under the constraints:*

First, the number of particle and global variable properties are finite, and all properties are from finite sets. Hence,

$$|P| < \infty, \quad |G| < \infty. \tag{6.275}$$

Second,

$$e, i, u, f, \overset{\circ}{e} \text{ are computable functions.} \tag{6.276}$$

Third, the evolve function is not allowed to produce particle

$$|_2 e(g, p_j)| \leq 1. \tag{6.277}$$

Proof.
For all the following steps, the conditions that $|P|$ and $|G|$ are finite (6.275) are essential.

Since the interact function i is computable (6.276) and the initial particle tuple $\mathbf{p}^1$ is finite ((3.8), def. 1), the first interact sub-function ι^{I} (3.10) is computable.

The second interact sub-function $\iota^{\mathrm{I}\times\mathrm{U}}$ (3.11) is computable because the first interact sub-function is computable, the neighborhood function is computable (6.276) and the neighborhood size is finite ((3.3), def. 1).

The third interact sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$ (3.12) is computable because the second interact sub-function $\iota^{\mathrm{I}\times\mathrm{U}}$ is computable, and the initial particle tuple $\mathbf{p}^1$ is finite (3.8).

The first evolve sub-function ϵ^{I} (3.13) is computable because the evolve function is computable.

The second evolve sub-function ϵ^{N} (3.14) is computable because the first evolve sub-function ϵ^{I} is computable and the initial particle tuple $\mathbf{p}^1$ is finite (3.8).

The step function s (3.15) is computable because the third interact sub-function $\iota^{\mathrm{N}\times\mathrm{U}}$, the second evolve function ϵ^{N}, and the evolve function of the global variable $\overset{\circ}{e}$ are computable (6.276). To decide if a particle method state is a final state, the stop function f is needed. It is also computable (6.276), hence, decidable.

The initial particle tuple $\mathbf{p}^1$ is finite (3.8) with a length of $m^1 := |\mathbf{p}^1|$ and the evolve function e can not produce particles (6.277). It is also the only place where particles can be produced. Hence, the number of particles of the t-th particle methods step is always smaller or equal to the previous step ($m^{t+1} \leq m^t$). From this, we can define the set of all states that can be reached from an instance with $m^1 = |\mathbf{p}^1|$ by

$$[G \times P^{*<m}] \quad \text{for} \quad P^{*<m} := \bigcup_{j=0}^{m} P^j. \tag{6.278}$$

From $m^1 < \infty$ (3.8) and $|P|$ and $|G|$ are finite (6.275), follows that

$$M := \left|[G\times, P^{*<m}]\right| < \infty. \tag{6.279}$$

Hence, there are only a finite set of reachable states. Therefore, after a maximum of M state transition steps, we know that either the states are repeating or the particle method stops, i.e., it does not halt or halts. Conclusively, we can decide the halting problem for particle methods under the conditions (6.275) to (6.277). □

6.6 Conclusion

Our definition of particle methods is practically universal, as seen in chapter 4. Despite this broad applicability, the theoretical limits of our particle methods definition were unclear.

Here we used automata theory to get insights into the theoretical power of particle methods. We have proven Turing powerfulness under two mutually non-containing sets of constraints and the halting decidability for one set of constraints.

The result that particle methods are Turing powerful, even if restricted, shows us the considerable theoretical potential of particle methods. Even though it does not tell how useful a specific particle method implementation of an algorithm is, we know we can express any algorithm as a particle method. Showing that particle methods are Turing powerful for mutually non-containing sets of constraints indicates that there is not one set of maximal restrictions but multiple ways of restriction of particle methods that can not be further restricted. The halting decidability gives us an idea under which conditions particle methods are not anymore Turing powerful. We did not prove that the restrictions we chose were tight. Hence, there may be more restrictive ones for Turing powerfulness and less restrictive ones for halting decidability.

Finding more restrictive constraints under which particle methods are still Turing powerful and less restrictive ones where particle methods are still halting decidable are topics future work could address. For halting decidability, future work could also investigate more practically relevant constraints. For example, many particle methods use counters and fixed limits for these counters when to stop. This could be a starting point for more practically relevant constraints.

Overall, this investigation regarding Turing powerfulness and halting decidability opens up the discussion of the theoretical abilities and limits of our particle methods definition.

Chapter 7

Particle Methods as a Basis for Scientific Software Engineering

7.1 Introduction

We have shown that our definition of particle methods allows the formulation of a wide range of algorithms (chapter 4) and the classification of particle methods in terms of their parallelizability (chapter 5) and computational power (chapter 6). These are more theoretical aspects of using our definition of particle methods. Therefore, they provide little insight into whether our definition can be used to design and implement scientific computing software and run algorithms on real computers.

We address this question by using our definition to design and implement a scientific computing software prototype using the particle method algorithm and instance as an interface. This allows generic parts of a particle method to be hidden from the user, such as the search for the neighborhood of each particle and the execution of the state transition function. We implement fast neighbor search methods for arbitrary-dimensional meshes and free particles for one-sided and two-sided interactions, as well as the state transition in its most general and hence, sequential form. We demonstrate the use of our prototype exemplary for the SPH application (sec. 4.4), PSE application (sec. 4.3), and the n-dimensional perfectly elastic collision algorithm (sec. 4.2), but the prototype is not limited to these three examples. In addition, we validate the software, particularly the runtime complexity concerning the fast neighbor access methods and the correctness of the results.

We want to emphasize this prototype serves the purpose of illustrating the applicability of our particle methods definition in software. Therefore, we did not implement the parallelization schemes but focused on the structure of the software and kept it sequential. A parallel version can replace the state transition in the future if desired.

The source code is available at:
https://git.mpi-cbg.de/mosaic/prototype-particle-methods-defintion-as-an-interface

7.2 Design of the Prototype

Class Structure of the Prototype

The fundamental design of the prototype is illustrated in figure 7.1 as a unified modeling language (UML) class diagram.

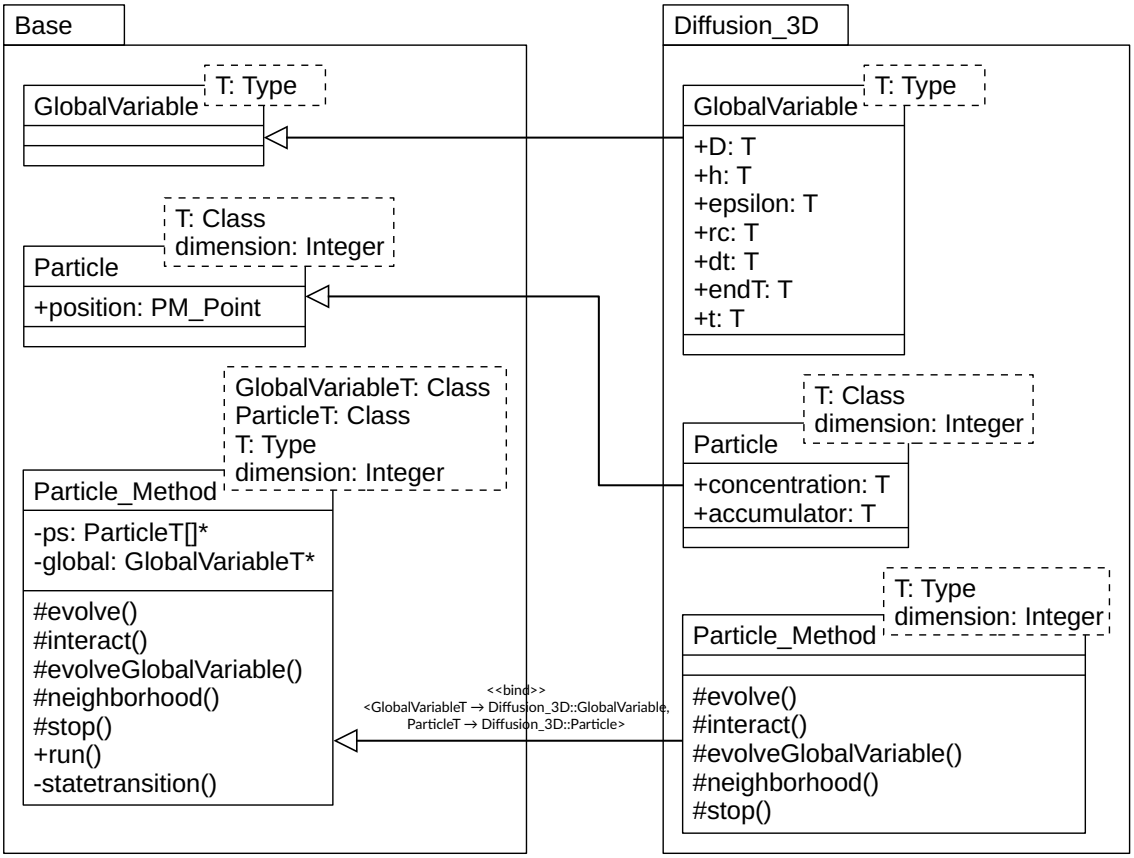


Figure 7.1: UML Class diagram of the base structure of the *prototype* and its application within the three-dimensional diffusion algorithm based on PSE.

We encapsulated the data structures and functions in the namespace `Base`. The two templated classes `GlobalVariable_Method` and `Particle` define the data structures and are mostly empty except for the `position` in `Particle`, which is necessary for the fast neighbor access algorithms. The templated class `Particle_Method` carries the bare bones of the five functions of the particle method algorithm and all hidden functionality. The hidden functionality is orchestrated by the `run()` method. It calls the appropriate fast neighbor access methods and chooses the correct state transition implementation. In addition, the function handles the neighbor search differently, whether ALL particles are neighbors or just those in a cutoff radius, whether there are free particles or mesh particles.

The uses of this prototype are then done in separate namespaces. We chose as an example the three-dimensional diffusion algorithm based on PSE (sec. 4.3). The user must create three new classes. These classes are subclasses from the three base classes. In the data

structure classes, the user adds the necessary properties. In the `Particle_Method` class, the user has to overwrite the functions `evolve()`, `interact()`, `evolveGlobalVariable()`, `neighborhood()`, and `stop()` precisely as described in the particle method algorithm except for `neighborhood()` due to the neighborhood access optimizations. Suppose the user decides not to overwrite certain functions. In that case, the prototype accounts for that and treats them respectively as identity functions, empty neighborhood, or stop after only one state transition step. The user needs to define the instance in addition to that separately and executes the particle method by calling the `run()` method on the instance.

Code examples

We show code examples to highlight our prototype's basic functionality and use. First, starting from the generic state transition function implementation, going over the user-defined particle method algorithm code to the user-defined instance of the particle method algorithm. For demonstration, we use the n-dimensional collision (sec. 4.2).

The generic state transition function is implemented into two functions the `run()` and the `statetransition()` function.

```
1   void run(){
2     neighborhood(global);
3     if(neighborhoodTyp==MESH) create_stencil_by_cutoff();
4     stop(global);
5     if (stopTest){
6       while(!stop(global)){
7         statetransition();
8       }
9     }
10    else {
11      statetransition();
12    }
13  }
```

Listing 7.1: State transition from the initial state to the finale state

The `run()` function coordinates the `statetransition()` function. In case the `stop()` function is not implemented/ overwritten by the user, the `statetransition()` function is called just once.

The function `statetransition()` calculates one state transition step and reads like this:

```
1   void statetransition(){
2     //interaction loop
3     if (neighborhoodTyp!=NONE) {
4       if (neighborhoodTyp==FREE) create_cell_list();
5       if (neighborhoodTyp==FREE || neighborhoodTyp==MESH){
6         for (index_I=0; index_I<ps.size(); index_I++){
7           fill_neighborhood(index_I);
8           for (int j=0; j<neighbors.size(); j++){
9             index_J=neighbors[j];
10            interact (global, ps[index_I],ps[index_J]);
```

```
11          }
12        }
13      }
14      else if (neighborhoodTyp==ALL){
15        for (index_I=0; index_I<ps.size(); index_I++){
16          for (index_J=0; index_J<ps.size(); index_J++){
17            interact (global, ps[index_I],ps[index_J]);
18          }
19        }
20      }
21    }
22    //evolution Loop
23    if (evolveTest){
24      for (index_I=0; index_I<ps.size(); index_I++){
25        evolve(global, ps[index_I]);
26      }
27    }
28    //evolution of the global variable
29    evolveGlobalVariable(global);
30  }
31  \texttt{statetransition()}
```

The `statetransition()` function handles the transition from one state to the next state and follows the state transition step definition (def. 17). However, this prototype supports some acceleration techies for the neighbor search. The function accounts for this and handles the neighbor search differently, whether ALL particles are neighbors or just the ones in a cutoff radius are neighbors, whether there are free particles or mesh particles.

The `run()` and the `statetranstion()` function are neither seen nor touched by the user. Instead, the user must define only a particle method and the instance.

The particle method algorithm is implemented, for instance, like this:

```
1   template <typename T, int dimension>
2   class Particle_Method : public Base::Particle_Method<
     GlobalVariable<T>, Particle<T,dimension>, T, dimension>{
3    public:
4    using Base::Particle_Method<GlobalVariable<T>, Particle<T,
     dimension>, T, dimension>::Particle_Method;
5
6    protected:
7    bool stop(const GlobalVariable<T>& g)override{
8      return (g.endT<g.t);
9    }
10   void neighborhood(const GlobalVariable<T>& g) override{
11     this->cutoffRadius=g.rc;
12     this->symmetricInteract=true;
13     this->neighborhoodTyp=Base::FREE;
14   }
15   void interact(const GlobalVariable<T>& g, Particle<T,dimension
     >& p, Particle<T,dimension>& q)override{
16     PM_Point<T,dimension> diff =q.position-p.position;
17     diff =diff/diff.abs2()*(diff*(q.velocity-p.velocity));
18
```

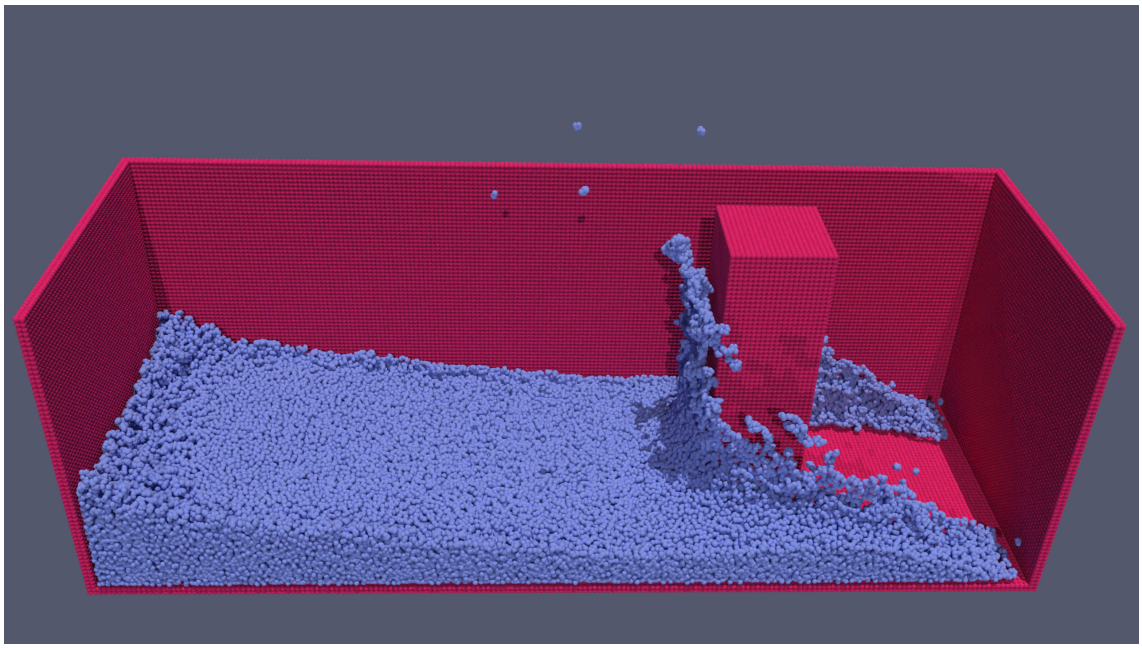
```
19        p.velocity +=diff;
20        q.velocity -=diff;
21      }
22      void evolve(GlobalVariable<T>& g,Particle<T,dimension>& p)
      override{
23        p.position +=g.dt*p.velocity;
24      }
25      void evolveGlobalVariable(GlobalVariable<T>& g)override{
26        g.t +=g.dt;
27      }
28    };
```

The `Particle_Method` class is the template for the functions of the particle method algorithm. The global variable and particle structure definition happen in separate classes. These two classes contain only the respective properties except the position. The position is predefined. In the here presented example, the `Particle_Method` class contains the implementation of all functions for the n-dimensional collision. Except for the neighborhood function, the code resembles exactly the formulation from the section 4.2. For example, the evolve function e adds in the formulation the motion to the position of each particle ($\underline{x}_j + \Delta t \underline{v}_j$) in the implementation, it is the same and reads like `p.position +=g.dt*p.velocity;`, which is essentially identical. In the code, it is not necessary to mention the not changed properties. The neighborhood function differs due to the neighborhood access optimizations.

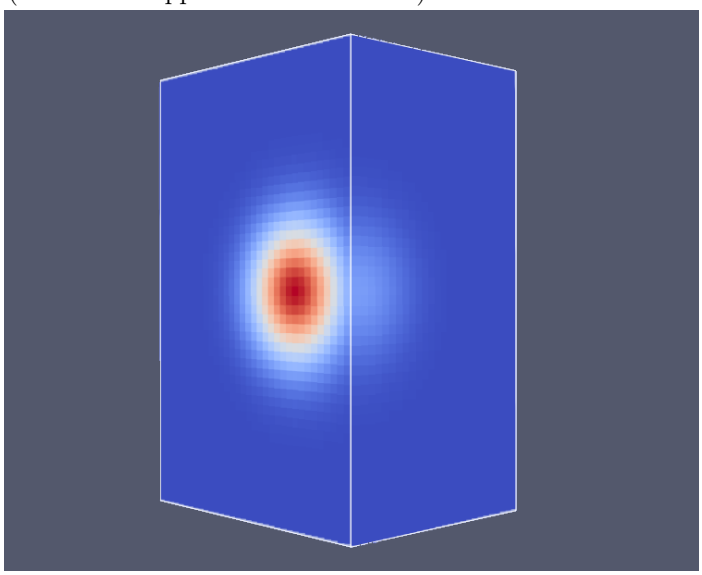
7.3 Applications, Comparisons, Convergence Study, and Run-time Evaluations

We demonstrate the use of our prototype exemplarily for SPH (fig. 7.2a), PSE (fig. 7.2b), and the n-dimensional perfectly elastic collision (fig. 7.2c) algorithms. We have also tested the software to ensure it works as expected. Therefore we compared our prototype to native C++ implementations, did a convergence study, and conducted two run-time evaluations.

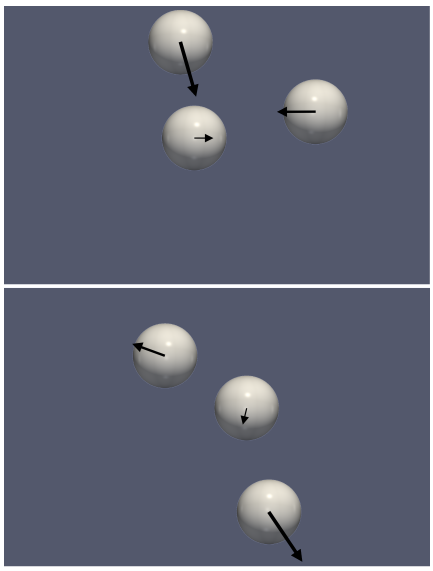
The demonstration applications are exactly the formulations from chapter 4, i.e., SPH (sec. 4.4), PSE (sec. 4.3), and n-dimensional perfectly elastic collision (sec. 4.2). They cover both continuous (SPH, PSE) and discrete (elastic collision) models, with free particles (SPH, elastic collision) and mesh-based (PSE). Hence, they demonstrate the variety of particle methods and this prototype.



(a) Fluid dynamics simulation using SPH to solve the standard "dam break" test case. Individual simulation particles are visualized as blue balls, while purple balls make up the walls of the container (front wall clipped for visualization).



(b) Diffusion simulation of a Gaussian pulse using PSE in 3D (domain clipped to half for better visualization).



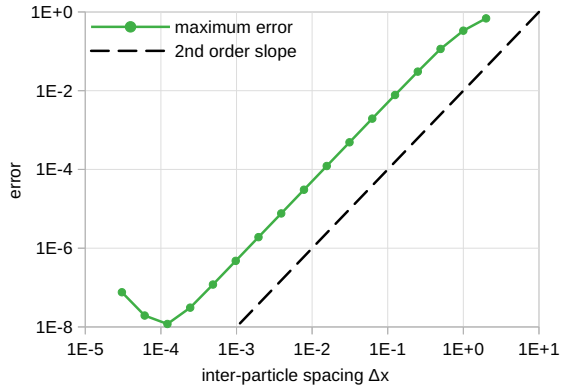
(c) Perfectly elastic collision simulation of three hard spheres with velocity vectors shown by black arrows.

Figure 7.2: Experiments with our prototype software (visualized using ParaView [85])

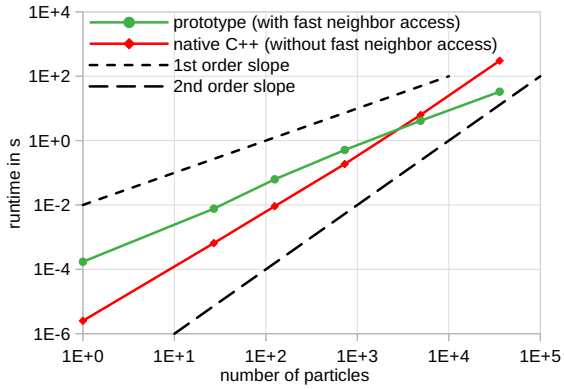
For the comparison to native C++, we implemented the perfectly elastic collision (sec. 4.2) and the 3D diffusion application in native C++. Then we calculated the difference to our particle methods implementation. The results are identical. Note that

we simulated an example with three balls that do not interact simultaneously (i.e., no three-way collisions). This is to ensure the result is independent of the indexing order of the particles, which is not preserved in a fast neighbor search algorithm. Since we use the native C++ implementation to validate the correctness of our implementation, we deliberately kept it as simple as possible to exclude implementation mistakes. Therefore, it does not use any fast neighbor search acceleration. The native C++ implementation without fast neighbor search acceleration also serves as a baseline for the time complexity test to show that our fast neighbor search implementation accelerates the computation.

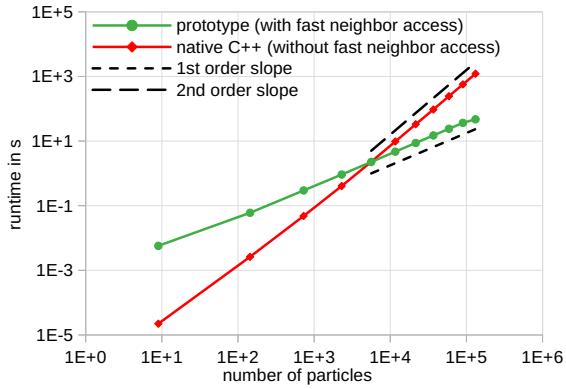
For the convergence study, we tested the PSE (sec. 4.3) approximation of the second derivative of a sine function in 1D using a second-order accurate kernel function. The analytical solution of the second derivative of $\sin(x)$ is $-\sin(x)$. We tested for different grid (or particle) spacings. Then, using the analytical solution, we plotted the error. The solution converges with decreasing inter-particle spacing with a second order to the analytical solution until the errors from finite-precision arithmetic start to dominate at around 10^{-8}, as expected [79].



(a) Convergence study of PSE with second-order kernel



(b) Run-time for PSE in our framework and native C++ for different numbers of particles.



(c) Run-time for perfectly elastic collision in our framework and native C++ for different numbers of particles.

Figure 7.3: Validation plots.

The run-time evaluations consider the PSE and the perfectly elastic collision implementations from our prototype and native C++. We see for the implementation in our prototype that we have a linear run-time complexity concerning the number of particles, while it is quadratic for the native C++ implementation without fast neighbor search, as expected.

All tests can be found in the source code.

7.4 Conclusion

We have shown that our definition of particle methods allows the formulation of a wide range of algorithms (chapter 4) and the classification of particle methods in terms of their parallelizability (chapter 5) and computational power (chapter 6). These are more theoretical aspects of using our definition of particle methods. Therefore, they provide little insight into whether our definition can be used to design and implement scientific computing software and run algorithms on real computers.

In this chapter, we address this gap by presenting the design and implementation of a scientific computing software prototype where the particle method algorithm and instance are used as an interface. We also showcased its applicability to three applications of the chapter 4, i.e., perfectly elastic collision (sec. 4.2), PSE (sec. 4.3), and SPH (sec. 4.4). Finally, based on these applications, we tested the prototype against native C++ implementations to compare the results and evaluate the convergence and runtime. With the result, the prototype behaves as expected.

Since the prototype servers only the purpose of evaluating the practical usability of our definition in software, we did not optimize the runtime as can be seen in the first data points in the figure 7.3b and 7.3c before the fast neighbor access made it at some point faster. If desired, the runtime can be improved by optimizing the state transition implementation, which impacts all applications. This is possible because the prototype takes advantage of the structure of our definition and hides more complex algorithms from the user in the background, e.g., the state transition function and fast neighbor search. The structure of our definition is based on the principle of the separation of concerns. This allows for implementing particle methods algorithms directly as they are formulated with pen and paper. Furthermore, the separation of concerns allows the reuse of the algorithms for different instances by only swapping the instance. Despite their interest, we also did not implement the parallelization schemes. However, since already advanced parallelization libraries exist, like OpenFPM [43], we decided to implement our definition of particle methods as an interface for one of them in future work.

Future work could also leverage the design of the presented prototype to better structure software frameworks for particle methods, such as the PPM Library [77], OpenFPM [43], POOMA [75], or FDPS [44]. This would render them more accessible and maintainable, as the formal definition provides a common vocabulary. Furthermore, our definition enables the classification and comparison of software frameworks concerning their expressiveness, coverage of the definition, or optimization toward specific classes of particle methods using the parallelizability results (chapter 5). The separation of concerns built into the prototype allows extending it to approaches for systems of particle methods, where multiple particle methods are used to solve or simulate a more extensive applica-

tion. Further work could use theoretical insights like the constraints for parallelizability to guide the implementation process in the form of a compiler for a domain-specific language, which could potentially compile down to a particle methods interface like the one of our prototype.

In summary, the presented prototype for implementing particle methods is a first step toward a sound and formal understanding of how generic frameworks can be designed and implemented for particle methods. It also provides practical software implementation guidance and enables comparative evaluation regarding performance and readability.

Chapter 8

Results, Discussion, Outlook, and Conclusion

8.1 Problem

Mathematical definitions provide a precise, unambiguous way to formulate concepts. Hence, they provide a common language, thus the basis for a well-founded scientific discussion. By providing a common language, mathematical definitions help bridge the gap between different scientific disciplines and allow for deeper insights into the defined subject based on mathematical theorems. Especially in computer science, mathematical definitions are extremely valuable. For example, they help derive the expected behavior of a computer program, such as the convergence rate of a numerical solver or an algorithm's time and space complexity. Together, mathematical definitions form the basis for informed scientific discussions in computer science and help design and implement advanced algorithms.

One of the classic and widely used classes of algorithms in computational science is particle methods. They are used in a wide range of fields such as plasma physics [40], computational fluid dynamics [17, 20], image processing [11, 1], computer graphics [36], and computational optimization [38, 63]. In addition to their versatility, particle methods can efficiently be parallelized on shared- and distributed-memory computers [48, 43, 77, 44, 75]. Furthermore, they simplify simulations in complex [78] and time-varying [5] geometries, as no computational mesh needs to be generated and maintained. Despite the structural similarities of all these algorithms and methods, there is no consensus on what constitutes particle methods and if the methods discussed before are all particle methods. Overall, particle methods are a widely used, parallelizable collection of loosely connected methods with a rich theory due to decades of advances but lack a unifying mathematical definition with precisely defined terminology.

8.2 Results

In this thesis, we addressed the lack of a unifying mathematical definition of particle methods by formulating a mathematical definition of the particle-based class of algorithms. Thereby, the presented definition unified the so far loosely connected notion under the term

particle methods and enabled joint formal investigations across methods.

To this end, we presented a general mathematical definition of particle methods based on the commonly used terms and their usual way of implementation (chapter 3). Hence, the proposed definition highlights the algorithmic commonalities across applications, enabling a sharp classification of particle methods.

Our definition unifies particle methods and allows the formulation of particle methods for non-canonical algorithms (chapter 4). We showcased that by formulating classic particle methods such as SPH, PSE, MD and DEM, and other algorithms not generally recognized as particle methods. As examples in this regard, we considered triangulation refinement, Conway's game of life, and Gaussian elimination.

Further, we analyzed the parallelizability of our definition of particle methods for shared- and distributed-memory parallel computers (chapter 5). We provided a shared- and a distributed-memory parallelization scheme independent of specific applications. We proved the correctness of the presented parallelization schemes by showing equivalence to the sequential particle methods definition under certain assumptions, which we also defined and listed. Finally, we derived upper bounds on the time complexity of the proposed schemes executed on both sequential and parallel computers, and we discussed the parallel scalability limits.

We also investigated the limits of the computational power of our particle methods definition (chapter 6). Therefore, we have proven Turing powerfulness under two mutually non-containing sets of constraints. For a more restrictive set of constraints, we proved the decidability of the halting problem for particle methods and, thus, the loss of Turing powerfulness.

In addition to the theoretical analysis, we demonstrated the practical applicability of the definition as an interface for scientific computing software (chapter 7). We designed, implemented, and tested a running prototype, where our definition serves as an interface. The prototype hides the generic parts of a particle method from the user, such as finding the neighborhood and running the state transition function. We showcased its use by implementing and executing examples from SPH, PSE, and DEM.

8.3 Discussion

We formulated the presented definition in the most general way to encompass everything that is termed "particle method". However, most practical instances do not exploit the full generality of the definition. We chose the presented formulation for its structural similarities to practical implementations and its separation of concerns. The most significant impact on the separation of concerns was the choice to split particle methods into three constituents, a particle method algorithm, a particle method instance, and a particle method state transition function.

The particle method algorithm contains all user-defined data structures and functions to describe an algorithm as a particle method independent of a specific problem. The particle methods instance defines the specific problem defined by the initial global variable and the initial particle tuple. The particle method state transition function is the "engine" of particle methods. It is not user-defined and, therefore, identical for all particle methods. It calculates from the particle method instance and algorithm the particle method's

final state. We designed the particle method state transition function to be sequential to incorporate as many algorithms as possible. This choice renders the state transition function most general.

To render the structure of the particle method algorithm to be as general as possible, we designed it to consist of two data structures and five functions. Not all of these data structures and functions are strictly necessary, but they all have a purpose.

We now proceed to take a closer look and discuss their purpose. The first data structure is a particle. The structure of the particle space follows directly from the observation that in any particle method, the particles are points in some space that carry/store some properties, like a color, a velocity, an acceleration, a Boolean flag, a position in some space, etc. Most particle methods treat the position separately from the other properties. We decided to integrate the position as one possible property because not all particle methods have a position. If they have a position, it can be from various spaces with method-dependent treatment. By integrating it as one possible property, our definition can capture a wider variety of methods.

The second data structure is the global variable. We chose to introduce a global variable accessible throughout the particle method, despite being unnecessary from the perspective of the computational power. But it simplifies the formulation of many particle method algorithms by encapsulating simulation properties that are not specific to a particle. Hence, the global variable complicates the definition but improves readability and implementation efficiency.

The first function is the interact function. It is one of the two crucial functions. It is in some form part of most canonical particle method algorithms. It specifies how two particles interact, so it is restricted to pairwise interactions. Higher-order interactions, e.g., three-body interactions, can be realized by multiple pairwise interactions. The interact function can change both particles. This provides a potential performance advantage in cases of symmetric interactions. In symmetric interactions, the absolute value of the change of both particles is identical.

The second function is the neighborhood function. We introduced this function to reduce computation. Suppose the number of contributing interactions is low compared to all particles. In that case, the neighborhood function helps reduce the computation by returning the indices of only those interaction partners contributing to the result. However, a neighborhood does not need to be defined geometrically but can be an arbitrary set of particle indices to incorporate all kinds of neighborhoods, like adjacencies in graphs. We chose that the neighborhood function operates on particle indices instead of particles directly because an index is an element's unique identifier in a tuple. Besides, indices provide stable identifiers of particles throughout the whole interaction phase. They remain the same, while the particle itself may change.

The third function is the evolve function. It is the second crucial function. In most canonical particle method algorithms, it takes part as "motion". But in our definition, it has a much broader purpose. It changes the properties of a particle due to its own properties and the global variable. Since it is called after all interactions are done, it provides a place to update properties that need to stay constant during all interactions or to reset properties that serve as temporary accumulators during the interactions. It is also the place to implement autonomous dynamics, i.e., dynamics that do not depend on other particles' properties. The evolve function can also change the global variable, for instance,

to implement global reduction operations like summing a property over all particles. The result of the evolve function consists, besides the global variable, of a tuple of particles. Hence, it is the only place where particles can be created or destroyed. This behavior is needed by many methods, e.g., in population dynamics simulations or adaptive-resolution methods for continuous models [72].

The fourth function is the evolve function of the global variable. It changes the global variable based only on the current value of the global variable. We chose to have this function to make changes to the global variable simple. The only other place to change the global variable is the evolve function of the particles. Without the evolve function of the global variable, it would be challenging to prevent multiple evolutions of the global variable.

The fifth and last function is the stopping function. It is designed to only depend on the global variable. However, this is not limiting since every particle can change the global variable in the evolve method. Hence, each particle can influence the outcome of the stopping condition.

Together, the chosen structure of particle methods provides separation of concerns while keeping its structure similar to classic particle methods implementations.

These similarities are shown by the easy applicability of canonical applications like DEM (sec. 4.2), SPH (sec. 4.4), MD (sec. 4.5), and PSE (sec. 4.3). Also, Conway's game of life (sec. 4.7) fits perfectly into our definition. But, not all algorithms naturally fit into the definition. The formulation of the triangulation refinement (sec. 4.6) as a particle method was challenging, but by choosing to identify a triangle as a particle instead of vertices and edges, it fitted well into the structure. Despite these applications, a particle method could potentially have a worse time- and space-complexity than a non-particle algorithm, especially for non-canonical problems but also for algorithms where only a part of the particles are active, like for the treatment of boundary conditions. Also, Gaussian eliminations (sec. 4.8) tends in this direction. It turned out to be challenging to translate it into our definition and is less efficient since most particles are inactive throughout the interaction and evolution phase even though the time complexity is the same ($\mathcal{O}(n^3)$ for n being the number of unknowns) since the difference is only in the prefactor.

Further, our definition is limited by its monolithic nature. An algorithm composed of smaller algorithms could become very large and complex with several nested cases when explicitly formulated in our definition. This tendency is already visible in the SPH example (sec. 4.4), but it could get worse. For instance, a vortex-method-based solver for the two-dimensional incompressible Navier-Stokes equation [21] would get huge. If considering the three terms, material derivative, vertex stretching, and vorticity diffusion are not zero, then the algorithm would consist of the calculation of the curl of the vorticity, the calculation of velocity from the curl of the vorticity by a Poisson solver, diffusion of the vorticity, time integration of the vorticity, advection of the particles, and interpolating the vorticity back on to the grid. These steps are consecutively calculated and can be addressed by a variety of algorithms. Even though we did not mention the boundary treatment between these calculations, it is clearly not desirable to integrate such extensive algorithms into a single particle method despite it being possible.

Since our definition is not unique, an alternative definition or an extension of our

definition might handle algorithm compositions more elegantly. Also, more expressive or compact but equivalent definitions are possible. We chose the presented formulation for its structural similarities to practical implementations. Notwithstanding these limitations and alternatives, the presented definition established a rigorous algorithmic class that contrasts the so far loose and empirical notion of particle methods in practice. This rigorousness paved the way for further research both in the theoretical and algorithmic foundations of particle methods and the engineering of their software implementation. We explored three application areas of the presented definition: its parallelizability, computational power, and capacity in scientific software engineering.

The presented parallelization schemes (chapter 5) are not novel, and they are, in fact, similar to what is commonly implemented in software, but in contrast to most other parallelization schemes, they are formulated such that they are independent of specific applications. Therefore, our analysis is of immediate practical relevance for general-purpose particle methods frameworks, even for critical applications, as the proofs guarantee correctness.

A limitation of the proposed schemes is that they assume only pull interactions between the particles. This neglects the potential runtime benefits and versatility of symmetric interaction evaluations as we used them in the SPH applications (sec. 4.4) and implementation (chapter 7) to save almost half of the computation compared to a pull interaction formulation, because the absolute values of the changes in the interaction are identical for all properties. However, pull interactions are suitable for more computer architectures, especially in a shared-memory setting, which could allow for combining the distributed scheme with the shared-memory scheme. In the distributed-memory scheme, pull interactions also reduce communication since only particles in the center cell of a process are changed. They do not need to be communicated back, as would be the case for push or symmetric interactions, which also change copies of particles from other processes [77]. We also restricted the neighborhood function such that the cell-list strategy became applicable. This limits the expressiveness of the particle method but allows the efficient distribution of moving particles. Further, the constraint that particles are not allowed to leave the domain keeps domain handling simple. Otherwise, cells would dynamically need to be added or removed as necessary, resulting in a much more complex and dynamic mapping of cells to processes. We also restricted particles to not moving 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. Additionally, we restricted the global variable only to be changed by the evolve function of the global variable and not by the evolve function of any particle. Therefore, no global operations are allowed where global variable changes require additional synchronization. Still, global variable changes can be independently computed locally, keeping them in sync without communication. Finally, the time complexity of the checkerboard-like communication scheme does not have an optimal pre-factor, leaving many processes inactive. Nevertheless, it scales linearly with the number of processes and abstracts the internal scheduling of the network sub-system, providing at least a bound on the scalability and permitting

correctness proofs.

Despite these limitations, with their proof of equivalence to the sequential particle methods definition, the present parallelization schemes stand in contrast with the mainly algorithm-specific or empirically tested parallelization schemes used so far. Our formal analysis shows the way for future research into the theory of parallel scientific simulation algorithms and the engineering of provably correct parallel software implementations independent of specific applications.

The result that particle methods are Turing powerful, even if restricted, underscores the considerable theoretical potential of particle methods. Even though it does not tell how practical a specific particle method implementation of an algorithm is, we know that we can express any algorithm as a particle method. Proving that particle methods are Turing powerful for mutually non-containing sets of constraints shows that there are different ways of constraining particle methods without losing the Turing powerfulness. This indicates that the question after the most restricted particle method that is still Turing powerful might not have a unique answer.

The proof that the halting problem is decidable for particle methods if suitably restricted unveils the limits of the Turing powerfulness of our particle methods definition. We did not prove that the restrictions we chose were tight. Hence, there may be more restrictive ones for the Turing powerfulness and less restrictive ones for the halting decidability.

The software prototype showcases the potential benefits of a general definition of particle methods, even though it is a partially developed software. The software prototype interface is almost identical to the particle method algorithm structure. It has the two data structures, the five functions, and the separation of instance, algorithm, and state transition. Therefore, the separation of concerns from the definition structure is maintained on a high level. The most significant difference in the structure, despite that the user has to program in C++ instead of writing formulas is that the neighborhood function contains the general setting for the prototype since these are related to the neighbor search. Further, the prototype shows how one can take advantage of the structure of the definition. It hides the more complex generic algorithms from the user, e.g., the state transition function and fast neighbor search. The implementation of the prototype is not optimized with respect to performance. As a result, we see a clear overhead in the runtime comparison tests. Since performance was not the prototype's scope, we did not invest in its optimization and testing. If desired, the runtime can be improved by optimizing the state transition implementation, which would directly impact all applications. Also, we did not implement any parallelization scheme, although we investigated them (chapter 5). Regardless of these limitations, the prototype shows how our definition can serve as an interface for a scientific simulation software framework, where it is almost possible to translate a particle method formulation one to one into the software.

So far, we have discussed the theoretical results and the practical implementation, pretending there is a simple translation between the two. However, this assumption neglects that computers work with finite-precision arithmetic, while applications are often designed with integer, rational, or real numbers. Integer, rational, or real numbers can be infinite in size and precision. However, their closest relatives in finite-precision arithmetic, the integers, and floating point numbers, are finite in size and precision, leading to discrepancies in the laws they follow. For instance, the addition and multiplication for integer, rational, or real numbers are commutative, associative, and distributive, but we lose the associative and distributive law in finite-precision arithmetic. This could result in a potential loss of order independence of the interact function of our definition. Order independence of the interact function is required in the parallelization scheme for distributed-memory systems (sec. 5.3), which is, in that case, formally not applicable anymore. But it gets theoretically worse, while integer and rational numbers are still commutable, this is not true for all real numbers, for instance, the Chaitin omega number [14]. However, in practice, such numbers appear almost never.

For our definition of particle methods, this means that the applications have, in theory, different results than in practice. This is very prominent in chaotic systems, where small perturbations can result in a vastly different outcome, like the passive double pendulum [54] or perfectly elastic collision of many balls (sec. 4.2). Using finite-precision arithmetic effectively rounds values. Hence, it introduces small perturbations. These small perturbations not only influence chaotic systems but also cause not desired behavior of a more stable system. For instance, the finite-precision arithmetic induced error can grow arbitrarily big in PSE. We can see this in the convergence study of PSE (fig. 7.3a), where the error gets bigger for smaller particle spacing. This error grows with second order and comes from dividing by the square of the particle spacing related parameter ϵ. This scales the almost machine precision small error to approximately $\frac{\text{machine precision}}{\epsilon^2}$. Many of these finite-precision arithmetic induced errors are investigated [61, 27], and some can be addressed [6, 47]. But the impact of finite precision arithmetic on the particle methods theory, and whether there can be investigations across particle methods in this regard, needs to be addressed in future work.

8.4 Outlook

Further, future work could develop a less monolithic definition that would allow modular combinations of different particle methods. While this could lead to a formulation that can potentially be exploited directly in software engineering or the design of domain-specific programming languages for particle methods [48, 50], one would first need to solve some theoretical problems: How can different types of particles from different methods interact, e.g., during interpolating stored values from one set of particles to another? How can access be restricted to a particle subset, e.g., for boundary conditions? Solving these problems might lead to additional data structures or functions in the presented definition. These new formulations must be supported by applications leveraging the new data structures and functions. But also, for the presented definition of particle methods, the applications could be expanded, especially in the direction of non-canonical fields, to clarify further the practical abilities and limitations of the presented definition. Also, multiple different

formulations of the same classical particle method in our definition can help to identify advantageous properties of specific formulation strategies.

Future theoretical work could optimize the presented schemes for computer architectures with parallel or synchronously clocked communication and computation. Global operations could also be allowed, and the checkerboard-like communication pattern of the shared-memory scheme could be relaxed, leading to an improved scalability prefactor (up to 26-fold in three dimensions). Also, the network topology of the machine's interconnect could be explicitly incorporated into the parallelization scheme. Furthermore, proofs for push, symmetric, and no interaction schemes could be beneficial for specific use cases, as well as combining parallelization schemes for shared and distributed memory to match the heterogeneous architecture of modern supercomputers better.

For the computational power, future work could find more restrictive constraints under which particle methods are still Turing powerful and less restrictive ones, where particle methods are still halting decidable. For halting decidability, future work could also investigate more practically relevant constraints. For example, many particle methods use counters and fixed stopping limits for these counters. This could be a starting point for more practically relevant constraints.

On the software engineering side, future work could leverage the presented parallelization schemes and proofs to design a new generation of theoretically founded software frameworks. As a result, they would potentially be more predictable, suitable for security- and safety-critical applications, and more maintainable and understandable as they would be based on a common formal framework [76]. Future work can also leverage the presented prototype's interface to better structure software frameworks for particle methods, such as the PPM Library [77], OpenFPM [43], POOMA [75], or FDPS [44]. This would render them more accessible and maintainable, as the formal definition provides a common vocabulary. Furthermore, the presented definition (chapter 3) enables the classification and comparison of software frameworks concerning their expressiveness, coverage of the definition, or optimization towards specific classes of particle methods using the parallelizability results (chapter 5).

8.5 Conclusion

Formal definitions reveal the concepts upon which a method is founded, and they render it possible to rationalize the fundamental characteristics of a method. The presented definition of particle methods is the first necessary step toward a sound and formal understanding of what particle methods are, what they can do, and how efficient and powerful they can be. The present applications show the unifying nature of our formal particle methods definition and its applicability not only to classical particle methods. Moreover, these examples broaden the view toward applications for particle methods far beyond the scope of classic particle methods. Hence, our definition provides a new way of comparing and relating algorithms from different fields in a common language. On the other hand, the investigation regarding Turing powerfulness and halting decidability opens up the discussion of the theoretical abilities and limits of our particle methods definition. This discussion leads to entire classes of particle methods in terms of their computational power and provides a means to determine the expressiveness of certain constraints. Another angle

to classify particle methods is regarding their parallelizability. We presented and proved parallelization schemes applicable to whole classes of particle methods. These classes are determined by the restrictions a particle method has to fulfill to be applicable to these schemes. These schemes parallelize particle methods on shared- and distributed-memory systems with full knowledge of their validity, performance, and assumptions. Furthermore, we proved that they compute the same result as the underlying sequential particle method. Therefore, using them in a general framework for particle methods is suitable, even for critical computations, since the proofs guarantee that the parallelizations do not change the results. The parallelization schemes thereby mark the starting point of a well-founded discussion about the parallelization of particle methods independent of specific applications. Lastly, we designed and developed a scientific simulation software prototype based on our particle methods definition. We hid generic elements like the state transition from the user and used the structure of the definition as an interface for the prototype. This interface approach renders the translation process from the pen-and-paper formulation of a particle method into software straightforward and fast. Further, the presented prototype for implementing particle methods is a first step toward a sound and formal understanding of how generic frameworks can be designed and implemented for particle methods. It also provides practical software implementation guidance and enables comparative evaluation on common grounds.

Ultimately, we formulated in this thesis a mathematical definition of the algorithmic class of particle methods. This definition unified, for the first time, the so far loosely connected notion. Thus, it marks the necessary starting point for a broad range of joint formal investigations and applications across fields.

Bibliography

[1] Y. Afshar and I. F. Sbalzarini. "A Parallel Distributed-Memory Particle Method Enables Acquisition-Rate Segmentation of Large Fluorescence Microscopy Images". In: *PLoS One* 11.4 (2016), e0152528. DOI: 10.1371/journal.pone.0152528.

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

[3] G. M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. 1967, pp. 483–485.

[4] C. Bays. "Gliders in Cellular Automata". In: *Encyclopedia of Complexity and Systems Science*. Ed. by R. A. Meyers. New York, NY: Springer New York, 2009, pp. 4240–4249. ISBN: 978-0-387-30440-3. DOI: 10.1007/978-0-387-30440-3_249. URL: https://doi.org/10.1007/978-0-387-30440-3_249.

[5] M. Bergdorf, I. F. Sbalzarini, and P. Koumoutsakos. "A Lagrangian particle method for reaction-diffusion systems on deforming surfaces". In: *J. Math. Biol.* 61 (2010), pp. 649–663. DOI: 10.1007/s00285-009-0315-2.

[6] E. K. Blum. "A Modification of the Runge-Kutta Fourth-Order Method". In: *Mathematics of Computation* 16.78 (1962). Publisher: American Mathematical Society, pp. 176–187. ISSN: 0025-5718. DOI: 10.2307/2003056. URL: https://www.jstor.org/stable/2003056 (visited on 01/07/2023).

[7] B. Boghosian et al. *A particle method for continuous Hegselmann-Krause opinion dynamics*. en. arXiv:2211.06265 [physics]. Nov. 2022. URL: http://arxiv.org/abs/2211.06265 (visited on 12/29/2022).

[8] G. C. Bourantas et al. "Using DC PSE operator discretization in Eulerian meshless collocation methods improves their robustness in complex geometries". In: *Computers & Fluids* 136 (2016), pp. 285–300.

[9] M. Bussmann et al. "Radiative Signatures of the Relativistic Kelvin-Helmholtz Instability". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: 10.1145/2503210.2504564. URL: https://doi.org/10.1145/2503210.2504564.

[10] D.-G. Caprace, G. Winckelmans, and P. Chatelain. "An immersed lifting and dragging line model for the vortex particle-mesh method". In: *Theoretical and Computational Fluid Dynamics* 34.1 (2020), pp. 21–48. URL: https://doi.org/10.1007/s00162-019-00510-1.

[11] J. Cardinale, G. Paul, and I. F. Sbalzarini. "Discrete region competition for unknown numbers of connected regions". In: *IEEE Trans. Image Process.* 21.8 (2012), pp. 3531–3545.

[12] M. Cardinot, J. Griffith, and C. O'Riordan. *Cyclic Dominance in the Spatial Coevolutionary Optional Prisoner's Dilemma Game*. en. arXiv:1702.04299 [physics]. Feb. 2017. URL: http://arxiv.org/abs/1702.04299 (visited on 12/29/2022).

[13] P. Carpentier, G. Cohen, and A. Dallagi. *Particle Methods For Stochastic Optimal Control Problems*. en. arXiv:0907.4663 [math]. July 2009. URL: http://arxiv.org/abs/0907.4663 (visited on 12/29/2022).

[14] G. J. Chaitin. "A Theory of Program Size Formally Identical to Information Theory". In: *J. ACM* 22.3 (1975), 329–340. ISSN: 0004-5411. DOI: 10.1145/321892.321894. URL: https://doi.org/10.1145/321892.321894.

[15] J. Chen et al. "A coupled DEM-SPH model for moisture migration in unsaturated granular material under oscillation". In: *International Journal of Mechanical Sciences* 169 (2020), p. 105313. ISSN: 0020-7403. DOI: https://doi.org/10.1016/j.ijmecsci.2019.105313. URL: https://www.sciencedirect.com/science/article/pii/S0020740319330917.

[16] N. Chomsky. "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813.

[17] A. J. Chorin. "Numerical study of slightly viscous flow". In: *JFM* 57.4 (1973), pp. 785–796.

[18] R. H. Cole and R. Weller. "Underwater Explosions". In: *Physics Today* 1.6 (Jan. 1948), p. 35. DOI: 10.1063/1.3066176.

[19] E. Corona et al. *Tensor Train accelerated solvers for nonsmooth rigid body dynamics*. en. arXiv:1808.02558 [math]. Aug. 2018. URL: http://arxiv.org/abs/1808.02558 (visited on 12/29/2022).

[20] G. H. Cottet and S. Mas-Gallic. "A Particle Method to Solve the Navier-Stokes System". In: *Numer. Math.* 57 (1990), pp. 805–827.

[21] G.-H. Cottet and P. Poncet. "Advances in direct numerical simulations of 3D wall-bounded flows by Vortex-in-Cell methods". In: *Journal of Computational Physics* 193.1 (2003), pp. 136–158. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2003.08.025. URL: https://www.sciencedirect.com/science/article/pii/S0021999103004248.

[22] G.-H. Cottet. "Two Dimensional Incompressible Fluid Flow". In: *Mathematical Topics in Fluid Mechanics* (2020), p. 32.

[23] G.-H. Cottet et al. "High order Semi-Lagrangian particle methods for transport equations: numerical analysis and implementation issues". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 48.4 (2014), pp. 1029–1060.

[24] A. J. C. Crespo, M. Gómez-Gesteira, and R. A. Dalrymple. "Boundary Conditions Generated by Dynamic Particles in SPH Methods". In: *Computers, Materials & Continua* 5.3 (2007), pp. 173–184. ISSN: 1546-2226. DOI: 10.3970/cmc.2007.005.173. URL: http://www.techscience.com/cmc/v5n3/23429.

[25] P. Degond and S. Mas-Gallic. "The Weighted Particle Method for Convection-Diffusion Equations. Part 1: The Case of an Isotropic Viscosity". In: *Math. Comput.* 53.188 (1989), pp. 485–507.

[26] P. Degond and S. Mas-Gallic. "The Weighted Particle Method for Convection-Diffusion Equations. Part 2: The Anisotropic Case". In: *mc* 53.188 (1989), pp. 509–525.

[27] L. Delves. "Round-off errors in variational calculations". In: *Journal of Computational Physics* 3.1 (1968), pp. 17–28. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(68)90002-8. URL: https://www.sciencedirect.com/science/article/pii/0021999168900028.

[28] S Despréaux, A. M. t. I. Symposium, and 2018. "GPaR: A Parallel Graph Rewriting Tool". In: *ieeexplore.ieee.org* (2019). DOI: 10.1109/SYNASC.2018.00021","publicationTitle":"2018. URL: https://ieeexplore.ieee.org/abstract/document/8750739/.

[29] S. Dumont. *On enhanced descend algorithms for solving frictional multi-contact problems : applications to the Discrete Element Method*. en. arXiv:1204.5392 [physics]. Apr. 2012. URL: http://arxiv.org/abs/1204.5392 (visited on 12/29/2022).

[30] J. D. Eldredge, A. Leonard, and T. Colonius. "A General Deterministic Treatment of Derivatives in Particle Methods". In: *J. Comput. Phys.* 180 (2002), pp. 686–709.

[31] E. H. Friend. "Sorting on Electronic Computer Systems". In: *J. ACM* 3.3 (1956), 134–168. ISSN: 0004-5411. DOI: 10.1145/320831.320833. URL: https://doi.org/10.1145/320831.320833.

[32] M. Gardner. "Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life"". In: *Scientific American* 223.4 (Oct. 1970), pp. 120–123. ISSN: 0036-8733. DOI: 10.1038/scientificamerican1070-120. URL: https://www.scientificamerican.com/article/mathematical-games-1970-10 (visited on 12/30/2022).

[33] R. Gassmoeller et al. "Flexible and scalable particle-in-cell methods for massively parallel computations". en. In: *Geochemistry, Geophysics, Geosystems* 19.9 (Sept. 2018). arXiv:1612.03369 [physics], pp. 3596–3604. ISSN: 1525-2027, 1525-2027. DOI: 10.1029/2018GC007508. URL: http://arxiv.org/abs/1612.03369 (visited on 12/29/2022).

[34] R. A. Gingold and J. J. Monaghan. "Smoothed particle hydrodynamics - Theory and application to non-spherical stars". In: *Royal Astronomical Society, Montly Notices* 181 (1977), pp. 375–378.

[35] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[36] M. Gross and H. Pfister. *Point-Based Graphics*. English. Elsevier, May 2011. ISBN: 9780080548821. URL: http://books.google.de/books?id=9LCQ86u9Sc4C&pg=PR1&dq=point+based+graphics&hl=&cd=1&source=gbs_api.

[37] J. Gustafson. "Reevaluating Amdahl's Law". In: *Commun. ACM* 31 (May 1988), pp. 532–533. DOI: 10.1145/42411.42415.

[38] N. Hansen and A. Ostermeier. "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation". In: *Proceedings of the 1996 IEEE Conference on Evolutionary Computation (ICEC '96)*. 1996, pp. 312–317.

[39] C. Hierholzer and C. Wiener. "Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren". In: *Mathematische Annalen* 6 (1873), pp. 30–32. URL: https://doi.org/10.1007/bf01442866.

[40] R. W. Hockney. "Computer Experiment of Anomalous Diffusion". In: *Phys. Fluids* 9.9 (1966), pp. 1826–1835.

[41] R. W. Hockney and J. W. Eastwood. *Computer Simulation using Particles*. Institute of Physics Publishing, 1988.

[42] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2: architecture, programming and algorithms*. CRC Press, 2019.

[43] P. Incardona et al. "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers". In: *Comput. Phys. Commun.* 241 (2019), pp. 155–177.

[44] M. Iwasawa et al. "Implementation and performance of FDPS: a framework for developing parallel particle simulation codes". In: *Publications of the Astronomical Society of Japan* 68.4 (2016), p. 54.

[45] A. Jarynowski and P. Nyczka. "Single parameter model of marriage divorce dynamics with countries classification". In: (2018). DOI: 10.48550/ARXIV.1805.02878. URL: https://arxiv.org/abs/1805.02878.

[46] W. Jin et al. "On the Fidelity of Computational Models for the Flow of Milled Loblolly Pine: A Benchmark Study on Continuum-Mechanics Models and Discrete-Particle Models". In: *Frontiers in Energy Research* 10 (2022). ISSN: 2296-598X. URL: https://www.frontiersin.org/articles/10.3389/fenrg.2022.855848 (visited on 12/29/2022).

[47] W. Kahan. "Pracniques: Further Remarks on Reducing Truncation Errors". In: *Commun. ACM* 8.1 (1965), p. 40. ISSN: 0001-0782. DOI: 10.1145/363707.363723. URL: https://doi.org/10.1145/363707.363723.

[48] S. Karol et al. "A Domain-Specific Language and Editor for Parallel Particle Methods". In: *ACM Trans. Math. Softw.* 44.3 (2018), p. 34.

[49] J. R. Karr et al. "A Whole-Cell Computational Model Predicts Phenotype from Genotype". In: *Cell* 150 (2012), pp. 389–401.

[50] N. Khouzami et al. "The OpenPME Problem Solving Environment for Numerical Simulations". In: *Proc. Intl. Conf. on Computational Science*. Vol. 12742. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2021, pp. 614–627.

[51] S. Koshizuka et al. "Chapter 1 - Introduction". In: *Moving Particle Semi-implicit Method*. Ed. by S. Koshizuka et al. Academic Press, 2018, pp. 1–23. ISBN: 978-0-12-812779-7. DOI: https://doi.org/10.1016/B978-0-12-812779-7.00001-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128127797000011.

[52] D. Kozen. *Automata and Computability*. Undergraduate Texts in Computer Science. Springer New York, 2012. ISBN: 9781461218449. URL: https://books.google.de/books?id=Vo3fBwAAQBAJ.

[53] J. E. Lennard-Jones. "Cohesion". In: *Proceedings of the Physical Society* 43.5 (1931), pp. 461–482. DOI: 10.1088/0959-5309/43/5/301. URL: https://doi.org/10.1088/0959-5309/43/5/301.

[54] R. B. Levien and S. M. Tan. "Double pendulum: An experiment in chaos". In: *American Journal of Physics* 61.11 (1993), pp. 1038–1044. DOI: 10.1119/1.17335. eprint: https://doi.org/10.1119/1.17335. URL: https://doi.org/10.1119/1.17335.

[55] P. Liu et al. *On the acceleration of spatially distributed agent-based computations: a patch dynamics scheme*. en. arXiv:1404.7199 [math]. Apr. 2014. URL: http://arxiv.org/abs/1404.7199 (visited on 12/29/2022).

[56] W. K. Liu, S. Jun, and Y. F. Zhang. "Reproducing Kernel Particle Methods". In: *Int. J. Numer. Meth. Fluids* 20 (1995), pp. 1081–1106.

[57] P. A. Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *IEEE Intelligent Transportation Systems Conference (ITSC)* (2018). URL: https://elib.dlr.de/124092/.

[58] L. B. Lucy. "A numerical approach to the testing of the fission hypothesis." In: *The Astronomical Journal* 82 (Dec. 1977). ADS Bibcode: 1977AJ.....82.1013L, pp. 1013–1024. ISSN: 0004-6256. DOI: 10.1086/112164. URL: https://ui.adsabs.harvard.edu/abs/1977AJ.....82.1013L (visited on 12/19/2022).

[59] F. B. Manning. "An Approach to Highly Integrated, Computer-Maintained Cellular Arrays". In: *IEEE Transactions on Computers* C-26.6 (1977), pp. 536–552. DOI: 10.1109/TC.1977.1674879.

[60] Z. Mao, Z. Li, and G. E. Karniadakis. "Nonlocal flocking dynamics: Learning the fractional order of PDEs from particle simulations". en. In: *Communications on Applied Mathematics and Computation* 1.4 (Dec. 2019). arXiv:1810.11596 [physics, stat], pp. 597–619. ISSN: 2096-6385, 2661-8893. DOI: 10.1007/s42967-019-00031-y. URL: http://arxiv.org/abs/1810.11596 (visited on 12/29/2022).

[61] G. Meurant and Z. Strakoš. "The Lanczos and conjugate gradient algorithms in finite precision arithmetic". In: *Acta Numerica* 15 (2006), 471–542. DOI: 10.1017/S096249290626001X.

[62] J. J. Monaghan. "Smoothed particle hydrodynamics". In: *Rep. Prog. Phys.* 68 (2005), pp. 1703–1759.

[63] C. L. Müller and I. F. Sbalzarini. "Gaussian Adaptation revisited — an entropic view on Covariance Matrix Adaptation". In: *Proc. EvoStar*. Vol. 6024. Lect. Notes Comput. Sci. Istanbul, Turkey: Springer, 2010, pp. 432–441.

[64] J. von Neumann. "The general and logical theory of automata". In: *von Neumann, J. and Collected Works* 288.5 (1963).

[65] J. von Neumann. "Theory of Self-Reproducing Automata". In: (1966). Ed. by A. W. Burks.

[66] J. von Neumann. "Zur Einführung der transfiniten Zahlen". In: *Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae, sectio scientiarum mathematicarum* 1 (1923), pp. 199–208.

[67] H. M. Nilsen, I. Larsen, and X. Raynaud. *Combining the Modified Discrete Element Method with the Virtual Element Method for Fracturing of Porous Media*. en. arXiv:1702.01552 [math]. Feb. 2017. URL: http://arxiv.org/abs/1702.01552 (visited on 12/29/2022).

[68] C. Peng et al. "A fully resolved SPH-DEM method for heterogeneous suspensions with arbitrary particle shape". In: *Powder Technology* 387 (2021), pp. 509–526. ISSN: 0032-5910. DOI: https://doi.org/10.1016/j.powtec.2021.04.044. URL: https://www.sciencedirect.com/science/article/pii/S0032591021003259.

[69] A. Petras et al. "A least-squares implicit RBF-FD closest point method and applications to PDEs on moving surfaces". en. In: *Journal of Computational Physics* 381 (Mar. 2019). arXiv:1901.01742 [math], pp. 146–161. ISSN: 00219991. DOI: 10.1016/j.jcp.2018.12.031. URL: http://arxiv.org/abs/1901.01742 (visited on 12/29/2022).

[70] K. Preston et al. "Basics of cellular logic with some applications in medical image processing". In: *Proceedings of the IEEE* 67.5 (1979), pp. 826–856. DOI: 10.1109/PROC.1979.11331.

[71] B Quentrec and C Brot. "New method for searching for neighbors in molecular dynamics computations". In: *Journal of Computational Physics* 13.3 (1973), pp. 430 –432. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(73)90046-6. URL: http://www.sciencedirect.com/science/article/pii/0021999173900466.

[72] S. Reboux, B. Schrader, and I. F. Sbalzarini. "A self-organizing Lagrangian particle method for adaptive-resolution advection–diffusion simulations". In: *J. Comput. Phys.* 231 (2012), pp. 3623–3646.

[73] P. Rendell. *Turing machine universality of the game of life*. Springer, 2016.

[74] P. Rendell. "Turing Universality of the Game of Life". In: *Collision-Based Computing*. Ed. by A. Adamatzky. Springer, 2002, pp. 513–539. DOI: 10.1007/978-1-4471-0129-1_18. URL: https://doi.org/10.1007/978-1-4471-0129-1_18.

[75] J. Reynders et al. "POOMA: a framework for scientific simulation on parallel architectures". In: *Proceedings. First International Workshop on High-Level Programming Models and Supportive Environments*. Ed. by A. Bode et al. IEEE Comput. Soc. Press, 1996, pp. 41–49. ISBN: 0 8186 7567 5.

[76] I. F. Sbalzarini. "Abstractions and middleware for petascale computing and beyond." In: *Intl. J. Distr. Systems & Technol.* 1(2) (2010), pp. 40–56.

[77] I. F. Sbalzarini et al. "PPM – A Highly Efficient Parallel Particle-Mesh Library for the Simulation of Continuum Systems". In: *J. Comput. Phys.* 215.2 (2006), pp. 566–588.

[78] I. F. Sbalzarini et al. "Effects of organelle shape on fluorescence recovery after photobleaching". In: *BJ* 89.3 (2005), pp. 1482–1492.

[79] B. Schrader, S. Reboux, and I. F. Sbalzarini. "Discretization Correction of General Integral PSE Operators in Particle Methods". In: *J. Comput. Phys.* 229 (2010), pp. 4159–4182.

[80] Q. Shi and M. Sakai. "Recent progress on the discrete element method simulations for powder transport systems: A review". In: *Advanced Powder Technology* 33.8 (2022), p. 103664. ISSN: 0921-8831. DOI: https://doi.org/10.1016/j.apt.2022.103664. URL: https://www.sciencedirect.com/science/article/pii/S0921883122002424.

[81] L. E. Silbert et al. "Granular flow down an inclined plane: Bagnold scaling and rheology". In: *Phys. Rev. E* 64 (2001), p. 051302.

[82] W. C. Swope et al. "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters". In: *The Journal of Chemical Physics* 76.1 (1982), pp. 637–649. DOI: 10.1063/1.442716. eprint: https://doi.org/10.1063/1.442716. URL: https://doi.org/10.1063/1.442716.

[83] A. Thornton et al. "Modeling of particle size segregation: Calibration using the discrete particle method". In: *Neuroethics* 23 (Jan. 2011). DOI: 10.1142/S0129183112400141.

[84] A. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 42.1 (1936), pp. 230–265. DOI: 10.2307/2268810.

[85] A. Utkarsh. "The ParaView Guide: A Parallel Visualization Application". In: *ACM Press* (2015).

[86] J.-L. Vay et al. "Warp-X: A new exascale computing platform for beam–plasma simulations". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 909 (2018). 3rd European Advanced Accelerator Concepts workshop (EAAC2017), pp. 476–479. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2018.01.035. URL: https://www.sciencedirect.com/science/article/pii/S0168900218300524.

[87] L. Verlet. "Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules". In: *Phys. Rev.* 159.1 (1967), pp. 98–103.

[88] S. Verma, G. Novati, and P. Koumoutsakos. "Efficient collective swimming by harnessing vortices through deep reinforcement learning". In: *Proceedings of the National Academy of Sciences* 115.23 (2018), pp. 5849–5854. DOI: 10.1073/pnas.1800923115. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1800923115. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1800923115.

[89] J. H. Walther and I. F. Sbalzarini. "Large-scale parallel discrete element simulations of granular flow". In: *Engineering Computations* 26.6 (2009), pp. 688–697.

[90] H. Wendland. "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree". In: *Advances in Computational Mathematics* 4.1 (Dec. 1, 1995), pp. 389–396. ISSN: 1572-9044. DOI: 10.1007/BF02123482. URL: https://doi.org/10.1007/BF02123482 (visited on 02/28/2022).

[91] S. Wolfram. *A New Kind of Science*. English. Wolfram Media, 2002. ISBN: 1579550088. URL: https://www.wolframscience.com.

[92] J. Xu et al. "Discrete particle methods for engineering simulation: Reproducing mesoscale structures in multiphase systems". In: *Resources Chemicals and Materials* 1.1 (2022), pp. 69–79. ISSN: 2772-4433. DOI: https://doi.org/10.1016/j.recm.2022.01.002. URL: https://www.sciencedirect.com/science/article/pii/S2772443322000022.

[93] R. Yokota et al. "Petascale turbulence simulation using a highly parallel fast multipole method on GPUs". en. In: *Computer Physics Communications* 184.3 (Mar. 2013). arXiv:1106.5273 [physics], pp. 445–455. ISSN: 00104655. DOI: 10.1016/j.cpc.2012.09.011. URL: http://arxiv.org/abs/1106.5273 (visited on 12/29/2022).

[94] R. Yokota and L. A. Barba. "FMM-based vortex method for simulation of isotropic turbulence on GPUs, compared with a spectral method". en. In: *Computers & Fluids* 80 (July 2013). arXiv:1110.2921 [physics], pp. 17–27. ISSN: 00457930. DOI: 10.1016/j.compfluid.2012.08.002. URL: http://arxiv.org/abs/1110.2921 (visited on 12/29/2022).

[95] V. Zago et al. "Semi-implicit 3D SPH on GPU for lava flows". In: *Journal of Computational Physics* 375 (Dec. 15, 2018), pp. 854–870. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.07.060. URL: https://www.sciencedirect.com/science/article/pii/S002199911830593X (visited on 03/01/2022).