IEEE Open Journal of the Computer Society

Received 3 February 2023; accepted 1 March 2023. Date of publication 8 March 2023;
date of current version 13 April 2023. The review of this article was arranged by Associate Editor Quan Chen.

Digital Object Identifier 10.1109/OJCS.2023.3254466

A Unifying Mathematical Definition of Particle Methods

JOHANNES PAHLKE [1,2,3] AND IVO F. SBALZARINI [1,2,3,4,5]

[1]Technische Universität Dresden, Faculty of Computer Science, 01187 Dresden, Germany
[2]Max Planck Institute of Molecular Cell Biology and Genetics, 01307 Dresden, Germany
[3]Center for Systems Biology Dresden, 01307 Dresden, Germany
[4]Cluster of Excellence Physics of Life, TU Dresden, 01307 Dresden, Germany
[5]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), 01187 Dresden, Germany

CORRESPONDING AUTHOR: IVO F. SBALZARINI (e-mail: ivo.sbalzarini@tu-dresden.de)

This work was supported in part by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907).
This article has supplementary downloadable material available at https://doi.org/10.1109/OJCS.2023.3254466, provided by the authors.

ABSTRACT Particle methods are a widely used class of algorithms for computer simulation of complex phenomena in various fields, such as fluid dynamics, plasma physics, molecular chemistry, and granular flows, using diverse simulation methods, including Smoothed Particle Hydrodynamics (SPH), Particle-in-Cell (PIC) methods, Molecular Dynamics (MD), and Discrete Element Methods (DEM). Despite the increasing use of particle methods driven by improved computing performance, the relation between these algorithms remains formally unclear, and a unifying formal definition of particle methods is lacking. Here, we present a rigorous mathematical definition of particle methods and demonstrate its importance by applying it to various canonical and non-canonical algorithms, using it to prove a theorem about multi-core parallelizability, and designing a principled scientific computing software based on it. We anticipate that our formal definition will facilitate the solution of complex computational problems and the implementation of understandable and maintainable software frameworks for computer simulation.

INDEX TERMS Algorithmics, formal definition, mesh-free methods, particle methods, simulation algorithms, parallelization, software engineering.

I. INTRODUCTION

Particle methods are a classic and widespread class of algorithms for computer simulation, with applications ranging from computational plasma physics [19] to computational fluid dynamics [11]. Historically, some of the first computer simulations in these domains used particle methods [8], [37], and the field is still under active development [6], [9]. A key advantage of particle methods is their versatility, as they can simulate both discrete and continuous phenomena stochastically or deterministically.

In simulations of discrete models, particles naturally represent the discrete entities of the model, such as atoms in molecular dynamics simulations [2], cars in simulations of road traffic [28], or grains of sand in discrete-element simulations of granular flows [38]. When simulating continuous models or numerically solving differential equations, particles represent mathematical collocation or Lagrangian tracer points of the discretization of the continuous fields [10], [32]. The evaluation of differential operators on these fields can directly be approximated on the particles using numerical methods such as Smoothed Particle Hydrodynamics (SPH) [16], [30], Reproducing Kernel Particle Methods (RKPM) [27], Particle Strength Exchange (PSE) [12], [14], and Discretization-Corrected PSE (DC-PSE) [4], [36]. 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 [19]. In addition to their versatility, particle methods can efficiently be parallelized on shared- and distributed-memory computers [20], [21], [22], [33], [35]. Furthermore, they simplify simulations in complex [34] and time-varying [3] geometries, as no computational mesh needs to be generated and maintained. Beyond the field of simulations, structurally similar algorithms have been developed, such as particle-based

image processing methods [1], [7], point-based computer graphics [17], and computational optimization algorithms using point samples [18], [31].

Even though all these methods share structural similarities, 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," which have not yet been rigorously described and rationalized as standalone concepts or together to formally define particle methods as an algorithmic class.

Formal mathematical descriptions, models, or definitions provide a rigorous way of formulating concepts and form the basis for well-founded scientific discussions in computer science. They allow more profound insights into the described subject based on mathematical theorems that are incontrovertible under the given definition. Mathematical theorems in computer science are extremely valuable, for example, to represent and query knowledge with the rigorous mathematical structure of answer set programming [15], [25], or for runtime complexity studies of algorithms in automata theory [5].

Despite their broad use, a formal mathematical definition of particle methods that covers all structural similarities of these algorithms from different fields is lacking. Hence, what constitutes a particle method, and what does not, remains unclear, hindering potential development in various research areas.

Here, we fill this knowledge gap by presenting a formal definition that unifies all particle methods and enables the design of novel particle methods also for non-canonical problems. We showcase this by formulating various algorithms as particle methods, ranging from classic SPH over molecular dynamics to Gaussian elimination. Furthermore, we use our definition for a theoretical analysis of the parallelizability of particle methods on multi-core computer systems and for designing and implementing a principled software framework for particle methods.

The presented definition of particle methods and its applications open up a new way of investigating and developing computer simulation algorithms and software systems based on these principles.

II. NOMENCLATURE AND NOTATION

We introduce the notation and nomenclature used and define the underlying mathematical concepts. We use, **bold symbols** for tuples of arbitrary length, e.g. $\mathbf{p} \in P^*$, **regular symbols with subscript indices** for the elements of these tuples, e.g. $\mathbf{p} = (p_1, \ldots, p_n)$, **vertical bars** around a tuple is the number of elements it contains, e.g., $|\mathbf{p}| := n \in \mathbb{N}$, **regular symbols** for tuples of determined length with specific element names, e.g., $p = (a, b, c) \in A \times B \times C$, an **indexed tuple** of determined length with named elements, e.g. $p_j = (a_j, b_j, c_j)$, **underline** for vectors, e.g. $\underline{v} \in A^n$, and the **length of a vector** if $A = \mathbb{R}$, $|\underline{v}| := \sqrt{v_1^2 + \ldots + v_n^2}$.

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:

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

$$A^* := \bigcup_{j=0}^{\infty} A^j. \tag{2}$$

Definition 2: The **composition operator** $*_h$ of a binary function h:

Be $h : A \times B \to A$ then $*_h : A \times B^* \to A$ is recursively defined as:

$$a *_h () := a \tag{3}$$

$$a *_h (b_1, b_2, \ldots, b_n) := h(a, b_1) *_h (b_2, \ldots, b_n). \tag{4}$$

Definition 3: The **concatenation** $\circ : A^* \times A^* \to A^*$ of tuples $(a_1, \ldots, a_n), (b_1, .., b_m) \in A^*$ is defined as:

$$(a_1, \ldots, a_n) \circ (b_1, .., b_m) := (a_1, \ldots, a_n, b_1, .., b_m). \tag{5}$$

Definition 4: We define the **construction of a subtuple** $\mathbf{b} \in A^*$ of $\mathbf{a} \in A^*$. Be $f : A^* \times \mathbb{N} \to \{\top, \bot\}$ ($\top = true$, $\bot = false$) 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:

$$\begin{aligned} & \mathbf{b} = (a_j \in \mathbf{a} : f(\mathbf{a}, j)) := (a_{j_1}, \ldots, a_{j_n}) \\ \Leftrightarrow \quad & \mathbf{a} = (a_1, \ldots, a_{j_1}, \ldots, a_{j_2}, \ldots, a_{j_n}, \ldots, a_m) \\ & \wedge \quad \forall k \in \{1, .., n\} :\ f(\mathbf{a}, j_k) = \top. \end{aligned} \tag{6}$$

Definition 5: Be $\alpha = (a_1, \ldots., a_n) \in A_1 \times \ldots \times A_n$ a potentially inhomogeneous tuple of length n, and being $j_1, \ldots, j_m \in \{1, 2, \ldots, n\}$ then

$$\langle \alpha \rangle_{(j_1, j_2, \ldots, j_m)} := (a_{j_1}, a_{j_2}, \ldots, a_{j_m}). \tag{7}$$

III. DEFINITION OF PARTICLE METHODS

To develop a formal definition of the algorithmic class of particle methods, we choose the specific formulation that naturally relates to the structural elements of practical implementations, i.e., to the methods and subroutines often found in particle methods-related literature and software. More specifically, our definition is based on observing and analyzing many existing particle methods and distilling their structural commonalities. The structural commonalities are also reflected in the terminology. "Particles" or "points" are the basic objects that get manipulated throughout the algorithms [1], [11], [17], [19], [31], [32], [37], [38]. "Particles" carry algorithm specific attributes or properties. These "particles" can "interact" [17], [28], [32], [37], [38] with each other within a "neighborhood," "surrounding," or "sampling radius" [1], [17], [19], [28], [31], [32], [37], [38]. In addition "Particles" can "evolve" [1], [17], [31], [32], [38]. The term "evolve" is used inconsistently. It is referred to as the change of a single "particle" depending only on its properties or the change of the whole system of "particles". The change of the whole system is more often called "step," "time step" or "state transition" [11], [17], [19], [28], [31], [32], [37], [38].

Using some of these terms and their underlying concepts, we subdivide our definition of particle methods into three

parts: First, the definition of the general particle method algorithm structure, including structural components, namely data structures and functions. 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 first state to the last 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. For deeper explanations and examples, the reader is referred to the supplementary material SM1.

A. 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 [24]. Both concepts describe state-transition systems working on discrete objects.

Definition 6: A **particle method algorithm** is a 7-tuple $(P, G, u, f, i, e, \mathring{e})$, consisting of the two data structures

$$P := A_1 \times A_2 \times \ldots \times A_n \quad \text{the particle space,} \tag{8}$$

$$G := B_1 \times B_2 \times \ldots \times B_m \quad \text{the global variable space,} \tag{9}$$

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

$$u : [G \times P^*] \times \mathbb{N} \to \mathbb{N}^* \quad \text{the neighborhood function,} \tag{10}$$

$$f : G \to \{\top, \bot\} \quad \text{the stopping condition,} \tag{11}$$

$$i : G \times P \times P \to P \times P \quad \text{the interact function,} \tag{12}$$

$$e : G \times P \to G \times P^* \quad \text{the evolve function,} \tag{13}$$

$$\mathring{e} : G \to G \quad \text{the evolve function of the global variable.} \tag{14}$$

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

B. 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 7: An initial state defines a **particle method instance** for a given particle method algorithm $(P, G, u, f, i, e, \mathring{e})$:

$$\left[g^1, \mathbf{p}^1\right] \in [G \times P^*]. \tag{15}$$

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

C. PARTICLE STATE TRANSITION FUNCTION

In a specific particle method, the elements of the tuple $(P, G, u, f, i, e, \mathring{e})$ (8)–(14) 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 initial state $[g^1, \mathbf{p}^1]$. 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 8: We define the *state transition function*

$$S : [G \times P^*] \to [G \times P^*] \tag{16}$$

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

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

$$\iota^{\mathrm{I}}_{(g,j)}(\mathbf{p}, k) := \big(p_1, .., p_{j-1}, \overline{p}_j, p_{j+1}, \ldots, p_{k-1}, \overline{p}_k, p_{k+1}, \ldots, p_{|\mathbf{p}|}\big), \quad \text{for } \big(\overline{p}_j, \overline{p}_k\big) := i\big(g, p_j, p_k\big). \tag{17}$$

The second interact sup-function $\iota^{\mathrm{I\times U}}$ 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 $\mathbf{p}$,

$$\iota^{\mathrm{I\times U}}_{g}(\mathbf{p}, j) := \mathbf{p} \ *_{\iota^{\mathrm{I}}_{(g,j)}} \ u([g, \mathbf{p}], j) \tag{18}$$

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

$$\iota^{\mathrm{N\times U}}([g, \mathbf{p}]) := \mathbf{p} \ *_{\iota^{\mathrm{I\times U}}_{g}} (1, .., |\mathbf{p}|) \tag{19}$$

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

$$\epsilon^{\mathrm{I}}_{\mathbf{p}}([g, \mathbf{q}], j) := \left[\overline{g}, \mathbf{q} \circ \overline{\mathbf{q}}\right] \ \text{ for } (\overline{g}, \overline{\mathbf{q}}) := e(g, p_j). \tag{20}$$

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

$$\epsilon^{\mathrm{N}}([g, \mathbf{p}]) := \left[g, ()\right] \ *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}} \ (1, .., |\mathbf{p}|) \tag{21}$$

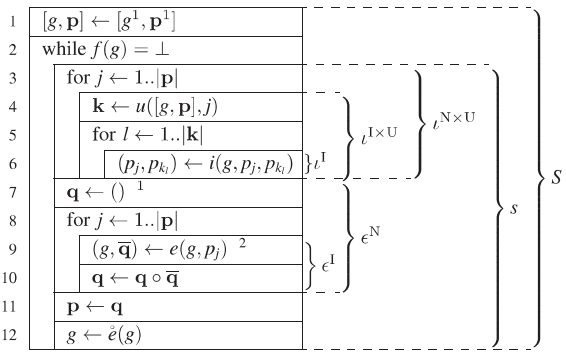


FIGURE 1. Nassi-Shneiderman diagram of the state transition function S with annotated sub-functions.

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

$$s\left([g,\mathbf{p}]\right) := \left[\mathring{e}(\overline{g}),\ \overline{\mathbf{p}}\right] \text{ for } \left[\overline{g},\overline{\mathbf{p}}\right] := \epsilon^{\mathrm{N}}\left([g,\ \iota^{\mathrm{N}\times\mathrm{U}}([g,\mathbf{p}])\right). \tag{22}$$

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

$$\begin{aligned} S\left(\left[g^1,\mathbf{p}^1\right]\right) = \left[g^T,\mathbf{p}^T\right] \quad \Longleftrightarrow \quad & f\left(g^T\right) = \top \ \wedge \\ & \forall t \in \{2,\ldots,T\}: \\ & \left[g^t,\mathbf{p}^t\right] = s\left(\left[g^{t-1},\mathbf{p}^{t-1}\right]\right) \ \wedge \ f\left(g^{t-1}\right) = \bot. \end{aligned} \tag{23}$$

We illustrate the state transition function and its sub-functions with the Nassi-Shneiderman diagram in Fig. 1.

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.

IV. APPLICATIONS OF THE DEFINITION OF PARTICLE METHODS

After having defined particle methods, a pivotal question arises: How can we leverage this mathematical definition in practice? To address this question, we focus on three main applications of this definition: First, we illustrate how several canonical and non-canonical particle methods can be formalized in the notational framework of the definition. Second, we use our definition to prove a theorem about the parallelization of particle methods formally. Third, we show the practical applicability of our definition by using it for designing and implementing scientific computing software.

A. KNOWN ALGORITHMS AS PARTICLE METHODS

We demonstrate how our definition provides a unifying notation for various canonical and non-canonical particle methods. Here, we present a three-dimensional simulation of the continuum diffusion equation (24).

We refer the reader to the supplementary material for more examples. As canonical particle methods, we present there an n-dimensional perfectly elastic collision simulation (SM2.1), a three-dimensional Smoothed Particle Hydrodynamics (SPH) [16], [29] fluid simulation (SM2.2), and a one-dimensional Lennard-Jones [26] molecular dynamics (MD) simulation (SM2.3). As non-canonical examples, we present there how triangulation refinement (SM2.4) and Gaussian elimination (SM2.5) can be formulated as particle methods. We thereby illustrate the unifying nature of the presented definition.

Particle Strength Exchange (PSE) is a classic particle method that numerically solves partial differential equations in time and space [12], [13], [14], [36]. 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:

$$\frac{\partial w(\underline{x},t)}{\partial t} = D\ \mathbf{\Delta} w(\underline{x},t) \tag{24}$$

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 [13]:

$$\mathbf{\Delta} w\ (\underline{x}_j) \approx \frac{h^3}{\epsilon^2}\sum_{k=1}^{N}\Big(w(\underline{x}_k) - w(\underline{x}_j)\Big)\,\eta_\epsilon(\underline{x}_j - \underline{x}_k). \tag{25}$$

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:

$$\eta_\epsilon(\underline{x}) = \frac{15}{\epsilon^3\pi^2}\frac{1}{\left(\frac{|\underline{x}|}{\epsilon}\right)^{10}+1}. \tag{26}$$

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:

$$\mathbf{\Delta} w\ (\underline{x}_j) \approx \frac{15\,h^3}{\epsilon^3\pi^2}\sum_{\underline{x}_k:\,0<|\underline{x}_k-\underline{x}_j|\le r_c}\frac{w(\underline{x}_k)-w(\underline{x}_j)}{\left(\frac{|\underline{x}_k-\underline{x}_j|}{\epsilon}\right)^{10}+1}. \tag{27}$$

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 $\Delta t := t_{n+1}-t_n$:

$$\frac{\partial w}{\partial t}(t_n) \approx \frac{w(t_{n+1})-w(t_n)}{\Delta t}. \tag{28}$$

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

Hence, the above differential equation is discretized as:

$$w(\underline{x}_j, t_{n+1}) \tag{29}$$

$$\approx w(\underline{x}_j, t_n) + D\,\Delta t\;\mathbf{\Delta} w(\underline{x}_j) \tag{30}$$

$$\approx w(\underline{x}_j, t_n) + \frac{15\,h^3 D \Delta t}{\epsilon^3 \pi^2} \sum_{x_k : 0 < |\underline{x}_k - \underline{x}_j| \le r_c} \frac{w(\underline{x}_k, t_n) - w(\underline{x}_j, t_n)}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1}. \tag{31}$$

To numerically solve (24), 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 $\frac{h}{\epsilon} \le 1$. 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:

$$p := \left(\underline{x},\, w,\, \Delta w\right) \quad \text{for } p \in P := \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R}, \tag{32}$$

$$g := \left(D, h, \epsilon, r_c, \Delta t, t_{end}, t\right) \quad \text{for } g \in \mathbb{R}^7, \tag{33}$$

and functions:

$$u([g, \mathbf{p}], j) := (k \in (1, \ldots, |\mathbf{p}|) : \; p_k, p_j \in \mathbf{p} \wedge |\underline{x}_k - \underline{x}_j| \in (1, r_c]), \tag{34}$$

$$f(g) := \left(t > t_{end}\right), \tag{35}$$

$$i\left(g, p_j, p_k\right) := \left(\begin{pmatrix} \underline{x}_j \\ w_j \\ \Delta w_j + \frac{w_k - w_j}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1} \end{pmatrix}^{\mathbf{T}},\; p_k\right), \tag{36}$$

$$e\left(g, p_j\right) := \left(g, \left(\begin{pmatrix} \underline{x}_j \\ w_j + \Delta t \frac{15 D h^3}{\epsilon^5 \pi^2} \Delta w_j \\ 0 \end{pmatrix}^{\mathbf{T}}\right)\right), \tag{37}$$

$$\mathring{e}(g) := \left(D, h, \epsilon, r_c, \Delta t, t_{end}, t + \Delta t\right). \tag{38}$$

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 (27). 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 (31). 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.

$$g^1 := \left(\underbrace{0.01}_{=D},\, \underbrace{0.02}_{=h},\, \underbrace{0.02}_{=\epsilon},\, \underbrace{0.06}_{=r_c},\, \underbrace{0.005}_{=\Delta t},\, \underbrace{0.5}_{=t_{end}},\, \underbrace{0}_{=t}\right) \tag{39}$$

$$\mathbf{p}^1 := (p_1, \ldots, p_{51^3}), \qquad p_j := (\underline{x}_j, w_j, \Delta w_j) \tag{40}$$

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

$$p_j := \left((h(j_1 - 25), h(j_2 - 25), h(j_3 - 25)), 0, 0\right), \tag{41}$$

and we set

$$p_{\frac{51^3+1}{2}} := \left((0, 0, 0), h^{-3}, 0\right). \tag{42}$$

The result of executing this instance is visualized for time $t = 0.5$ in the Figs. 4(b) and 2.

B. THEORY ABOUT PARTICLE METHODS

In addition to formalizing algorithms, our definition can also be used to prove theorems about particle methods. We demonstrate this by proving that one can parallelize the state transition function on a shared memory system under five conditions. After verifying that a particle method fulfills these conditions, we know directly if it is parallelizable *in principle* with this parallelization scheme. We exemplify this on the particle methods application for the simulation of the three-dimensional diffusion (Section IV-A).

We assume that parallel reading from one storage location is possible but not parallel writing for a shared memory system. Hence, we need to ensure that the calculations can be done in parallel and that we do not have race conditions while writing. Therefore, we need the five conditions:

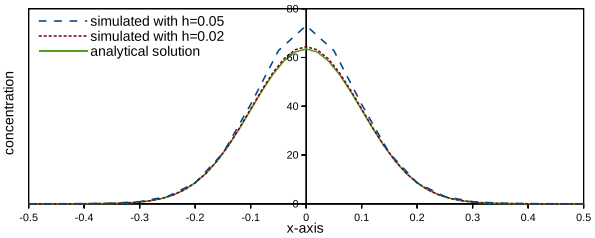


FIGURE 2. **Particle Strength Exchange (PSE) simulation of diffusion in three dimensions, compared with the analytical solution at time $t = 0.5$. See main text for problem description. The plot shows the concentration values along the x-axis.**

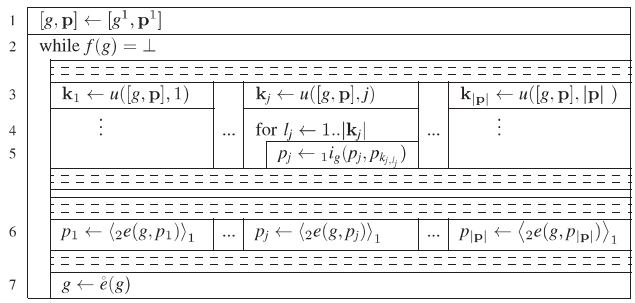


FIGURE 3. **Nassi-Shneiderman diagram of the parallelized state-transition function for shared-memory systems. The dashed lines enclose the parallel part.**

condition one, pull-interaction

$$i(g, p_j, p_k) = (\overline{p}_j, p_k), \tag{43}$$

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

Condition two, interaction independence of previous interactions

$$\begin{aligned} {}_1i_g({}_1i_g(p_j, p_k), p_{k'})) &= {}_1i_g(p_j, p_k) \\ \text{for } {}_1i_g(p_j, p_k) &:= \langle i(g, p_j, p_k)\rangle_1, \end{aligned} \tag{44}$$

condition three, neighborhood independence of previous interactions

$$u([g, \mathbf{p}], j) = u\left(\left[g, \mathbf{p} *^{\mathrm{I}}_{\iota_{(g,k')}} (k'')\right], j\right), \tag{45}$$

condition four, constant number of particles

$$e(g, p) = (\overline{g}, (\overline{p})), \tag{46}$$

condition five, global variable independence of particles

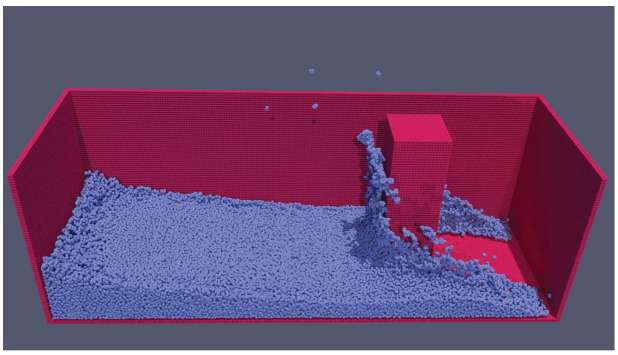
$$e(g, p) = (g, \mathbf{q}). \tag{47}$$

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

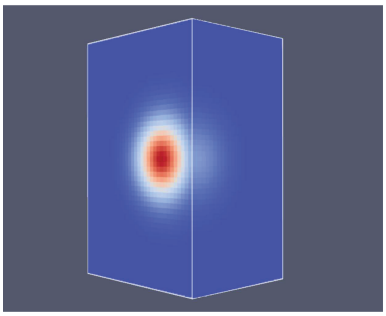
$$\mathbf{k}_j = u([g, \mathbf{p}], j), \tag{48}$$

and $[\text{for } l_j \leftarrow 1..|\mathbf{k}_k|]$ together with $[p_j \leftarrow {}_1i_g(p_j, p_{k_{j,l_j}})]$ (lines 4, 5) to

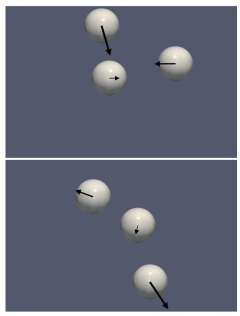
$$\overline{p}_j = p_j *_{1i_g} \left(p_{\langle \mathbf{k}_j\rangle_1}, \ldots, p_{\langle \mathbf{k}_j\rangle_{|\mathbf{k}_j|}}\right). \tag{49}$$



(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 4. **Experiments with the prototype software.**

Combining (48) with (49) for all particles results in the tuple

$$\left(p_1 *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},1)}, \ \ldots, \ \ p_{|\mathbf{p}|} *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right). \tag{50}$$

All particles are constantly overwritten at that part of the scheme. Also the next part $[p_1 \leftarrow \langle {}_2e(g, p_1)\rangle_1]$ (line 6) overwrites each particle. Hence, inside these parts are potential writing conflicts, but not between them because they are parallel. Therefore, we can take them together to get

$$\left(\left\langle {}_2e\left(p_1 *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},1)}\right)\right\rangle_1, \ldots, \ \left\langle {}_2e\left(p_{|\mathbf{p}|} *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)\right\rangle_1\right). \tag{51}$$

Adding to this, the evolution of the global variable function results in a parallelized step of the particle method state transition function. The rest of the state transition is identical to the sequential state transition. Hence, to prove that the Nassi-Shneiderman diagrams (Figs. 1 and 3) have the same result, it is sufficient to prove the statement

$$\begin{aligned} &\forall [g, \mathbf{p}] \in [G, P^*] : s([g, \mathbf{p}]) \\ &= \Big[\mathring{e}(g), \Big(\left\langle {}_2e\left(p_1 *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},1)}\right)\right\rangle_1, \\ &\quad \ldots, \quad \left\langle {}_2e\left(p_{|\mathbf{p}|} *_{1i_g} \langle \mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)\right\rangle_1\Big)\Big]. \end{aligned} \tag{52}$$

The proof relies on the five lemmas.

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

$$\forall p_j, p_k \in P, g \in G : i(g, p_j, p_k) = (\overline{p}_j, p_k)$$
$$\implies \quad \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,j)}} (k_1, \ldots, k_n)$$
$$= \left(p_1, \ldots, \ p_j *_{{}_1 i_g} (p_{k_1}, \ldots, p_{k_n}) , \ldots, p_{|\mathbf{p}|}\right), \tag{53}$$

Proof:

$$\forall p_j, p_k \in P, g \in G : i(g, p_j, p_k) = (\overline{p}_j, p_k) \tag{54}$$
$$\implies$$
$$\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,j)}} (k_1, \ldots, k_n) \tag{55}$$
$$= \left(p_1, \ldots, {}_1 i(g, p_j, p_{k_1}), \ldots, p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} (k_2, \ldots, k_n) \tag{56}$$
$$= \left(p_1, \ldots, {}_1 i_g \left({}_1 i_g (p_j, p_{k_1}), p_{k_2}\right) , \ldots, p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} (k_3, \ldots, k_n) \tag{57}$$
$$= \left(p_1, \ldots, \ p_j *_{{}_1 i_g} (p_{k_1}, p_{k_2}) , \ldots, p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} (k_3, \ldots, k_n) \tag{58}$$
$$= \left(p_1, \ldots, \ p_j *_{{}_1 i_g} (p_{k_1}, \ldots, p_{k_n}) , \ldots, p_{|\mathbf{p}|}\right) *_{\iota^{\mathrm{I}}_{(g,j)}} () \tag{59}$$
$$\underline{= \left(p_1, \ldots, \ p_j *_{{}_1 i_g} (p_{k_1}, \ldots, p_{k_n}) , \ldots, p_{|\mathbf{p}|}\right).} \tag{60}$$

■

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.

$$\forall p_j, p_k, p_{k'} \in P, g \in G : {}_1 i_g(p_j, {}_1 i_g(p_k, p_{k'})) = {}_1 i_g(p_j, p_k)$$
$$\implies {}_1 i_g \left(p_j, \ p_k *_{{}_1 i_g} \left(p_{k'_1}, \ldots, p_{k'_n}\right)\right) = {}_1 i_g \left(p_j, \ p_k\right) \tag{61}$$

Proof:

$$\forall p_j, p_k, p_{k'} \in P, g \in G : {}_1 i_g \left(p_j, {}_1 i_g (p_k, p_{k'})\right) = {}_1 i_g (p_j, p_k) \tag{62}$$
$$\implies {}_1 i_g \left(p_j, \ p_k *_{{}_1 i_g} \left(p_{k'_1}, \ldots, p_{k'_n}\right)\right) \tag{63}$$
$$= {}_1 i_g \left(p_j, \ \left(p_k *_{{}_1 i_g} \left(p_{k'_1}, \ldots, p_{k'_{n-1}}\right)\right) *_{{}_1 i_g} (p_{k'_n})\right) \tag{64}$$
$$= {}_1 i_g \left(p_j, \ {}_1 i_g \left(p_k *_{{}_1 i_g} \left(p_{k'_1}, \ldots, p_{k'_{n-1}}\right) , \ p_{k'_n}\right)\right) \tag{65}$$
$$= {}_1 i_g \left(p_j, \ p_k *_{{}_1 i_g} \left(p_{k'_1}, \ldots, p_{k'_{n-1}}\right)\right) \tag{66}$$
$$\vdots$$
$$= {}_1 i_g \left(p_j, \ p_k *_{{}_1 i_g} ()\right) \tag{67}$$
$$\underline{= {}_1 i_g \left(p_j, \ p_k\right).} \tag{68}$$

■

Lemma 3: The neighborhood needs to be interaction-independent, so the neighbors on the processors do not leak a possibly changed particle.

$$\forall j, k, {}' k' \in \{1, \ldots, |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*] :$$
$$u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j)$$
$$\implies u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,l_1)}} (k_{1,1}, \ldots, k_{1,n_1}) *_{\iota^{\mathrm{I}}_{(g,l_2)}} \ldots *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,1}, \ldots, k_{m,n_1})], j)$$
$$= u([g, \mathbf{p}], j) \tag{69}$$

Proof:

$$\forall j, k, {}' k' \in \{1, \ldots, |\mathbf{p}|\}, [g, \mathbf{p}] \in [G \times P^*] :$$
$$u([g, \mathbf{p}], j) = u([g, \mathbf{p} *_{\iota^{\mathrm{I}}_{(g,k')}} (k'')], j) \implies$$
$$u([g, \underbrace{\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,l_1)}} (k_{1,1}, \ldots, k_{1,n_1}) *_{\iota^{\mathrm{I}}_{(g,l_2)}} \ldots *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,1}, \ldots, k_{m,n_1})}_{=:\mathbf{q}}], j) \tag{70}$$
$$= u([g, \underbrace{\mathbf{q} *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,1}, \ldots, k_{m,n_1})}_{=:\overline{\mathbf{q}}}], j) \tag{71}$$
$$= u([g, \overline{\mathbf{q}} *_{\iota^{\mathrm{I}}_{(g,l_m)}} (k_{m,n_1})], j) \tag{72}$$
$$\underline{= u([g, \mathbf{p}], j).} \tag{73}$$

■

Lemma 4: Under the constraints that the interact function i is a pull-interaction (43) and is independent of the previous interactions (44) and the neighborhood is independent of previous interactions (45), the outer interaction loop is parallelizable. In our notation:

$$\forall [g, \mathbf{p}] \in [G, P^*] : \ \iota^{\mathrm{N} \times \mathrm{U}} ([g, \mathbf{p}]) = \left(p_1 *_{{}_1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}, \ldots, p_{|\mathbf{p}|} *_{{}_1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right). \tag{74}$$

Proof:

$$\iota^{\mathrm{N} \times \mathrm{U}} ([g, \mathbf{p}]) \tag{75}$$
$$= \mathbf{p} *_{\iota^{\mathrm{I} \times \mathrm{U}}_g} (1, \ldots, |\mathbf{p}|) \tag{76}$$
$$= \left(\mathbf{p} *_{\iota^{\mathrm{I}}_{(g,1)}} u(g, \mathbf{p}, 1)\right) *_{\iota^{\mathrm{I} \times \mathrm{U}}_g} (2, \ldots, |\mathbf{p}|) \tag{77}$$
$$\overset{\text{Lemma 1}}{=} \underbrace{\left(\underbrace{p_1 *_{{}_1 i_g} \langle \mathbf{p} \rangle_{u(g,\mathbf{p},1)}}_{\overline{p}_1}, \ p_2, \ldots, p_{|\mathbf{p}|}\right)}_{=: {}^1\overline{\mathbf{p}}} *_{\iota^{\mathrm{I} \times \mathrm{U}}_g} (2, \ldots, |\mathbf{p}|) \tag{78}$$
$$= \left({}^1\overline{\mathbf{p}} *_{\iota^{\mathrm{I}}_{(g,2)}} u(g, {}^1\overline{\mathbf{p}}, 2)\right) *_{\iota^{\mathrm{I} \times \mathrm{U}}_g} (3, \ldots, |\mathbf{p}|) \tag{79}$$

$$\overset{\text{Lemma 1}}{=}\underbrace{\Big(\overline{p}_1,\underbrace{p_2 *_{1 i_g}\langle {}^1\overline{\mathbf{p}}\rangle_{u(g,{}^1\overline{\mathbf{p}},2)}}_{\overline{p}_2},\ p_3,\ldots,p_{|\mathbf{p}|}\Big)}_{=:\,{}^2\overline{\mathbf{p}}} *_{\iota_g^{\mathrm{I}\times\mathrm{U}}}(3,\ldots,|\mathbf{p}|) \tag{80}$$

$$\vdots$$

$$=\left({}^{j-1}\overline{\mathbf{p}} *_{\iota^{\mathrm{I}}_{(g,j)}} u(g,{}^{j-1}\overline{\mathbf{p}},j)\right) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}}(j+1,\ldots,|\mathbf{p}|) \tag{81}$$

$$\overset{\text{Lemma 1}}{=}\left(\overline{p_1},\ \overline{p_2},\ldots,\ p_j *_{1 i_g}\langle {}^{j-1}\overline{\mathbf{p}}\rangle_{u(g,\,{}^{j-1}\overline{\mathbf{p}},j)},\ \ldots,\ p_{|\mathbf{p}|}\right) \tag{82}$$

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

$$\overset{\text{Lemma 3}}{=}\left(\overline{p}_1,\ \overline{p}_2,\ \ldots,\ p_j *_{1 i_g}\langle {}^{j-1}\overline{\mathbf{p}}\rangle_{u(g,\mathbf{p},j)},\ \ldots,\ p_{|\mathbf{p}|}\right) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}}(j+1,\ldots,|\mathbf{p}|) \tag{83}$$

$$\overset{\text{Lemma 2}}{=}\left(\overline{p}_1,\ \overline{p}_2,\ \ldots,\ p_j *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},j)},\ \ldots,\ p_{|\mathbf{p}|}\right) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}}(j+1,\ldots,|\mathbf{p}|) \tag{84}$$

$$=\left(p_1 *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},1)},\ p_2 *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},2)},\ \ldots,\ p_j *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},j)},\ \ldots,p_{|\mathbf{p}|}\right) *_{\iota_g^{\mathrm{I}\times\mathrm{U}}}(j+1,\ldots,|\mathbf{p}|) \tag{85}$$

$$=\underline{\left(p_1 *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},1)},\ \ldots\ p_{|\mathbf{p}|} *_{1 i_g}\langle\mathbf{p}\rangle_{u(g,\mathbf{p},|\mathbf{p}|)}\right)} \tag{86}$$

■

Lemma 5: Under the constraints that the number of particles stays constant (46) and the global variable is independent of all particles (47), the evolve loop is parallelizable. In our notation:

$$\epsilon^{\mathrm{N}}([g,\mathbf{p}])=\left[g,\ \left(\langle\, {}_2e(p_1)\rangle_1,\ldots,\left\langle\, {}_2e(p_{|\mathbf{p}|})\right\rangle_1\right)\right] \tag{87}$$

Proof: Under the constraints (46) and (47), we can rewrite the first evolution sub-function (20) to

$$e^I([g,\mathbf{p}],\mathbf{q},j)=\left[g,\mathbf{q}\circ\left(\langle\, {}_2e(g,p_j)\rangle_1\right)\right]. \tag{88}$$

Using this result, we can rewrite the second evolution sub-function (21) to

$$\epsilon^{\mathrm{N}}([g,\mathbf{p}]) \tag{89}$$

$$=\left[g,()\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(1,..,|\mathbf{p}|) \tag{90}$$

$$=\epsilon^{\mathrm{I}}([g,\mathbf{p}],(),1) *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(2,\ldots,|\mathbf{p}|) \tag{91}$$

$$=\left[g,()\circ\left(\langle\, {}_2e(g,p_1)\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(2,\ldots,|\mathbf{p}|) \tag{92}$$

$$=\epsilon^{\mathrm{I}}\left([g,\mathbf{p}],\left(\langle\, {}_2e(g,p_1)\rangle_1\right),2\right) *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(3,\ldots,|\mathbf{p}|) \tag{93}$$

$$=\left[g,\left(\langle\, {}_2e(g,p_1)\rangle_1\right)\circ\left(\langle\, {}_2e(g,p2)\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(3,\ldots,|\mathbf{p}|) \tag{94}$$

$$=\left[g,\left(\langle\, {}_2e(g,p_1)\rangle_1,\langle\, {}_2e(g,p2)\rangle_1\right)\right] *_{\epsilon^{\mathrm{I}}_{\mathbf{p}}}(3,\ldots,|\mathbf{p}|) \tag{95}$$

$$\vdots$$

$$=\left[g,\ \left(\langle\, {}_2e(p_1)\rangle_1,\ldots,\left\langle\, {}_2e(p_{|\mathbf{p}|})\right\rangle_1\right)\right] \tag{96}$$

■

Using these interim results, we can prove that the parallelization scheme in Fig. 3 produces identical results to the sequential state transition function.

Proof: Therefore, we need to prove (52). We use Lemma 4 and Lemma 5 to prove it. We insert (87) and (74) into (22) and get directly (52).

B. 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 (46) restricts the number of particles to be constant over all state transition steps

$$\forall t\in\{1,\ldots,T\}:\left|\mathbf{p}^t\right|=\left|\mathbf{p}^1\right|. \tag{97}$$

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, $\tau_{\overset{\circ}{e}}$, 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

$$\tau_{S([g^1,\mathbf{p}^1])}\le T\left(\left|\mathbf{p}^1\right|(\varsigma_u^{\mathbf{p}^1}\tau_i+\tau_u^{\mathbf{p}^1}+\tau_e)+\tau_f+\tau_{\overset{\circ}{e}}\right), \tag{98}$$

and for the time complexity of the parallelized state transition function on n_{CPU} processors with shared memory

$$\tau_{n_{\mathrm{CPU}},S^{||}([g^1,\mathbf{p}^1])}\le T\left(\frac{|\mathbf{p}^1|}{n_{\mathrm{CPU}}}(\varsigma_u^{\mathbf{p}^1}\tau_i+\tau_u^{\mathbf{p}^1}+\tau_e)+\tau_f+\tau_{\overset{\circ}{e}}\right). \tag{99}$$

B. APPLICATION TO THE EXAMPLES

We can now use the theorem to check if we can parallelize the particle method from Section III-A.

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:

$$ {}_1i_g(p_j, {}_1i_g(p_k, p_{k'})) \tag{100} $$

$$ = {}_1i_g\left(p_j, \left(\underline{x}_k,\ w_k,\ \Delta w_k + \frac{(w_{k'} - w_k)}{\left(\frac{|\underline{x}_{k'} - \underline{x}_k|}{\epsilon}\right)^{10} + 1}\right)\right) \tag{101} $$

$$ = \left(\underline{x}_j,\ w_j,\ \Delta w_j + \frac{(w_k - w_j)}{\left(\frac{|\underline{x}_k - \underline{x}_j|}{\epsilon}\right)^{10} + 1}\right) \tag{102} $$

$$ =_1 i_g(p_j, p_k) \tag{103} $$

Independence of the neighborhood function from previous interactions.

$$ u([g, \mathbf{p} *_{t^{\mathrm{I}}_{(g,k')}}(k'')], j) \tag{104} $$

$$ = u([g, (p_1, \ldots, {}_1i_g(p_{k'}, p_{k''}), \ldots, p_{|\mathbf{p}|})], j) \tag{105} $$

$$ = u\left(\left[g, \left(p_1, \ldots, \begin{pmatrix} \underline{x}_{k'} \\ w_{k'} \\ \Delta w_{k'} + \frac{(w_{k''} - w_{k'})}{\left(\frac{|\underline{x}_{k''} - \underline{x}_{k'}|}{\epsilon}\right)^{10} + 1} \end{pmatrix}^{\mathbf{T}}, \ldots, p_{|\mathbf{p}|}\right)\right], j\right) \tag{106} $$

$$ = (k \in (1, \ldots, |\mathbf{p}|) : p_k, p_j \in \mathbf{p} \wedge 0 < |\underline{x}_k - \underline{x}_j| \leq r_c) \tag{107} $$

$$ = u([g, \mathbf{p}], j) \tag{108} $$

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 the shared memory scheme.

C. A BASIS FOR SCIENTIFIC SOFTWARE ENGINEERING

Our definition can be leveraged to implement scientific computing software where the particle method algorithm and instance are used as an interface. This allows hiding all generic parts of a particle method from the user, such as finding the neighborhood and running the state transition function. Such software could potentially also encapsulate theoretical results, such as the result from Section III-B. We showcase the hiding of the neighbor search and the state transition function by designing a software prototype where we implement fast neighbor access 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 exemplarily for SPH (Fig. 4(a)), 3D diffusion (Fig. 4(b)), and the n-dimensional perfectly elastic collision (Fig. 4(c)) algorithms, but the prototype is not limited to these three examples. In addition, we validated the software, particularly the runtime complexity concerning the fast neighbor access

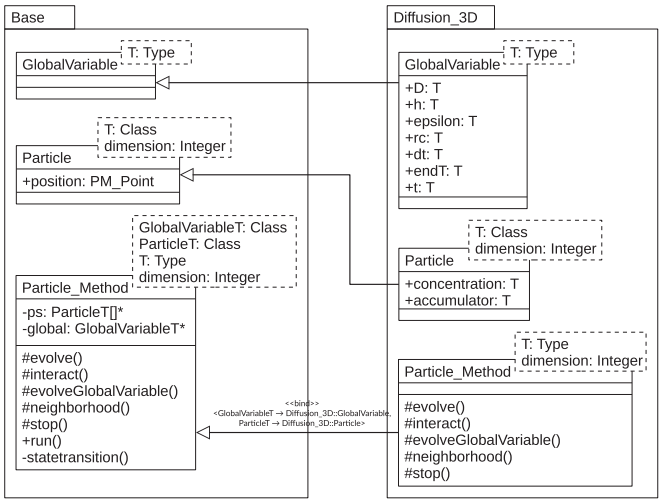
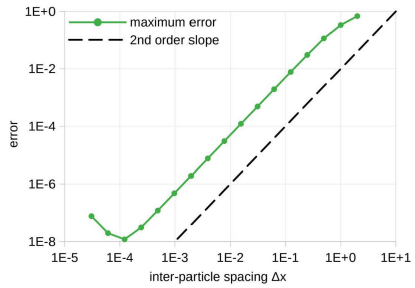


FIGURE 5. UML class diagram of the structure of the software prototype ("Base") and its application to the 3D PSE diffusion simulation example as a client application ("Diffusion_3D").

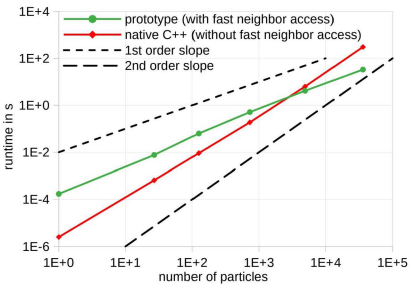
methods and the correctness of the results. The source code is available at: https://git.mpi-cbg.de/mosaic/prototype-particle-methods-defintion-as-an-interface.

The fundamental design of the prototype is illustrated in Fig. 5 as a UML class diagram. 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. 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.

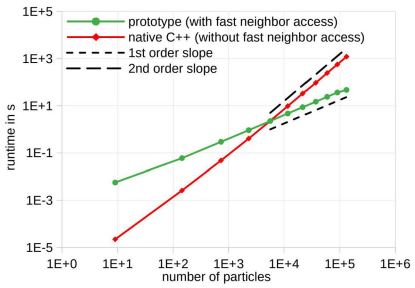
The uses of this prototype are then done in separate namespaces. We chose as an example the three-dimensional diffusion again. The user has to create three new classes that inherit from the three base classes. In the data structure classes, the user adds the necessary properties. In the `Particle_Method` class, the user has to overwrite the functions `evolve()`, `interact()`, `evolveGlobalVariable()`, `neighborhood()`, and `stop()` exactly 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.



(a) Convergence of PSE with second-order kernel (green symbols) compared with analytically expected convergence (black dashed line).



(b) Runtime of the PSE simulation implemented in the proposed framework (green circles) compared to a native C++ implementation (red diamonds) for different numbers of particles.



(c) Runtime of the perfectly elastic collision simulation implemented in the proposed framework (green circles) compared to a native C++ implementation (red diamonds) for different numbers of particles.

FIGURE 6. Validation and runtime measurements of the example PSE application client with the proposed formal software prototype.

We have tested the software to make sure it works as expected. Therefore we compared our prototype to native C++ implementations, did a convergence study, and conducted two run-time evaluations.

For the comparison to native C++, we implemented the perfectly elastic collision (Section SM2.1) and the 3D PSE 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 (Section IV-A) 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 [36].

The run-time evaluations consider the 3D diffusion 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.

V. DISCUSSION

Particle methods are used in a wide range of fields such as plasma physics [19], computational fluid dynamics [8], [11], image processing [1], [7], computer graphics [17], and computational optimization [18], [31]. Formulation of what constitutes a particle method demands a generalized view. Here, we presented a general definition and showed its applicability in developing a common interface for this important class of algorithms.

The proposed definition highlights the algorithmic commonalities across applications, enabling a sharp classification of particle methods. Furthermore, the presented definition unifies all particle methods and allows the formulation of particle methods for non-canonical problems. In addition, this formulation enables theoretical analyses of particle methods and provides a rigorously defined structure for implementing software frameworks for 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. For example, one would frequently restrict a particle method to be order-independent, i.e., to produce results independent of the particle’s indexing order.

Such restrictions are frequently needed to parallelize particle methods on multi-processor computer architectures, such as computer clusters or GPUs. We have shown how such definition restrictions can lead to a parallelizable state transition function, whereas it is sequential in its most general form in the definition.

Even though the presented definition is general and easily applicable to, e.g., SPH, MD, and PSE, a particle method could potentially have a worse time and space complexity than a non-particle algorithm, especially for non-canonical problems. 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. However, we chose the presented formulation for its similarities to practical implementations, as we showcased in our software framework prototype. The prototype interface is almost the same as the particle method algorithm structure. Hence, it takes advantage of the structure of the definition and hides more complex algorithms from the user,e.g., the state transition function and fast neighbor search. Although we investigated the parallelization of the state transition, this is not implemented in our software prototype. The presented parallelization scheme is not implemented in the software. Therefore, its value lies not in its direct applicability in software but in its theoretical significance.

Notwithstanding these limitations, the present definition establishes a rigorous algorithmic class that contrasts the so far loose empirical notion of particle methods. This rigorousness paves the way for future research both in the theoretical and algorithmic foundations of particle methods and the engineering of their software implementation.

Future theoretical work could define standard classes of particle methods by formulating class-specific restrictions to our definition. Such restrictions enable classifying particle methods concerning their parallelizability, algorithmic complexity, and computational power. The presented parallelization scheme for particle methods restricted to pull interactions on shared memory systems requires minor constraints. Hence, we expect that the presented proof for this scheme will be the foundation of proofs for the parallelizability of push(-pull) interaction schemes for shared and distributed memory. It seems intuitive that the presented definition of particle methods is Turing-powerful since one could use a single particle to implement a universal Turing machine in the evolve method. However, this trivial reduction offers no insight into the algorithmic structure and computational power. Studying the computational power of certain classes of particle methods could provide exciting insights into what is possible with different amounts of computational resources. Other possible directions of theoretical research include the derivation of complexity bounds for certain classes of particle methods.

On the engineering side, future work can leverage the presented definition to better structure software frameworks for particle methods, such as the PPM Library [35], OpenFPM [20], POOMA [33], or FDPS [21]. This would render them more accessible and maintainable, as the formal definition provides a common vocabulary. The present definition also enables the classification and comparison of software frameworks concerning their expressiveness, coverage of the definition, or optimization toward specific classes of particle methods.

Future work could also 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 [22], [23], 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.

VI. CONCLUSION

Mathematical definitions reveal the concepts upon which a method is founded, and they render it possible to rationalize the fundamental characteristics of a method. After defining what constitutes a particle method, we leveraged this knowledge to formalize canonical and non-canonical particle methods algorithms and to design and implement new computer software to simulate various physical systems, from fluid dynamics to elastic collision and diffusion.

The presented formal 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. It also provides practical software implementation guidance and enables comparative evaluation on common grounds. We therefore hope that the present work will generate downstream investigation and studies in branches of science developing or using particle methods.

The source code of the prototype generic software framework is available at: https://git.mpi-cbg.de/mosaic/prototype-particle-methods-defintion-as-an-interface.

ACKNOWLEDGMENT

We thank Udo Hebisch (TU Bergakademie Freiberg, Germany) for discussions and comments on an early version of this manuscript and for proofreading. We thank Georges-Henri Cottet, Jean-Baptiste Keck, Christophe Picard, Aude Maignan, Clément Pernet (all Université Grenoble Alpes, France) for discussions on the application examples. We thank Pietro Incardona (University of Bonn) for his C++ insights. We thank Ulrik Günther, Lennart Schulze, Justina Stark, Edgar Dorausch, and Sophie Thery for proofreading. We thank Michele Marass (MPI-CBG) for his editorial advice.

REFERENCES

[1] Y. Afshar and I. F. Sbalzarini, "A parallel distributed-memory particle method enables acquisition-rate segmentation of large fluorescence microscopy images," *PLoS one*, vol. 11, no. 4, 2016, Art. no. e0152528.
[2] B. J. Alder and T. E. Wainwright, "Molecular dynamics simulation of hard sphere system," *J. Chem. Phys.*, vol. 27 pp. 1208–1218, 1957.
[3] M. Bergdorf, I. F. Sbalzarini, and P. Koumoutsakos, "A Lagrangianparticle method for reaction-diffusion systems on deforming surfaces," *J. Math. Biol.*, vol. 61, pp. 649–663, 2010.
[4] G. C. Bourantas, B. L. Cheeseman, R. Ramaswamy, and I. F. Sbalzarini, "Using DC PSE operator discretization in Eulerian meshless collocation methods improves their robustness in complex geometries," *Comput. Fluids*, vol. 136, pp. 285–300, 2016.
[5] P. G. Bradford, G. J. E. Rawlins, and G. E. Shannon, "Efficient matrix chain ordering in polylog time," *SIAM J. Comput.*, vol. 27, no. 2, pp. 466–490, 1998.
[6] D.-G. Caprace, G. Winckelmans, and P. Chatelain, "An immersed lifting and dragging line model for the vortex particle-mesh method," *Theor. Comput. Fluid Dyn.*, vol. 34, no. 1, pp. 21–48, 2020.

[7] J. Cardinale, G. Paul, and I. F. Sbalzarini, "Discrete region competition for unknown numbers of connected regions," *IEEE Trans. Image Process.*, vol. 21, no. 8, pp. 3531–3545, Aug. 2012.
[8] A. J. Chorin, "Numerical study of slightly viscous flow," *J. fluid Mechanics*, vol. 57, no. 4, pp. 785–796, 1973.
[9] G.-H. Cottet, "Two dimensional incompressible fluid flow with singular initial data," *Math. Topics Fluid Mechanics*, vol. 1, pp. 32–49, 2020.
[10] G.-H. Cottet, J.-M. Etancelin, F. Pérignon, and C. Picard, "High order semi-lagrangian particle methods for transport equations: Numerical analysis and implementation issues," *ESAIM: Math. Model. Numer. Anal.*, vol. 48, no. 4, pp. 1029–1060, 2014.
[11] G. H. Cottet and S. Mas-Gallic, "A particle method to solve the Navier-stokes system," *Numer. Math.*, vol. 57, pp. 805–827, 1990.
[12] P. Degond and S. Mas-Gallic, "The weighted particle method for convection-diffusion equations. Part 1: The case of an isotropic viscosity," *Math. Comput.*, vol. 53, no. 188, pp. 485–507, 1989.
[13] P. Degond and S. Mas-Gallic, "The weighted particle method for convection-diffusion equations. Part 2: The anisotropic case," *Math. Comput.*, vol. 53, no. 188, pp. 509–525, 1989.
[14] J. D. Eldredge, A. Leonard, and T. Colonius, "A general deterministic treatment of derivatives in particle methods," *J. Comput. Phys.*, vol. 180, pp. 686–709, 2002.
[15] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "1 potassco: The potsdam answer set solving collection," *AI Commun.*, vol. 24 pp. 107–124, 2011.
[16] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics - theory and application to non-spherical stars," *Roy. Astronomical Soc., Montly Notices*, vol. 181, pp. 375–378, 1977.
[17] M. Gross and H. Pfister, *Point-Based Graphics*. Amsterdam, The Netherlands: Elsevier, May 2011.
[18] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. IEEE Conf. Evol. Comput.*, 1996, pp. 312–317.
[19] R. W. Hockney, "Computer experiment of anomalous diffusion," *Phys. Fluids*, vol. 9, no. 9, pp. 1826–1835, 1966.
[20] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, and I. F. Sbalzarini, "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers," *Comput. Phys. Commun.*, vol. 241, pp. 155–177, 2019.
[21] M. Iwasawa, A. Tanikawa, N. Hosono, K. Nitadori, T. Muranushi, and J. Makino, "Implementation and performance of FDPS: A framework for developing parallel particle simulation codes," *Pub. Astronomical Soc. Jpn.*, vol. 68, no. 4, 2016, Art. no. 54.
[22] S. Karol, T. Nett, J. Castrillon, and I. F. Sbalzarini, "A domain-specific language and editor for parallel particle methods," *ACM Trans. Math. Softw.*, vol. 44, no. 3, 2018, Art. no. 34.
[23] N. Khouzami et al., "The OpenPME problem solving environment for numerical simulations," in *Proc. Int. Conf. Comput. Sci.*, 2021, pp. 614–627.
[24] D. Kozen, *Automata and Computability. Undergraduate Texts in Computer Science*. New York, NY, USA: Springer, 2012.
[25] C. Lefévre, C. Béatrix, I. Stéphan, and L. Garcia, "Asperix, a first-order forward chaining approach for answer set computing," *Theory Pract. Log. Prog.*, vol. 17, no. 3, pp. 266–310, 2017.
[26] J. E. Lennard-Jones, "Cohesion," *Proc. Phys. Soc.*, vol. 43, no. 5, pp. 461–482, Sep. 1931.
[27] W. K. Liu, S. Jun, and Y. F. Zhang, "Reproducing kernel particle methods," *Int. J. Numer. Methods Fluids*, vol. 20, pp. 1081–1106, 1995.
[28] P. A. Lopez et al., "Microscopic traffic simulation using sumo," in *Proc. IEEE 21st Intell. Transp. Syst. Conf.*, 2018, pp. 2575–2582.
[29] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *Astronomical J.*, vol. 82, pp. 1013–1024, Dec. 1977.
[30] J. J. Monaghan, "Smoothed particle hydrodynamics," *Rep. Prog. Phys.*, vol. 68, pp. 1703–1759, 2005.
[31] C. L. Müller and I. F. Sbalzarini, "Gaußian Adaptation revisited – An entropic view on covariance matrix adaptation," in *Proc. EvoStar*, 2010, vol. 6024, pp. 432–441.
[32] S. Reboux, B. Schrader, and I. F. Sbalzarini, "A self-organizing Lagrangianparticle method for adaptive-resolution advection–diffusion simulations," *J. Comput. Phys.*, vol. 231, pp. 3623–3646, 2012.
[33] J. Reynders et al., "POOMA: A framework for scientific simulation on parallel architectures," in *Proc. 1st Int. Workshop High-Level Program. Models Supportive Environ.*, A. Bode, M. Gerndt, R. Hackenberg, and H. Hellwagner, Eds., 1996, pp. 41–49.
[34] I. F. Sbalzarini, A. Mezzacasa, A. Helenius, and P. Koumoutsakos, "Effects of organelle shape on fluorescence recovery after photobleaching," *Biophysical J.*, vol. 89, no. 3, pp. 1482–1492, 2005.
[35] I. F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos, "PPM–A highly efficient parallel particle-mesh library for the simulation of continuum systems," *J. Comput. Phys.*, vol. 215, no. 2, pp. 566–588, 2006.
[36] B. Schrader, S. Reboux, and I. F. Sbalzarini, "Discretization correction of general integral PSE operators in particle methods," *J. Comput. Phys.*, vol. 229, pp. 4159–4182, 2010.
[37] L. Verlet, "Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules," *Phys. Rev.*, vol. 159, no. 1, pp. 98–103, 1967.
[38] J. H. Walther and I. F. Sbalzarini, "Large-scale parallel discrete element simulations of granular flow," *Eng. Comput.*, vol. 26, no. 6, pp. 688–697, 2009.

JOHANNES PAHLKE is currently a Mathematician and Computer Scientist with the graduation degree in applied mathematics from TU Bergakademie Freiberg, Freiberg, Germany, majoring in computer science. At the end of his studies, he started work as a development Engineer with aSpect Systems GmbH, Dresden, Germany. In October 2016, he began his doctoral studies with the Faculty of Computer Science of Technische Universität Dresden (TU Dresden), joining the DFG Research Training Group RoSI (Role-Based Software Infrastructures) under the supervision of Prof. Ivo F. Sbalzarini. He also joined the International Max Planck Research School, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden. In 2019, he had a research stay with the Laboratoire Jean Kuntzmann of the Université Grenoble Alpes, Grenoble, France, with Prof. Georges-Henri Cottet. His research interests include the theoretical foundations and commonalities of algorithms, specifically particle methods.

IVO F. SBALZARINI received the graduation degree (Hons.) in mechanical engineering from ETH Zürich, Zürich, Switzerland, with majors in computational fluid dynamics and control theory, and the Doctorate (Dr. sc. techn.) degree (Hons.) in computer science from ETH Zürich under the supervision of Prof. Petros Koumoutsakos. In 2006, he became an Assistant Professor of computational science with the Department of Computer Science, ETH Zürich. In 2012, he became one of the founding Members of the Center for Systems Biology Dresden (CSBD), and in 2014, the Chair of Scientific Computing for Systems Biology, Faculty of Computer Science, Technische Universität (TU) Dresden, Germany. Since 2021, he has been the Dean of the Faculty of Computer Science. In addition to being a Professor of computer science with TU Dresden, he is a Director of the CSBD and a Senior Research Group Leader with the Max Planck Institute of Molecular Cell Biology and Genetics, Dresden. He is also an Area Lead for Applied Data Science and AI with the Federal Center for Scalable Data Analytics and Artificial Intelligence and the Research Avenue Leader for Scientific Computing and Systems Microscopy with the DFG Cluster of Excellence Physics of Life. His research interests include particle methods, numerical algorithms, data-driven modeling, and parallel computing, all with applications in multi-scale problems of biological systems. He was the recipient of the Willi Studer Award from ETH Zürich for the graduation degree and Chorafas Award from the Weizmann Institute of Science, Israel, for the Doctoral degree.