The Journal of Chemical Physics

RESEARCH ARTICLE | JULY 05 2023

MATILDA.FT: A mesoscale simulation package for inhomogeneous soft matter FREE

Zuzanna M. Jedlinska ; Christian Tabedzki ; Colin Gillespie ; Nathaniel Hess; Anita Yang ; Robert A. Riggleman

J. Chem. Phys. 159, 014108 (2023)
https://doi.org/10.1063/5.0145006

The Journal of Chemical Physics ARTICLE pubs.aip.org/aip/jcp

MATILDA.FT: A mesoscale simulation package for inhomogeneous soft matter

Cite as: J. Chem. Phys. 159, 014108 (2023); doi: 10.1063/5.0145006
Submitted: 2 February 2023 • Accepted: 12 June 2023 •
Published Online: 5 July 2023

Zuzanna M. Jedlinska,[1] Christian Tabedzki,[2] Colin Gillespie,[2] Nathaniel Hess,[2] Anita Yang,[2] and Robert A. Riggleman[2,a)]

AFFILIATIONS

[1] Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, Pennsylvania 19104, USA
[2] Department of Chemical and Biomolecular Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104, USA

[a)] **Author to whom correspondence should be addressed:** rrig@seas.upenn.edu

ABSTRACT

In this paper, we announce the public release of a massively parallel, graphics processing unit (GPU)-accelerated software, which is the first to combine both coarse-grained particle simulations and field-theoretic simulations in one simulation package. MATILDA.FT (Mesoscale, Accelerated, Theoretically Informed, Langevin, Dissipative particle dynamics, and Field Theory) was designed from the ground-up to run on CUDA-enabled GPUs with Thrust library acceleration, enabling it to harness the possibility of massive parallelism to efficiently simulate systems on a mesoscopic scale. It has been used to model a variety of systems, from polymer solutions and nanoparticle-polymer interfaces to coarse-grained peptide models and liquid crystals. MATILDA.FT is written in CUDA/C++ and is object oriented, making its source-code easy to understand and extend. Here, we present an overview of the currently available features, and the logic of parallel algorithms and methods. We provide the necessary theoretical background and present examples of systems simulated using MATILDA.FT as the simulation engine. The source code, along with the documentation, additional tools, and examples, can be found on the GitHub MATILDA.FT repository.

I. INTRODUCTION

Polymers are a ubiquitous type of material that is important in both biological and industrial settings. Polymers are an umbrella term, gathering macromolecules composed of smaller, repeating monomers. In industrial settings, polymers are extensively used in the tire industry, the production of flexible composite materials, and as durable adhesives. In addition, they have been exploited in more precise applications, such as drug delivery[1] and the design of artificial catalysis centers.[2] This is possible due to the propensity of polymers to self-assemble into higher-order structures and their ability to undergo phase separation in solution. Controlled phase separation has been exploited to create nano-capsules with well-defined pore sizes, by first inducing phase separation in the capsule shell and then flushing-out one of the components.[3] Similar approaches using non-solvents to induce phase separation in a polymer solution are common methods to produce polymer membranes.[4,5] The design of these materials requires precise knowledge of the thermodynamics and microstructure of polymer materials under a variety of conditions, where molecular modeling can play an important role. Similarly, the structure and phase behavior of polymers in a biological context has recently been shown to be important for many cellular functions through the formation of membraneless organelles[6,7] where coarse-grained models can potentially provide insight.

Various open-source Molecular Dynamics (MD) codes have been released, which are capable of simulating polymeric species on an atomistic or coarse-grained level. Some notable examples include the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS),[8] NAMD,[9] and GROMACS.[10] LAMMPS can perform all-atom simulations on polymer chains using available force fields. It is also equipped with biologically oriented force fields, which enable coarse grained simulations of biomolecules. In addition, the user can define their own coarse-grained polymer model and expand it to include the required potentials. On the other hand, both GROMACS and NAMD have been specifically designed to model biological molecules, such as proteins and nucleic acids on a fully atomistic level. Polymer simulations are challenging, in

general, due to the wide range of length- and time-scales required for accurate simulation. As a result, highly coarse-grained models with relatively high particle densities are frequently used to describe phenomena on larger length scales.

In many soft matter fields, particularly those involving the design of materials using polymers, polymer field theory and related techniques have played a crucial role in the design of new materials and the interpretation of experimental results.[11–16] Polymer field theories are developed by beginning with a description of the system in terms of coarse-grained potentials, such as chains obeying Gaussian statistics, Flory contact repulsions governed by a χ parameter, and partial charges on the various species. One writes down the partition function for this particle model using one of a variety of transformation techniques,[11,13,17] decouples the particle interactions, and transforms the model to one where one molecule of each type interacts with chemical potential fields generated by the various interaction potentials. The field-theoretic approach is attractive because it enables a variety of analytic analyses, such as the mean-field approximation, which gives rise to self-consistent field theory (SCFT) or a variety of loop expansions. The particle-to-field transformation is formally exact, and there are examples in the literature showing quantitative agreement between the particle and field versions of the model.[18] However, open-source codes that efficiently perform field-theoretic simulations (FTS) are scarce. The PSCF code[19] and its recent graphics processing unit (GPU) extension[20] are notable exceptions, but this software is primarily focused on unit-cell calculations for ordered phases under the mean-field approximation. While this is a broad and important class of problems for which FTS is used, phenomena driven by fluctuation effects, such as complex coacervation and large-cell simulations, are beyond the scope of the current version of PSCF.

In more recent years, several methods have been developed to sample the original particle model efficiently,[21–25] which generally fall under the umbrella of theoretically informed coarse-grained (TICG) models. In these methods, the underlying particle coordinates are retained, and the particles are mapped to density fields to efficiently calculate the non-bonded forces and energies. At this highly coarse-grained level, the typical coarse-grained bead density ρ_0 can be relatively high ($\rho_0 \in [3, 50]$ dimensionless units), and calculating the local pair-wise interactions can be costly using traditional neighbor lists. By mapping to a density field, a significant speedup can be achieved because converged results can be obtained with a grid density that is less than the bead density. The need for multiple density fields makes the TICG class of methods difficult to implement on top of many particle-based simulation codes, where the only density field is typically the charge density used in computing electrostatic forces with particle-to-mesh Ewald techniques.[26] While there is an open-source code for performing the closely related single-chain in mean-field (SCMF) method[27] that has been optimized for parallel architectures such as GPUs, to the best of our knowledge, it is primarily designed to study uncharged polymeric species.

In this work, we present the first version of our code for Mesoscale, Accelerated, Theoretically Informed, Langevin, Dissipative particle dynamics (DPD), and Field Theory, MATILDA.FT. MATILDA.FT is written from the ground-up intended to be run on GPUs, and the bulk of the code is written using the CUDA programming language. MATILDA.FT is capable of modeling both systems consisting of a few molecules as well as those containing millions of particles. Its strength lies specifically in being able to efficiently simulate polymeric and other soft materials (e.g., liquid crystalline systems) on the mesoscale. On this scale, the coarse-grained interactions are typically "soft" (finite at overlap) and the particle density high; in this limit, it becomes more efficient to evaluate the non-bonded interactions using density fields. These large-scale molecular assemblies of interest can correspond to block polymers of arbitrary architecture, biomolecular coacervates in explicit solvent, artificially synthesized ionomers, side-chain liquid crystalline polymers, or polymer-infiltrated nanoparticle packings.

The outline of this paper is as follows: In Sec. II, we describe the structure of the models being used in molecular dynamics and field-theoretical simulations. Next, in Sec. III, we outline the main features of the code and available functionalities at a higher level, before taking a more detailed look at the code structure in Sec. IV. Next, in Sec. V, we show results for selected example systems. We end with the planned developments in Sec. VI and conclude with Sec. VII.

II. STRUCTURE OF THEORETICALLY INFORMED COARSE-GRAINED AND FIELD-THEORETIC MODELS

A. Particle-based models

In this section, we provide the necessary theoretical background to understand the models used in MATILDA.FT and the logic of the simulation workflow. The starting ingredients for all of the modeling handled by MATILDA.FT are highly coarse-grained models for soft-matter systems. For simplicity, we will describe the basic structure in terms of a simple A–B Gaussian chain diblock copolymer melt, although the generalization to other systems will become apparent below. For a polymer melt with n polymer chains each containing $N_A + N_B = N$ monomers, the microscopic polymer densities are

$$\hat{\rho}_K(\mathbf{r}) = \sum_j^n \sum_s^{N_K} \delta(\mathbf{r} - \mathbf{r}_{j,s}), \tag{1}$$

where K is either species A or B, and $\mathbf{r}_{j,s}$ is the position of the sth bead on the jth chain. The monomers on each chain are typically connected via harmonic bonds,

$$\beta U_0 = \sum_j^n \sum_s^{N-1} \frac{3}{2b^2} |\mathbf{r}_{j,s} - \mathbf{r}_{j,s+1}|^2, \tag{2}$$

where b is the statistical segment size, and we have assumed equal b for species A and B. The Flory repulsion is written in one of two equivalent forms,[18,28,29] depending on whether the model is implemented as a field- or particle-based model. In the particle-based approaches, we make the potential non-local as

$$\beta U_1 = \frac{\chi}{\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \hat{\rho}_A(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \hat{\rho}_B(\mathbf{r}'), \tag{3}$$

where u_G is a unit Gaussian potential, $u_G(r) = (2\pi\sigma^2)^{-\mathbb{D}/2} e^{-r^2/2\sigma^2}$, $\mathbb{D}$ is the dimensionality of the system, and σ controls the range of the interactions. The standard Flory–Huggins model is recovered in

the limit $\sigma \to 0$. The final potential penalizes deviations of the local density from the average[30] $\rho_0 = nN/V$,

$$\beta U_2 = \frac{\kappa}{2\rho_0} \int d\mathbf{r} \int d\mathbf{r}' [\hat{\rho}_+(\mathbf{r}) - \rho_0] u_G(\mathbf{r} - \mathbf{r}') [\hat{\rho}_+(\mathbf{r}') - \rho_0]. \quad (4)$$

With these ingredients in hand, we can write the partition function as

$$\mathcal{Z} = z_0 \int d\mathbf{r}^{nN} e^{-\beta U}, \quad (5)$$

where z_0 is a prefactor that contains all of the self-energy terms, factors accounting for molecular indistinguishability, and the thermal de Broglie wavelengths. In equilibrium particle-based simulations, one is primarily interested in calculating averages of quantities that can be expressed as functions of the particle coordinates, $M(\mathbf{r}^{nN})$, as

$$\langle M \rangle = \frac{1}{\mathcal{Z}} \int d\mathbf{r}^{nN} M(\mathbf{r}^{nN}) e^{-\beta U}, \quad (6)$$

and expressions for the usual thermodynamic quantities of interest, such as the average density, energy, and pressure, can be readily obtained from expressions commonly used in molecular dynamics simulations.[31]

In a particle-based Theoretically Informed Langevin Dynamics (TILD) simulation, the total force on a monomer is computed from the potentials described above. Bonded potentials, such as Eq. (2), have their forces computed based on particle coordinates, while non-bonded potentials are computed from density fields. Briefly, a potential, such as the Flory potential [Eq. (3)], gives rise to a force on species A of the form

$$\mathbf{f}_A(\mathbf{r}) = -\frac{\chi}{\rho_0} \int d\mathbf{r}' \nabla u_G(\mathbf{r} - \mathbf{r}') \hat{\rho}_B(\mathbf{r}'). \quad (7)$$

After mapping the particles to the density field to define $\hat{\rho}_B(\mathbf{r})$, this force, which takes the form of a convolution integral, is then evaluated efficiently in Fourier-space and subsequently mapped back to the particles. The total force is accumulated on the particles, their positions are updated using one of several numerical integration schemes for a Langevin equation (detailed below), and the density fields are regenerated.

B. Field-based models

For field-theoretic approaches, we use a local potential but render the densities non-local by distributing the point particles over a Gaussian distribution $h(r) = (2\pi a^2)^{-\mathbb{D}/2} e^{-r^2/2a^2}$, and the Gaussian-distributed particle density is given by

$$\check{\rho}_K(\mathbf{r}) = \int d\mathbf{r}' h(\mathbf{r} - \mathbf{r}') \hat{\rho}_K(\mathbf{r}') = [h * \hat{\rho}_K](\mathbf{r}), \quad (8)$$

where the final equality introduces our short-hand notation for a convolution integral. For the choice $\sigma^2 = 2a^2$, we can exactly re-write[18,28,29] the non-bonded potentials in Eqs. (3) and (4) as

$$\beta U_1 = \frac{\chi}{\rho_0} \int d\mathbf{r} \check{\rho}_A(\mathbf{r}) \check{\rho}_B(\mathbf{r}), \quad (9)$$

and

$$\beta U_2 = \frac{\kappa}{2\rho_0} \int d\mathbf{r} [\check{\rho}_+(\mathbf{r}) - \rho_0]^2. \quad (10)$$

Using known Gaussian functional integrals,[11,13] one can then exactly transform the particle-partition function in Eq. (5) to a field-theoretic one of the form,

$$Z = z_1 \int \mathcal{D}\{w\} e^{-\mathcal{H}[\{w\}]}, \quad (11)$$

where z_1 contains the constants from z_0 as well as the normalizing factors from the Gaussian functional integrals, $\{w\} = \{w_+, w_{AB}^{(+)}, w_{AB}^{(-)}\}$ is the set of chemical potential fields, and $\mathcal{H}$ is the effective Hamiltonian governing the weights of the microstates. For the diblock copolymer model considered here, $\mathcal{H}$ takes the form,

$$\begin{aligned} \mathcal{H} = &\frac{C}{\chi N_r} \int d\mathbf{r} \left([w_{AB}^{(+)}(\mathbf{r})]^2 + [w_{AB}^{(-)}(\mathbf{r})]^2 \right) \\ &+ \frac{C}{2\kappa N_r} \int d\mathbf{r} [w_+(\mathbf{r})]^2 - iC \int d\mathbf{r}\, w_+(\mathbf{r}) \\ &- n_D \log Q_D[\mu_A, \mu_B], \end{aligned} \quad (12)$$

where the first line contains the potential fields that arise due to the Flory interaction,[18] the second line contains the terms that arise from the Helfand potential, and the final line contains the excess chemical potential of the polymers in a given field. The potential fields μ_A and μ_B experienced by monomers A and B computed using

$$\begin{aligned} \mu_{A,c}(\mathbf{r}) &= \left\{ i(w_+ + w_{AB}^{(+)}) - w_{AB}^{(-)} \right\}(\mathbf{r})/N_r, \\ \mu_{B,c}(\mathbf{r}) &= \left\{ i(w_+ + w_{AB}^{(+)}) + w_{AB}^{(-)} \right\}(\mathbf{r})/N_r, \end{aligned} \quad (13)$$

with the smeared potential fields appearing in Eq. (12) calculated as $\mu_K(\mathbf{r}) = [h * \mu_{K,c}](\mathbf{r})$.

While the particle implementation can report qualitatively realistic dynamic quantities, the FT implementation is strictly interested in equilibrium quantities. Equilibrium averages are typically expressed as functionals of the potential fields and calculated as

$$\langle M \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}\{w\} M[\{w\}] e^{-\mathcal{H}}. \quad (14)$$

Since $\mathcal{H}$ is typically complex-valued, sampling the integral over the field configurations is non-trivial; this is typically accomplished through the mean-field approximation, leading to SCFT through complex Langevin (CL) sampling[11,32,33] or Monte Carlo sampling.[34]

To update the chemical potential fields in either a CL or an SCFT calculation, the effective "forces" on the fields must be obtained as functional derivatives of $\mathcal{H}$. These can be obtained through explicit differentiation,

$$\mathbf{F}_w(\mathbf{r}) = -\frac{\delta \mathcal{H}}{\delta w(\mathbf{r})}, \quad (15)$$

where $w(\mathbf{r})$ is one of the three fields $w_+(\mathbf{r}), w_{AB}^{(+)}$, or $w_{AB}^{(-)}$. In a CL simulation, the fields are sampled using an overdamped Langevin equation,

$$\frac{\partial w}{\partial t} = \lambda_w \mathbf{F}_w(\mathbf{r}) + \eta(\mathbf{r}, t), \quad (16)$$

where $\eta(t)$ is a stochastic noise term chosen to satisfy the fluctuation–dissipation theorem.[11,35] An algorithm that drives the

system to an SCFT solution is easily obtained from Eq. (16) by simply setting the noise term to zero.

III. FEATURE OVERVIEW

In this section, we provide a brief overview of the features available in MATILDA.FT, which are summarized in Fig. 1. They are later described in more detail in the following Sec. IV.

A. Particle-based (TILD) features

In the current implementation, particle-based TILD simulations are performed in the NVT ensemble, in a fully-periodic orthogonal box, either in two or in three dimensions. Although MATILDA.FT can perform simulations of free particles, it has been designed specifically to efficiently model systems of polymer melts and solutions. Polymers are modeled as discrete Gaussian or worm-like chains with monomers that are connected through harmonic springs, and the density of each monomer is spread around its center through convolution with a unit Gaussian. The strength of the repulsive interactions between chemically distinct species is mediated through the Flory–Huggins χ parameter. Monomers can be either neutral or charged. If they carry a net charge, then in addition to the repulsive potential, they also interact through Coulombic electrostatic forces. Regardless of their net charge, the monomers can be made polarizable through the use of (classical) Drude Oscillators as detailed in Sec. V C.

The user interacts with the code through input scripts written in a plain-text format. Before the simulation is started, the entire script is read, and appropriate variables and data structures are initialized. The maximum number of time steps and the time step size are parameters specified in the input script. Then, the system can undergo an optional equilibration period before subsequently entering the production run stage. For a TILD simulation, two files need to be provided. The first one is the main input file, providing information about simulation dimensionality, box size, density grid spacing, and interaction potentials between particle types. It also defines the particle groups and assigns integrators and forces to act on them during each time step. The second file contains information about particle coordinates, types, molecules they belong to, and, optionally, their charge. Currently, this file can be provided in the format consistent with the LAMMPS data file, in either *angle* or *charge* atom style. To allow the use of data generated by other codes, the initial configuration can also be read from a GSD-format file developed by the Glotzer Lab.[36]

B. Field theory features

Currently, FTSs are limited to mean-field calculations as in self-consistent field theory (SCFT) with linear, discrete Gaussian chain models. The molecules can be of arbitrary blockiness with an arbitrary number of components, and the potentials implemented include the Flory contact repulsion and the Helfand weak compressibility. More details about the interactions between the species are provided in Sec. II. As detailed below, the key elements of the FTS implementation are three classes: **Potentials**, which govern the non-bonded interactions and act on **Species**. The **Species** class stores the total density of each chemical component and is populated by individual **Molecule** classes. For example, an A-homopolymer/B-homopolymer/AB-diblock copolymer blend would have two species (A and B) and three molecules. A single text input file is used to specify the parameters of a FTS.

As the development of the FTS features of the code began well after the development of the particle-based TILD methods, the feature set and breadth of capabilities are comparatively limited. Furthermore, as can be observed by comparing Figs. 2 and 3, the class structure is also rather distinct. As discussed below, planned future development will work to merge the two branches to a more common class structure and the addition of numerous extensions.

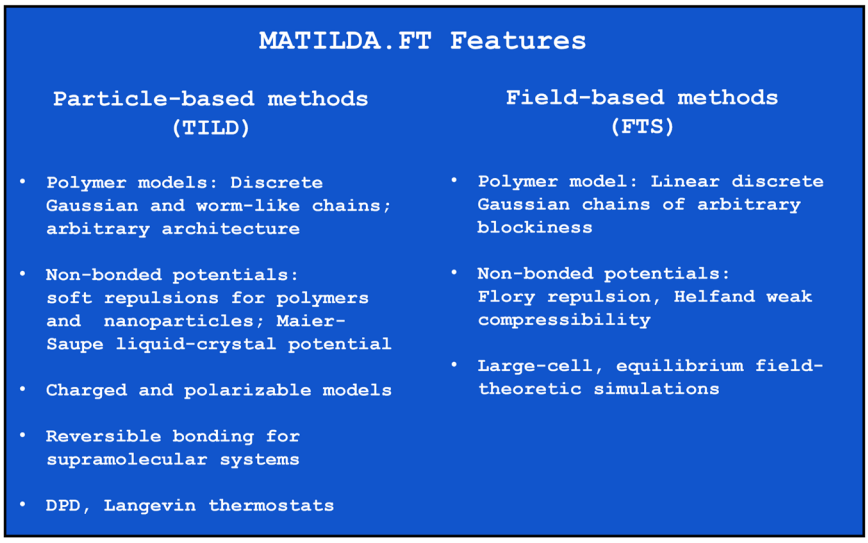

FIG. 1. Summary of the features of the two styles of simulations capable by MATILDA.FT.

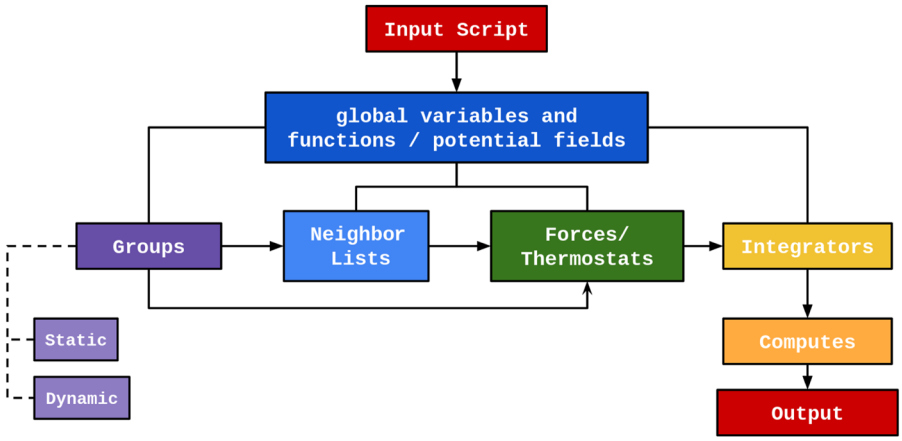

FIG. 2. Schematic outline of code structure of the TILD branch.

IV. CODE DETAILS AND CLASS STRUCTURE

The unique feature, which sets MATILDA.FT apart from other popular MD codes, is that the code is intended for highly coarse-grained models where the non-bonded forces can be evaluated using density fields and not summing over neighboring pairs of particles. It uses a dedicated CUDA/C++ programming language in order to fully harness parallel capabilities. Its model of parallelization differs from the conventional central processing unit (CPU) domain decomposition. In contrast to the CPU, where groups of particles are assigned to different processors based on their spatial arrangement, GPU parallelization occurs on the particle or individual grid location level, where each thread is responsible for processing instructions for the selected particle/grid point. It is simply handled by assigning a separate thread to individual particles and filtering the thread IDs.

MATILDA.FT also makes extensive use of the Thrust library, which is an extension of the C++ Standard Template Library (STL) to work with GPUs.[37] The Thrust Library provides dedicated storage containers (equivalent to STL vectors in C++), which enable easier host-device communication by avoiding the requirement for explicit *cudaMemcpy* calls. The Thrust Library also makes available dedicated parallel algorithms to operate on these containers and achieve better performance. In addition, by avoiding complicated host-device memory transfer syntax, the use of thrust makes it easy for those who do not have much GPU-programming experience to understand and expand the MATILDA.FT source code.

A. Particle-based (TILD) implementation

The code takes advantage of the C++ object oriented programming approach. It is divided into classes, which interact with each other and exchange data as needed. Each class is responsible for handling a particular functionality. The base class serves as an interface used to interact with other parts of the code. Then specialized sub-classes are derived from the base class to provide specific functionality. This organization makes extending the code to include new functionalities a relatively easy and straightforward process, with simple integration of the new components into the existing code. For example, the neighborList class is responsible for constructing and storing the neighbor list for the selected group of particles. This neighbor list is then used by the additional forces (created as a subclass of the ExtraForce class) to accelerate the operations performed

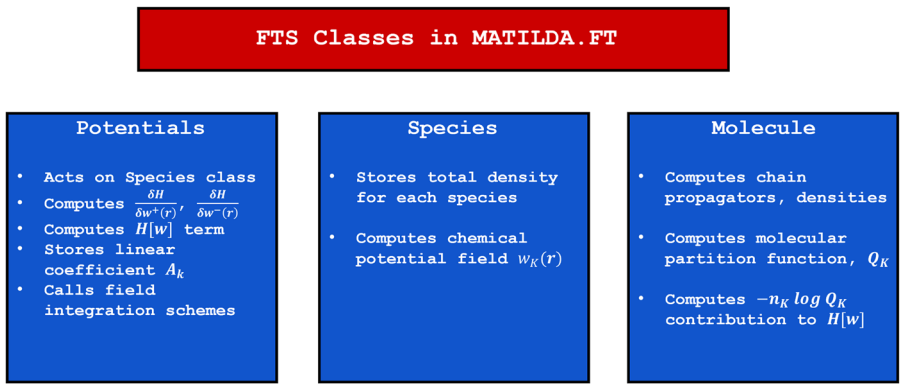

FIG. 3. Basic actions and roles of the FTS classes in MATILDA.FT.

on this group. Depending on the nature of the additional force, specialized neighbor lists can be used in order to further accelerate the performance. A diagram of class code organization is shown in Fig. 2. Below, we provide a brief description of selected classes. The outline of the code structure along with a detailed description of all class functionalities and options are provided in the documentation.

1. Input script

The first step of the simulation is selecting the method to be used, either TILD or FT. This is passed as a command line argument when the program is called. Using `./MATILDA.FT -particles` will run the TILD simulation, whereas the `-ft` option will initialize an FT run. Additional command line arguments, like the name of the input script to be used, are described in the documentation. Two files are required for a TILD simulation. The main input file is responsible for setting up simulation parameters, such as the dimensionality of the system, size of the simulation box, grid density, time step size, and the number of time steps to perform. The input script also defines the interaction potentials between selected particle types, along with the parameters that determine the details of electrostatic interactions. The same file also contains information about the particle groups, neighbor list, and any additional forces to be applied.

The second (data) file provides the initial positions of the particles, their types, and the molecules they belong to. This data file also initializes the static bonds and angles used in the simulation. Currently, the initial atom configuration can be read either from the LAMMPS data file (in angle or charge style) or from a GSD file. Static bonds are established at the beginning of the simulation and cannot be broken. Dynamic bonds, on the other hand, can be created and destroyed over the course of the simulation. Like regular bonds, they are assigned an equilibrium length and a corresponding constant. However, they are also assigned formation/breaking energy, which is responsible for the reaction constant between the bonded and nonbonded states. Dynamic bonds can be used to simulate the effects of polymerization, network formation, or supramolecular assembly. To accelerate performance, dynamic bonds are coupled with a dedicated neighbor-list.

2. Global variables space

The global variable space holds the main data structures used in the simulation. It stores the global arrays containing particle types, positions, forces acting upon them, velocities, and static bonds, arranged according to the particle ID. Reading of the input script is also handled at this level. Before the beginning of the simulation, these arrays are initialized and then periodically updated by other classes during the time-stepping process. In the future release, these structures will be placed in a separate Box class to closely resemble the organization of the FT branch of the code, which is described below.

3. Group class

The **Group** base class provides data structures that store indices of the member particles. It sets device-specific variables (BLOCK and GRID sizes) that are used in kernel calls dispatch on the group particles. All forces and neighbor lists, in MATILDA.FT, operate on specific groups. Pointers to each group object are stored in a globally accessible vector array. Each group is assigned a unique name, which is used to pass its pointer to their classes. Groups can be static or dynamic. Static groups are initialized at the beginning of the simulation, and their content remains unchanged over the simulation course. Dynamic groups, on the other hand, periodically check and update their members based on the specified membership criterion. A special group, named "*all*," is initialized by default at the beginning of the simulation and contains all particles in the simulation box. Currently, two static group types are available—grouping by particle **type** or by its global **id**. Type-based groups collect all the particles with the same type (as specified in the input.data). Id-based groups require the user to provide an external plain text file, which contains the indices of the particles to be included in the group. Currently available dynamic group style, "regions," allows the user to define a separate region in space (along all or only specific axis). Particles found within that region get assigned to the group.

4. Neighbor list class

We have implemented several neighbor list variants for efficient particle–particle operations. In addition to the standard cell/neighbor list combinations where all particles store all of their neighbors, specialized subclasses are built upon this base to optimize the performance of the associated operations. Currently, two sub-classes are available. The "distance" neighbor list is intended to be used with the DPD force. Its structure is designed such that each particle only stores the indices of its neighbors, which have lower indices than its own. In this way, when the pairwise interactions are calculated, unnecessary "if" statements to perform the calculation for each pair only once are avoided, thus avoiding wasted threads and thread divergence.

A more elaborate neighbor list is coupled with dynamic bonding. The main goal of this structure is to again minimize the amount of atomic operations and thread divergence. An associated group partitions its members into donors and acceptors, whose indices are stored in separate lists. The binning step is performed only for acceptor particles. Subsequently, the grid position of each donor particle is calculated, and each donor is assigned a list of neighborhood acceptors. During the binding step, only donor particles can initialize bond making or breaking and can only couple with acceptor particles. With the pre-calculated neighbor-list, superfluous checks of the particle type and wasted threads are avoided.

5. ExtraForce class

In addition to electrostatic and repulsive interactions, selected groups of particles can be subject to additional user-defined forces. These are specified in the input script using the *extraforce* command. The **ExtraForce** base-class is responsible for assigning the force to the specific group of particles and ensuring that it is applied to this group at the specified time-steps (whether each step or user-defined frequency). In addition, some range-limited forces require a neighbor list to restrict the search space only to the particles present within the specific range. Currently, available forces are

- **Wall**—which enables the particles to be confined within a specific region or to simulate surface interactions. The user can choose from available wall-particle potentials or specify their own form of interaction, by extending the source code.

- **Langevin**—Adds random noise to the selected group of particles. Can be used with Velocity Verlet (VV) integrator to simulate Brownian dynamics.
- **Midpush**—Adds a force to push the selected group of particles toward the center of the box along a specified axis.
- **DPD**—Dissipative Particle Dynamics. This subclass of ExtraForce provides an alternative way to introduce random noise into the simulation and should be used along with the Velocity Verlet integrator. In contrast the Langevin thermostat, however, it is pair-wise additive and conserves local momentum. Thus, it is capable of correctly reproducing the hydrodynamic behavior of the system. Since the force acts over a limited range, a neighbor list needs to be constructed for the particles of interest. The particle-level operations add significantly to the expense of the simulation.
- **Lewis** - These additional forces can be used to introduce dynamic bonds in the simulation. While static bonds are initialized at the beginning of the simulation and remain unchanged, dynamic bonds can be formed and broken according to the specified acceptance criterion. This force requires a specialized neighbor list (*bonding*), which has been designed to optimize the required computations.

More details about the ExtraForce class can be found in the documentation.

6. Compute class

The *Compute* class is responsible for performing on-the fly calculations of the properties of the system. This enables the user to monitor the evolution of the system in real time and also saves time spent on post-processing.

- **Average Structure Factor** $\langle S(k)\rangle$—this compute provides information about the average static structure factor of the particle system. The static structure factor, given by

$$S(k) = \frac{1}{N}\langle \hat{\rho}_k \hat{\rho}_{-k}\rangle, \tag{17}$$

is defined as the correlation function of the system density represented in the Fourier space. The density is given by $\rho(r) = \sum_{i=1}^{N} \delta(r - r_i)$, and in Fourier space, it becomes $\hat{\rho}_k = \sum_{i=0}^{N} e^{ik\cdot r_i}$.[38] It performs the calculation and writes the data to an external file according to user-specified frequency.

7. Integrator class

In order to solve the equations of motion and propagate the particle coordinates in time, numerical integration is required. In MATILDA.FT, three different numerical algorithms are available and are briefly described below:

- **Velocity-Verlet (VV).** Needs to be coupled with an additional thermostat. Available thermostats include Langevin noise or Dissipative Particle Dynamics, which are part of the ExtraForce class.
- **Euler–Maruyama (EM).** Generates the thermal noise internally during the update and serves as the simplest stochastic integration scheme to implement.
- **Grønbech–Jensen and Farago (GJF).** Generates thermal noise internally during the update, and we find that this algorithm allows time steps up to 10× larger than the EM algorithm with no loss of accuracy or stability.

B. FTS implementation

1. Class structure

As the FTS branch was begun more recently, many planned features are still in development. There is also a difference in class organization between the older (TILD) and the newer (FT) branches. The classes that comprise an FT simulation are more tightly integrated, and the scope of the object-oriented organization is larger, as compared to the TILD branch, which still uses global variables. In the near future, we are planning to refactor the TILD branch to also be fully class-based. However, due to the inherent differences between the TILD and FT simulations, it will not be possible for both of the branches to have exactly the same organization.

A field-theoretic simulation lives in an **FTS_Box** class, which contains three key classes: **Potentials**, **Molecules**, and **Species** (see Fig. 3 for a graphical outline of FT branch organization). The **Potentials** class performs all of the functions that are related to the various non-bonded interactions, including updating the potential fields associated with a particular interaction. The densities that show up in the effective forces are taken from the **Species** class, which serves as a container for these densities. **Species** generate the unsmeared chemical potential fields, $\mu_{K,c}(\mathbf{r})$, by looping over the interaction potentials and accumulating the relevant potential fields. Next, the **Molecules** class takes these potential fields, applies any density smearing that may be necessary, and computes both the center and smeared density fields. The *smeared* density fields are then accumulated into the relevant **Species** class. The general flow of the code is summarized in Fig. 4.

2. Field update schemes

Currently, two schemes have been implemented to update potential fields, in order to evolve in time equations of motion such

FTS Simulation Outline

1. **Initialization:**
   - 1.1. Read input file - initialize potential fields
   - 1.2. **Molecule class** - compute density fields
   - 1.3. Accumulate them in the **Species class**
2. **Time Step:**
   - 2.1. Forces calculated by **Potentials class**, using density fields.
   - 2.2. Update fields using a selected scheme
   - 2.3. Zero densities of all **Species**
   - 2.4. Each **Molecule** - compute own density field, then accumulate in **Species** density
   - 2.5. If step < max_steps: go to 2.1, else go to 3.
3. **Write final output and exit**

FIG. 4. Outline of an FTS simulation as implemented in MATILDA.FT. The termination condition could be convergence to within a prescribed tolerance in SCFT or reaching the maximum desired number of time steps in a CL simulation.

as Eq. (16). The straightforward explicit Euler–Maruyama (EM) integration scheme discretizes the equation in time and uses the forces at the current time to estimate the field configurations at a future time as

$$w^{t+\delta t}(\mathbf{r}) = w^t + \delta t \lambda_w F_w^t(\mathbf{r}) + \sqrt{2\delta t \lambda_w}\zeta_t(\mathbf{r}), \tag{18}$$

where δt is the size of the time step and $\zeta_t(\mathbf{r})$ is purely real Gaussian noise with a unit variance that is uncorrelated in both space and time. As mentioned above, simply neglecting the noise term converts this algorithm to one that drives the system to a mean-field solution.

The other algorithm that is implemented is a first-order, semi-implicit (1S) updating scheme that has been shown to allow for time steps significantly larger than allowed by the EM scheme.[28,35,39] In this approach, one derives an approximate expression for the force $F_w^{t,lin}(\mathbf{r})$ that is linear in the potential field. In real-space, these expressions take the form of a convolution,

$$F_w^{t,lin}(\mathbf{r}) = \int d\mathbf{r}' A_w(\mathbf{r}-\mathbf{r}')w(\mathbf{r}'), \tag{19}$$

where $A_w(\mathbf{r})$ is the linear coefficient. As a result of this convolution, the 1S updating scheme is most effectively handled in Fourier space where we have

$$F_w^{t,lin}(\mathbf{k}) = A_w(\mathbf{k})w(\mathbf{k}). \tag{20}$$

To affect the semi-implicit scheme, Eq. (18) is written in Fourier space and modified by subtracting the linear term at $t+\delta t$ and adding it at t giving,

$$w^{t+\delta t}(\mathbf{k}) = \delta t \lambda_w \big[F_w^t(\mathbf{k}) + A_w(\mathbf{k})w^t(\mathbf{k}) - A_w(\mathbf{k})w^{t+\delta t}(\mathbf{k})\big] + w^t(\mathbf{k}) + \sqrt{2\delta t \lambda_w}\zeta_t(\mathbf{k}). \tag{21}$$

We note that $\zeta_t(\mathbf{k})$ is generated as a spatially uncorrelated noise field in real-space that is explicitly Fourier transformed. Equation (21) can be readily solved for the field at $t+\delta t$ giving

$$w^{t+\delta t} = \frac{w^t + \delta t \lambda_w \big[F_w^t + A_w w^t\big] + \sqrt{2\delta t \lambda_w}\zeta_t}{1 + \delta t \lambda_w A_w}, \tag{22}$$

where we have suppressed the wavevector dependence for brevity.

The functional form of the linear coefficients A_w generally contains one or two contributions that have a stabilizing effect on the time integration.[35,39] The first arises from the terms that are quadratic in the fields in $\mathcal{H}$ [e.g., the first two lines in Eq. (12)]; this term is included for every type of interaction potential. During the initialization of an FT simulation, the **Potentials** class adds this relevant term to A_w. The second contributions are the linear approximate of the density operators, which involve convolutions of Debye functions with the potential fields; these contributions are handled by the **Molecules** class during initialization.

3. Molecule types

Currently, the only implemented molecule type is a linear, discrete Gaussian chain with an arbitrary number of blocks. This class handles the calculation of the chain propagators, and during initialization, the code automatically checks whether the molecule is symmetric to avoid calculating the complimentary propagator if possible. The other key step taken in the initialization is to accumulate the relevant Debye-function-like contributions to the linear coefficients associated with each potential. From previous studies,[35,39] not all terms that show up in the precise linear expansion of the force are stabilizing; to that end, we do not include the Debye terms in the $w_{AB}^{(-)}$ field, but they are included in the w_+ and $w_{AB}^{(+)}$ fields.

V. INTERACTION POTENTIALS AND EXAMPLE SYSTEMS

A. Bonded interactions

Bonded interactions are present whenever polymer chains are modeled. Currently, they represent the harmonic springs connecting adjacent monomers of the same molecule. The calculation of the resulting forces has been parallelized to be performed on the GPU, and Newton's third law is *not* used to avoid the use of atomic operations. The contribution of the bonded interactions is calculated individually for each particle by assigning it to a separate thread.

Two common angle potentials are also implemented in MATILDA.FT. To enable simulations of discrete worm-like chains, we have the cosine form,

$$u_{wlc}(\theta_{ijk}) = \lambda\big[1 + \cos(\theta_{ijk})\big], \tag{23}$$

where λ controls the stiffness of the potential and θ_{ijk} is the *inside* angle between particles i, j, and k. The second potential implements harmonic angles as

$$u_h(\theta_{ijk}) = k_\theta\big(\theta_{ijk} - \theta_0\big)^2, \tag{24}$$

with spring constant k_θ and equilibrium angle θ_0. The angle styles are specified in the input script along with the type ("wlc" or "harmonic"), followed by the force constant (for both styles) and the equilibrium angle if harmonic angles are used.

B. Non-bonded interactions

Long-range interactions include repulsive interactions mediated by the Flory–Huggins χ parameter and the electrostatic forces acting between the charged monomers. The distinctive feature of MATILDA.FT is the way in which it handles these interactions. While bonded interactions use explicit coordinates to calculate inter-particle distances, long-range interactions use the mass/charge density field to compute the resulting forces. In this process, a Particle-to-Mesh (PM) method is used.[40] In this scheme, the box is divided into a discrete grid, with the number of grid-points in each direction being a user-defined quantity. At the beginning of the simulation, a Fourier-space representation of the inter-particle potential is calculated using the Fast Fourier transform (FFT) and is stored for the rest of the simulation. We note that no cut-offs are needed for the potentials. Next, at each time-step, every particle assigns its density contribution to nearby grid points, using a spline interpolation scheme with the weights given in the Appendix of Ref. 26. The order of the interpolating spline can be chosen from 1 to 4, with higher order interpolation requiring more computation. Regardless of the form of the pair potential $u(r)$, forces are given in real space by

$$f(\mathbf{r}) = -\int d\mathbf{r}' \nabla u(\mathbf{r}-\mathbf{r}')\rho(\mathbf{r}'). \tag{25}$$

The convolution in Eq. (25) is evaluated in Fourier space using FFTs, where it becomes a simple multiplication. Then, an inverse FFT is used to transform the forces back into real space where the forces are interpolated back onto the particle centers. The process is illustrated schematically in Fig. 5. The current pair potentials implemented in MATILDA.FT includes Gaussian forms as well as the nanoparticle-nanoparticle and nanoparticle-monomer forms demonstrated in previous work by some of us.[18]

The other primary non-bonded potential implemented in the particle-based methods is electrostatic interactions. The electrostatic potential $\phi(\mathbf{r})$ is calculated by solving the Poisson equation,

$$\nabla^2\phi(\mathbf{r}) = -4\pi l_B \breve{\rho}_c(\mathbf{r}), \tag{26}$$

where l_B is the Bjerrum length and $\rho_c(\mathbf{r})$ is the Gaussian-smeared charge density. Equation (26) is readily solved in Fourier space for a given charge density, and the electric field is computed as the gradient of the electrostatic potential. Since forces from pair potentials as in Eq. (25) and solving Poisson's equation both require the Fourier transform of a density field, an important feature of our particle-based methods is that including long-range electrostatics is no more expensive than other interaction potentials.

As an example system that incorporates polymer connectivity, excluded volume interactions, and electrostatics, a system consisting of a total of 434 molecules of polymer chains, each with a degree of polymerization, $N = 82$, has been simulated. Half of these molecules carry positively charged monomers, with the other half having each monomer with negative charges of the same magnitude, and there is an excluded volume repulsion that penalizes overlaps of the polymer chains. No explicit solvent is present. This simulation is a particle model of the system considered in previous work.[41] The system starts in a random, homogeneous phase and, over the course of the simulation phase, separates into polymer-rich and polymer-depleted regions as coacervation occurs, and the concentration in the droplets agrees with the previous field theoretic simulations.[41] The snapshots from the beginning (left) and the end (right) are shown in Fig. 6. On an Nvidia Quadro RTX 5000 GPU, this simulation took 1399 s to perform 2 000 000 time steps, with the resulting speed of 1429.6 ts/s. This example can be found in the GitHub repository, in the `examples/coacervate` directory.

C. Polarizable monomers

Polarization can play an important role in the phase behavior of polymer solutions, especially in the biological context. Molecular polarizability has an influence on polymer solubility, and it can also modify how polymer chains interact with salt ions present in the environment. In MATILDA.FT, polarizability effects are introduced through the use of classical Drude oscillators. In this approach, a "Drude particle" is attached to the parent particle via a harmonic spring with stiffness k_D and zero equilibrium length. This Drude particle is assigned a partial charge δq_D and the partner particle $-\delta q_D$, such that the net charge of the two-particle pair remains unchanged. The magnitude of the spring constant k_D can be related to molecular polarizability, with polarizability decreasing with increasing stiffness. The Drude particle gets assigned a small mass so that it can be integrated with other particles using standard equations of motion. This simplification circumvents the issue of treating polarizability effects on the quantum-mechanical level, while still being able to reproduce spatial variations in polarization. Drude particles do not participate in excluded volume interactions, and thus, the only forces acting on them are electrostatic in nature. We note that our implementation is different from those typically used in atomistic or more fine-scale coarse-grained models, where the Drude particle is typically thermostatted independently at a low temperature, enabling the polarizability to be estimated with the classical expression $\alpha = \delta q_D^2/k_D$. Since our charges are distributed over a unit Gaussian, this expression does not apply, and we have parameterized our effective dielectric constant as a function of the various parameters of the Drude oscillators (q_D, k_D, and the spread of the Gaussian, σ), which is shown in Fig. 7. Since the use of Drude oscillators introduces more particles to the simulation box, there is an associated increase in the computational cost of simulating polarizable materials through the increase in the number of particles in the simulation box.

As an example of a system that incorporates polarization on a polymer chain, 1330 diblock co-polymer chains with $N = 74$, and blocks of equal size, were simulated in explicit solvent. Both the solvent and polymer chains were made polarizable through the use of classical Drude oscillators. Since, in this system, only the polymer concentration of the condensate was of interest, all simulations were

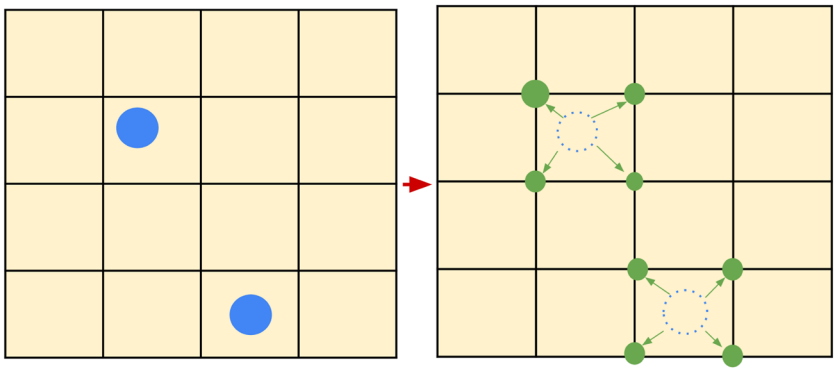

FIG. 5. Schematic illustration of the Particle-to-Mesh (PM) scheme. Specifically, shown here is a first-order spline interpolation, where the particle density is mapped to the two nearest grid points in each dimension. The same spline weights are used to map the forces back to the particles.

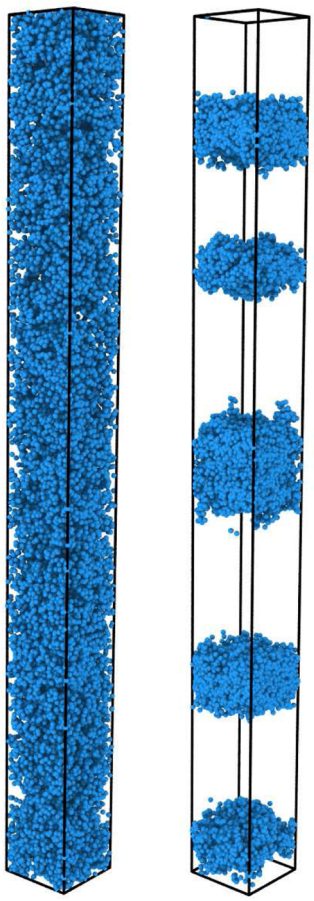

FIG. 6. Initial (left) and final (right) snapshot from the simulation of coacervating binary mixture. This simulation is a particle-based implementation of the model considered previously as a field-theory by one of us[41] with a dimensionless excluded volume parameter $B = 0.05$ and dimensionless Bjerrum length $E = 10\,000$.

started with the polymers in a dense "slab" configuration. In this configuration, all particles are biased to migrate toward the middle of the simulation box, creating a homogeneous, dense polymer phase. During the production run, this bias is removed, and the slab is allowed to expand.

The parameters for the Drude oscillator attached to the solvent molecules have been chosen to reproduce the dielectric constant of water. In this way, the Bjerrum length can be set to the value it has in the vacuum, and dielectric screening is then emergent from the polarizable solvent. The calculated dielectric constant for the chosen combination of parameters is shown in Fig. 7. The dielectric constant of water is around 78.4 so the optimal choice of the parameter corresponds to $a_0 \approx 0.5$ and $k_D \approx 1.0$ In Fig. 8, we show the plot of the reduced concentration c^* of the dense and dilute phases and the corresponding value of χ between the polymer monomers and the solvent molecules. We also include corresponding snapshots of the final structure of the system.

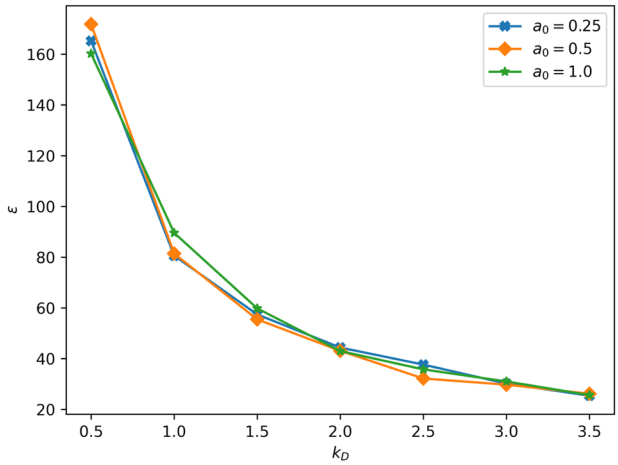

FIG. 7. Calculated values of dielectric constant for the solvent molecule, as a function of charge spreading length, a_0, and the spring constant of the Drude oscillator, k_D.

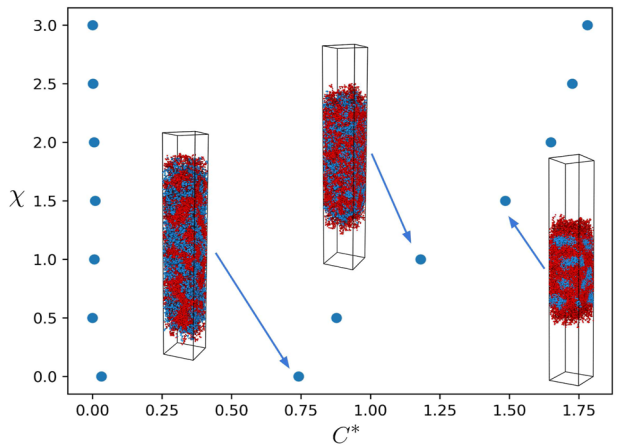

FIG. 8. Plot of the density obtained in the dilute and dense phases as the value of χ between the monomers and solvent is varied. Included are also renders of the three-dimensional structure of the system corresponding to the selected data points. Positively charged monomers are displayed in red, while negatively charged ones are colored blue.

D. Liquid crystals

We model liquid crystalline interactions through a modified Maier–Saupe (MS) potential that is a discrete version of the McMillan model.[42] In our implementation, the MS interactions involve two pairs of particles: one of each pair is the "center" of the interaction i and the other becomes a partner particle j that is used to define the local molecular orientation on particle i, $\mathbf{u}_i = \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|}$ (see schematic in Fig. 9 below). The local orientation vector is used to define an orientation tensor for each particle $\mathbf{S}_i = \mathbf{u}_i\mathbf{u}_i - \mathbf{I}/\mathbb{D}$, which is mapped onto an orientation field similar to the density fields,

$$\mathbf{S}(\mathbf{r}) = \sum_i \mathbf{S}_i \delta(\mathbf{r} - \mathbf{r}_i). \tag{27}$$

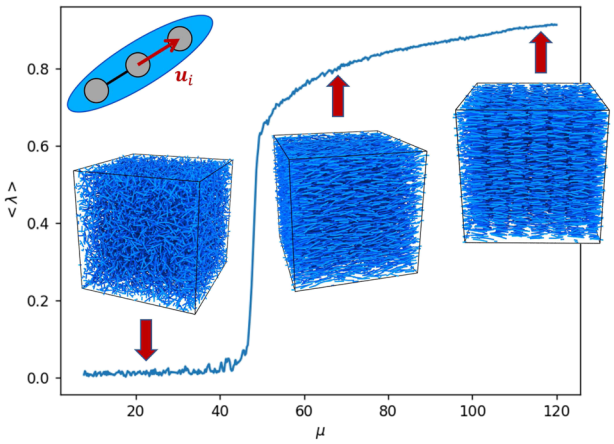

FIG. 9. Liquid crystalline order parameter as a function of the strength of the Maier–Saupe parameter μ calculated during a simulation where μ was continuously ramped throughout.

In Eq. (27), the sum is over the LC centers. The orientation field $\mathbf{S}(\mathbf{r})$ is then used to compute the Maier–Saupe potential energy,[43]

$$\beta U_{MS} = -\frac{\mu}{\rho_0}\int d\mathbf{r}\int d\mathbf{r}'\mathbf{S}(\mathbf{r}):\mathbf{S}(\mathbf{r}')u_G(|\mathbf{r}-\mathbf{r}'|), \qquad (28)$$

where μ is the Maier–Saupe potential parameter and $u_G(r)$ is the Gaussian potential that renders the interactions non-local. The forces are derived by explicit differentiation and are presented in the documentation of the code, and as we show below, this model captures both nematic and smectic A phases.

Particles that carry an orientation vector $\mathbf{u}_i$ are specified in an additional input file that is similar, in nature, to the lists of bonded partners. When specifying that the MS potential is to be used, the name of the additional input file is also provided; this file contains a list of pairs of particles i and j that are used to define the orientation vector associated with particle i. This implementation allows for the easy creation of either main-chain liquid crystalline polymers or side-chain liquid crystalline polymers, a detailed study of which will be the subject of a forthcoming publication. Furthermore, by making one of the end sites within an LC mesogen a different site type, one can indirectly control anchoring conditions at phase boundaries by making this other type more or less repulsive with a particular species in the nearby phase.

A simple model of a pure liquid crystal was simulated where the mesogen was discretized into three interaction sites with the Maier–Saupe (MS) interaction taken from the center of the mesogen, see the inset in Fig. 9. Bonds between adjacent liquid crystal sites used a force constant $k_b = 100$ and equilibrium distance 1, and the orientation was maintained with a worm-like chain angle potential with prefactor $\lambda = 50$. Finally, an additional Helfand potential was employed to maintain an approximately uniform density with $\kappa = 100$, and the total site density was $\rho_0 = 3$. This combination of the anisotropic molecular shape with a MS interaction taken from its center makes the model similar to the McMillan mean-field model.[42]

Figure 9 shows the average liquid-crystalline order parameter λ that is calculated on the fly as the MS μ parameter is increased from 0 to 120. λ is taken as 3/2 times the largest eigenvalue of the average $\mathbf{S}$ tensor. When $\mu \approx 45$, a sharp increase in $\langle\lambda\rangle$ indicates the isotropic-to-nematic transition. A more subtle feature at high $\mu \approx 105$ is indicative of the nematic to smectic transition.

VI. PLANNED FUTURE DEVELOPMENTS

In this section, we present the changes we plan to implement in MATILDA in the near-future. The code is still in development, and we plan to keep extending its capabilities and optimizing the existing algorithms. The main change we intend to implement in the particle-based methods (TILD) is to convert it to a fully object-oriented style, following the design of the newer FT branch. This will enable us to fully utilize the utilities provided in the Thrust library. It will also make the two branches operate more seamlessly, making future modifications easier and minimizing the learning curve for researchers wanting to modify the code. As High-Performance Computing (HPC) clusters are often equipped with multi-GPU nodes, we are planning to extend our software to be able to take advantage of multiple GPUs to further extend its parallelism. This would enable an efficient implementation of enhanced sampling techniques, such as parallel tempering where multiple simulation boxes are present at once. By accessing multiple GPUs at once, each simulation box could execute concurrently on separate devices.

We also plan to implement several key features for the FT simulation methods that should be available in the near term. While the time evolution equations of the fields described above are presented from the perspective of a complex Langevin (CL) simulation, the CL equations are not yet implemented, and all of the equations of motion are currently noise free, leading to mean-field solutions. Finally, the inclusion of electrostatic interactions, including polar and polarizable polymer monomers,[44,45] is planned in the near future.

Having a code base capable of simulating both particles and field-theoretic methods also has the potential to unlock novel simulations, especially since the two methods' strengths are complementary. Particle-based simulations are more efficient with lower molecular weight polymers and dilute solutions [i.e., at low $C = n/(V/Rg^3)$], while field-theoretic simulations are more powerful as C increases. One could, thus, envision exploiting both of these strengths to simulate phase coexistence using the Gibbs ensemble.[46] If one of the coexisting phases is polymer-rich and the other polymer-depleted, such as in complex coacervate-forming systems, the polymer-rich simulation box could be considered using field-theoretic approaches while the polymer-depleted box could be studied with particle methods. While the implementation will surely have details to resolve, since the particle-to-field transformation proceeds independently in the two simulation boxes,[47] the approach is expected to be viable.

VII. DISCUSSION AND CONCLUDING REMARKS

In this paper, we have presented MATILDA.FT, an open-source simulation software for highly coarse-grained soft matter simulations that is designed to run on GPUs. Both the particle-based and field-theoretic methods implemented are designed for potentials that are finite at the overlap and where the particle density will be relatively high. In the particle-based simulations, this leads to

a gain in overall efficiency by evaluating the non-bonded interactions using density fields rather than neighbor lists, and the finite potentials eliminate so-called ultraviolet divergences from field-based simulations.[28] As far as we are aware, MATILDA.FT is the first published open-source software to combine both coarse-grained Langevin dynamics and field-theoretic simulation frameworks into a single code base. As an example of the speedup generated by the GPU code, we performed a brief 2D simulation of a system containing 6172 polymer chains of length $N = 25$ each in a square simulation where the non-bonded forces were evaluated on a $M = 63 \times 63$ grid. Using one of our group's old serial codes, it took ~15 min to complete 10 000 time steps on an 2.2 GHz Intel I7-10870H laptop processor; MATILDA.FT completed the same number of time steps in 15 s on a laptop with Nvidia GeForce RTX 3080 GPU.

ACKNOWLEDGMENTS

This work used Bridges-2 GPU at Pittsburgh Supercomputing Center through Allocation No. DMR150034 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) program, which is supported by the National Science Foundation under Grant Nos. 2138259, 2138286, 2138307, 2137603, and 2138296. This work was supported by the National Science Foundation through Grant Nos. MRSEC/DMR-1720530 (R.A.R. and partial Z.M.J.), NSF awards OISE-1545884 (C.T.), and CHE-220375 (C.G.), and the Department of Physics and Astronomy at the University of Pennsylvania (Z.M.J.). The images used in this publication were generated using the free visualization software Ovito version 2.9.0.[48] C.T. is grateful for the helpful discussions with Zachariah Vicars.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Zuzanna M. Jedlinska: Software (supporting); Validation (equal); Visualization (lead); Writing – original draft (lead). **Christian Tabedzki**: Software (supporting); Writing – review & editing (supporting). **Colin Gillespie**: Software (supporting); Writing – review & editing (supporting). **Nathaniel Hess**: Software (supporting). **Anita Yang**: Software (supporting). **Robert A. Riggleman**: Conceptualization (lead); Software (lead); Validation (equal); Visualization (supporting); Writing – original draft (supporting); Writing – review & editing (lead).

DATA AVAILABILITY

The source code is open source and available under the Gnu public license (GPL2) at www.github.com/rar-ensemble/MATILDA.FT. The "examples" folder contains all of the input files needed to reproduce the simulations presented herein.

REFERENCES

[1] A. S. Indulkar, Y. Gao, S. A. Raina, G. G. Z. Zhang, and L. S. Taylor, Mol. Pharm. **13**, 2059 (2016).
[2] H. V. Sureka, A. C. Obermeyer, R. J. Flores, and B. D. Olsen, ACS Appl. Mater. Interfaces **11**, 32354 (2019).
[3] W. Shi and D. A. Weitz, Macromolecules **50**, 7681 (2017).
[4] M. F. Haase, H. Jeon, N. Hough, J. H. Kim, K. J. Stebe, and D. Lee, Nat. Commun. **8**, 1234 (2017).
[5] D. R. Tree, K. T. Delaney, H. D. Ceniceros, T. Iwama, and G. H. Fredrickson, Soft Matter **13**, 3013 (2017).
[6] M.-T. Wei, S. Elbaum-Garfinkle, A. S. Holehouse, C. C.-H. Chen, M. Feric, C. B. Arnold, R. D. Priestley, R. V. Pappu, and C. P. Brangwynne, Nat. Chem. **9**(11), 1118 (2017).
[7] S. F. Banani, H. O. Lee, A. A. Hyman, and M. K. Rosen, Nat. Rev. Mol. Cell Biol. **18**(5), 285 (2017).
[8] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, Comput. Phys. Commun. **271**, 108171 (2022).
[9] J. C. Phillips, "NAMD - Scalable molecular dynamics" (2022).
[10] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, SoftwareX **1-2**, 19 (2015).
[11] G. Fredrickson, *The Equilibrium Theory of Inhomogeneous Polymers* (Oxford University Press on Demand, 2006), Vol. 134.
[12] G. H. Fredrickson and K. T. Delaney, *Field-Theoretic Simulations in Soft Matter and Quantum Fluids* (Oxford University Press, 2023), Vol. 173.
[13] S. F. Edwards, Proc. Phys. Soc. **85**, 613 (1965).
[14] L. Leibler, Macromolecules **13**, 1602 (1980).
[15] W. Zheng and Z.-G. Wang, Macromolecules **28**, 7215 (1995).
[16] S. T. Milner, T. A. Witten, and M. E. Cates, Macromolecules **21**, 2610 (1988).
[17] S. F. Edwards and K. F. Freed, J. Phys. C: Solid State Phys. **3**, 739 (1970).
[18] J. Koski, H. Chao, and R. A. Riggleman, J. Chem. Phys. **139**, 244911 (2013).
[19] A. Arora, J. Qin, D. C. Morse, K. T. Delaney, G. H. Fredrickson, F. S. Bates, and K. D. Dorfman, Macromolecules **49**, 4675 (2016).
[20] G. K. Cheong, A. Chawla, D. C. Morse, and K. D. Dorfman, Eur. Phys. J. E **43**, 15 (2020).
[21] F. A. Detcheverry, H. Kang, K. C. Daoulas, M. Müller, P. F. Nealey, and J. J. De Pablo, Macromolecules **41**, 4989 (2008).
[22] D. Q. Pike, F. A. Detcheverry, M. Müller, and J. J. de Pablo, J. Chem. Phys. **131**, 084903 (2009).
[23] K. C. Daoulas and M. Müller, J. Chem. Phys. **125**, 184904 (2006).
[24] V. Ganesan and V. Pryamitsyn, J. Chem. Phys. **118**, 4345 (2003).
[25] H. Chao, J. Koski, and R. A. Riggleman, Soft Matter **13**, 239 (2017).
[26] M. Deserno and C. Holm, J. Chem. Phys. **109**, 7678 (1998).
[27] L. Schneider and M. Müller, Comput. Phys. Commun. **235**, 463 (2019).
[28] M. C. Villet and G. H. Fredrickson, J. Chem. Phys. **141**, 224115 (2014).
[29] A. Weyman, V. G. Mavrantzas, and H. C. Öttinger, J. Chem. Phys. **155**, 024106 (2021).
[30] E. Helfand, J. Chem. Phys. **62**, 999 (1975).
[31] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, 2017).
[32] G. Parisi, Phys. Lett. B **131**, 393 (1983).
[33] J. R. Klauder, Phys. Rev. A **29**, 2036 (1984).
[34] P. Stasiak and M. W. Matsen, Macromolecules **46**, 8037 (2013).
[35] E. M. Lennon, G. O. Mohler, H. D. Ceniceros, C. J. García-Cervera, and G. H. Fredrickson, Multiscale Model. Simul. **6**, 1347 (2008).
[36] G. Lab, "GSD 2.7.0 documentation" (2023).
[37] N. Bell and J. Hoberock, *GPU Computing Gems Jade Edition* (Elsevier, 2012), pp. 359–371.
[38] K. Zhang, "On the concept of static structure factor," arXiv:1606.03610 [cond-mat] (2016).
[39] H. D. Ceniceros and G. H. Fredrickson, Multiscale Model. Simul. **2**, 452 (2004).

[40]R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (CRC Press, 2021).
[41]R. A. Riggleman, R. Kumar, and G. H. Fredrickson, J. Chem. Phys. **136**, 024903 (2012).
[42]W. L. McMillan, Phys. Rev. A **4**, 1238 (1971).
[43]V. Pryamitsyn and V. Ganesan, J. Chem. Phys. **120**, 5824 (2004).
[44]J. M. Martin, W. Li, K. T. Delaney, and G. H. Fredrickson, J. Chem. Phys. **145**, 154104 (2016).
[45]A. Garaizar and J. R. Espinosa, J. Chem. Phys. **155**, 125103 (2021).
[46]A. Z. Panagiotopoulos, Mol. Simul. **9**, 1 (1992).
[47]R. A. Riggleman and G. H. Fredrickson, J. Chem. Phys. **132**, 024104 (2010).
[48]A. Stukowski, Modell. Simul. Mater. Sci. Eng. **18**, 015012 (2010).