

# MATILDA.FT

## Mesoscale, Accelerated, Theoretically-Informed, Langevin, Dissipative particle dynamics, and Field Theory

### Documentation

## 1 Introduction

MATILDA.FT is a CUDA-enabled simulation package that can perform two styles of simulation: theoretically-informed Langevin dynamics (TILD) and polymer field theoretic simulations (FTS). TILD is a unique style of simulation where the beads on different molecules do not directly interact with each other but instead interact with a density field that is determined by mapping bead coordinates onto a grid, in contrast for most common particle simulations where beads directly interact with each other. TILD is closely related to techniques pioneered by the de Pablo and Müller research groups.

FTS is another approach for simulating the coarse-grained models used in TILD where the particle coordinates are integrated out and replaced with chemical potential fields using a variety of analytic techniques, such as Hubbard-Stratonovich transformations. FTS is most commonly implemented as self-consistent field theory (SCFT), though MATILDA.FT will eventually allow for sampling field fluctuations using complex Langevin sampling.

MATILDA.FT is maintained by the Riggleman research group at the University of Pennsylvania and is hosted on [github.com/rar-ensemble/MATILDA.FT](https://github.com/rar-ensemble/MATILDA.FT). All questions, comments, and feedback should be directed to Rob Riggleman, [rrig@seas.upenn.edu](mailto:rrig@seas.upenn.edu).

## 2 Compiling MATILDA.FT

MATILDA.FT has been designed to run on Nvidia GPUs. To compile the code, *CUDA toolkit* and *nvcc* compiler are required. This version of the code has been tested with CUDA versions 11.4 and 11.8.

Before compiling, one must make a small edit the makefile in `MATILDA.FT/SRC`. Set the `CUDA_LOC` variable to point to the top of the directory containing the CUDA library. A common example would be to edit it to read

```
CUDA_LOC = /usr/local/cuda-11.4
```

The code should compile with a simple “make” command from within the `src` folder. If successful, the executable `matilda.ft` should appear in the top level directory.

### Troubleshooting

On some older GPU architectures, it may be necessary to set architecture flag `-arch` before compiling. In the makefile, example options are shown on the commented-out `CFLAGS` lines.

We have internally successfully compiled and run MATILDA.FT on the following systems

- 1) Ubuntu 18.04, GCC 7.5.0, CUDA 11.8, GPU: NVIDIA GeForce RTX 2060
- 2) Ubuntu 20.04, GCC 9.4.0, CUDA 12.0, GPU: NVIDIA Quadro RTX 5000
- 3) Bridges2 PSC HPC Cluster, GCC 8.3.1, CUDA 11.7, GPU: NVIDIA Tesla V100-SXM2
- 4) Windows subsystem for Linux on Windows 11, GCC 9.4.0, CUDA 11.4, GPU: NVIDIA GeForce RTX 3080 (laptop version)

## Output Files: Particle-based (TILD) Simulations

`data.dat`: written every `log_freq` time steps, contains the various contributions to the energy in a plain text file.

`traj.lammpstrj`: written every `traj_freq` time steps, is a LAMMPS-compatible lammpstrj text file. Will use a lot of disk space if written frequently.

`traj.gsd`: written every `gsd_freq` time steps, binary GSD format developed by Glotzer group. Can be read by OVITO.

`grid_densities.bin`: appended every `binary_freq` time steps. `grid_densities` contains the density fields for each species in a binary format. Can be converted to a `.tec` plain text file using the `dump-grid-densities.cpp` program in `utilities/post-proc/`, which should compile with  
`g++ -lm -O3 dump-grid-densities.cpp -o dump-grid-densities.`

`positions.bin`: appended every `binary_freq` time steps. `positions.bin` contains the positions of all the particles in a custom binary format. Can be converted to a `.lammpstrj` plain text file using the `dump-particles.cpp` program in `utilities/post-proc/`, which should compile with  
`g++ -lm -O3 dump-particles.cpp -o dump-particles.`

## 3 Input scripts

### 3.1 Particle-based simulation

A sample input script, providing basic TILD functionality is shown in Listing 1. Here, the simulation is performed in 2 dimensions for 10,000 time steps. The script specifies various log and output frequencies. The variable `pmeorder` controls the order of interpolation for constructing the density fields. The dimensionless time step is set to 0.005. The configuration data is read from the `input.data` file. An external force `midpush` (along the y-axis, with the dimensionless magnitude of 0.5) is applied to all particles. The integrator is chosen to be GJF. The lines beginning with `pair_style` specify the repulsive interactions between the selected particles types. A more detailed description of all available options can be found below.

```
1 Dim 2
2
3 max_steps 10001
4 log_freq 1000
5 binary_freq 1000
6 pmeorder 1
7 traj_freq 1000
8
9 delt 0.005
10 read_data input.data
11
12 Nx 65
13 Ny 65
14
15 extraforce all midpush 1 0.5
16
17 integrator all GJF
18
19 potential gaussian 1 1 1.5625 1.0
20 potential gaussian 2 2 1.5625 1.0
21 potential gaussian 1 2 3.00 1.0
```

Listing 1: Example input script

## 4 Commands

### 4.1 Command line arguments

- `-ft`

Sets the simulation style to FT

- `-particle`

Sets the simulation style to TILD

- `-in [file_name]`

Changes the input script from default `input`. This TILD option is used in the command line argument not in the input script.

### 4.2 Input Script Commands

#### 4.2.1 TILD Commands

- `angle [int] [string] arguments`

Defines angle style to be used. First int: angle type. String: angle style, currently only “wlc” and “harmonic” supported. Float: force constant.

Worm-like chain (“wlc”) potential:  $u(\theta_{ijk}) = k_{\theta}(1 + \cos(\theta_{ijk}))$ .

arguments = [float  $k_{\theta}$ ]

Harmonic (“harmonic”) potential:  $u(\theta_{ijk}) = k_{\theta}(\theta_{ijk} - \theta_0)^2$ .

arguments = [float  $k_{\theta}$ ] [float  $\theta_0$ ]

- `bond [integer bond type] [float prefactor] [float equilibrium bond separation]`

Parameters for the harmonic bond of the form  $u(r_{ij}) = k(r_{ij} - r_0)^2$ .

- `binary_freq [integer]`

Frequency for writing to the binary data files

- `equil_binary_freq [integer] (optional argument)`

Frequency for writing equilibration data to binary data files. Requires equilibration data collection to be set using `equil_steps` or `prod_steps`. If not used, defaults to `binary_freq`.

- `charges [float] [float]`

Indicates to read in a charge configuration (follows LAMMPS “charges” format). The first float is the Bjerrum length for electrostatic interactions, the second is the smearing length for the charges (assumed Gaussian distributions).

- `compute [style] [arguments]`

Enable compute for on-the-fly computations. Currently enabled styles:

- avg\_sk [integer type] [optional arguments]

Optional arguments are: “wait [integer number of time steps]” and “freq [integer number of time steps]”

This compute calculates a running average of the static structure factor on the fly based on the density field of the indicated species type. The “wait” option delays the calculation for some number of time steps (default = 0), and the “freq” option sets the frequency of performing the compute (default = 100 time steps).

- Widom [molec ID] [number configs] This compute uses Widom insertions to grow a molecule to estimate the chemical potential. The topology and particle types are taken from molecule “molec ID”, and “number configs” is the number of trial configurations to generate at each call to this compute.

This compute generates output “Widom.dat” containing the calculated energies of the inserted molecule.

Optional arguments are the same as for avg\_sk.

- chemical\_potential [int min. molec ID] [int max molec ID] [float molec frac] [optional arguments]

This compute calculates the change in potential energy of a system upon removal of one of the molecules in the range given. The minimum and maximum molecule IDs provided are inclusive. The molecule fraction argument is the fraction of molecules in the range to remove (they are chosen at random) at each estimate. This will affect the computational expense of the method.

This compute generates an output “chemical\_potential.dat” containing the change in energy each time the compute was performed.

Optional arguments are the same as for avg\_sk.

- delt [float]

Size of the time step

- diffusivity [int A] [float D]

Sets diffusivity of beads of type A. Only used with GJF integrator

- Dim [integer]

Sets the dimensionality of the system.

- extraforce [string group] [string style] [arguments]

Adds extra forces to the particles in specified group. Supported styles currently include:

- dpd [string neigh\_list] [float sigma] [float  $r_{cutoff}$ ]

Activates dissipative particle dynamics in the simulation. Should be coupled with the Velocity Verlet integrator.

- langevin [float gamma]

Adds Langevin noise and friction to the forces, would need to be used with velocity Verlet algorithm to give proper thermostating.

- lewis [string n\_list] [float  $k_{spring}$ ] [float  $e_{bond}$ ] [float  $r_0$ ] [float  $q_{ind}$ ] [int bond\_freq] [int bond\_log\_freq] [string bond\_log\_file\_name]

Enables dynamic bonding in the system. Setting the induced charge to a non-zero value enables the formation of induced dipole between the bonded particles.

It generates a log file of dynamic bonds in the system. The format is:

```

TIMESTEP: [timestep] [no. bonds] [free donors] [total donors]
[donor id] [acceptor id]
[donor id] [acceptor id]
...
TIMESTEP: [timestep] [no. bonds] [free donors] [total donors]
...
```

It reports the global id of the particle.

- midpush [int axis] [float magnitude]

Adds a force of given magnitude pushing molecules towards the center of the simulation box along the chosen axis. This is useful for creating macrophase separated systems with a particular species in the center of the box. The magnitude of the forces should generally be small.

- wall [string wall\_style] [args]

Adds impermeable walls to the simulation box. Walls repel particles towards the center of the enclosed space. Arguments are passed as [int dim] [args], optional [int dim] [args] ..., where [dim] is the axis on which the walls will be placed x = 0, y = 1, z = 2. Different wall styles have different repulsive potentials. Available styles include:

- \* hard [int dim] [float low] [float high]

Creates hard walls in the simulation box

Example: extraforce all wall hard 1 3.0 15.5 2 2.5 4.5 will place confining walls along the y-axis, one at y = 3.0 and another at y = 15.5.

A second set of walls will be added along the z-axis at z = 2.5 and 4.5

- \* exp [int dim] [float A] [float low] [float high]

Repels particles with potential  $V = Ae^r$ , where  $r$  is the distance of particle's COM from the corresponding wall

- grid\_freq [integer]

Frequency of writing the grid data to a human-readable text file. Use sparingly.

- equil\_grid\_freq [integer] (optional argument)

Frequency for writing equilibration grid data to a human-readable text file. Requires equilibration data collection to be set using equil\_steps or prod\_steps. If not used, defaults to grid\_freq.

- gsd\_freq [integer]

Frequency of writing the trajectory to the GSD file format, written by the Glotzer lab.

- `gsd_name` [string]

Changes the name of writing GSD file. Default is “traj.gsd”.

- `group` [string name] [string style] [style options...]

- `type` [int type1 type2 ...]

Makes a group containing all particles of the listed types.

- `id` [string file]

Initializes the group based on global particle IDs (matching the data file) from a plain text file.

- `integrator` [group\_name] [style] [options]

Integration scheme to employ; current options are VV (Velocity Verlet) or GJF (Grønbech-Jensen and Farago)

- GJF

- VV [float `v_max`] [int distribution]

`c_max` - max magnitude of the initial velocity

distribution - distribution of velocities (0 - normal, 1 - uniform)

- `log_freq` [integer]

Frequency of writing to data.dat file. Requires host-device communication.

- `equil_log_freq` [integer] (optional argument)

Frequency for writing equilibration data to equil\_data.dat file. Requires equilibration data collection to be set using `equil_steps` or `prod_steps`. If not used, defaults to `log_freq`.

- `max_steps` [integer]

Total number of time steps

- `prod_steps` [integer] (optional argument)

Sets number of production steps. If called with `max_steps`, sets `equil_steps` to the difference and enables equilibration data collection. If called with `equil_steps`, sets `max_steps` to the sum. Note- you cannot call `max_steps`, `prod_steps`, and `equil_steps` in the same input script, or `prod_steps` greater than `max_steps`.

- `equil_steps` [integer] (optional argument)

Sets number of equilibration steps. If called with `max_steps`, sets `prod_steps` to the difference and enables equilibration data collection. If called with `prod_steps`, sets `max_steps` to the sum and enables equilibration data collection. Note- you cannot call `max_steps`, `prod_steps`, and `equil_steps` in the same input script, or `equil_steps` greater than `max_steps`.

- `nlist` [string group] [string style] [string name] [float  $r_n$ ] [float  $r_{skin}$ ] [float density] [float frequency] [options]

Constructs a neighbour list for a given group of particles.  $r_n$  is the cut-off radius of the associated

force.  $r_{skin}$  is the skin radius. Density specifies the capacity of each cell on the grid. Frequency is the update frequency of the neighbour list. Positive integer specifies the number of time-steps after which the update occurs. -1 triggers automatic, displacement based update.

- bonding [string file\_name]  
Creates a neighbour list to be associated with lewis force (dynamic bonding). Additional file maps particles to donor and acceptor types. The donors [type 1] and acceptors [type 0] are specified in the additional file that matches type to the id within the group.
- half\_distance  
Creates a neighbour list in which particles only store the information about their neighbours with a lower index than their own.
- Nx [integer]  
Ny [integer]  
Nz [integer]  
  
Sets the grid points in the x-, y-, and z-directions
- pmeorder [integer]  
  
Should be an integer 1, 2, 3, or 4. Determines the order of the spline used to map particle densities to the grid.
- potential [style] [style-specific arguments]
  - charges [float] [float]  
  
Indicates to read in a charge configuration (follows LAMMPS “charges” format). The first float is the Bjerrum length for electrostatic interactions, the second is the smearing length for the charges (assumed Gaussian distributions).
  - erf [integer type 1] [integer type 2] [float prefactor] [float std deviation]  
  
This defines a non-bonded interaction between particles of type 1 and 2 of the form  

$$u(r) = A \operatorname{erf}\left(\frac{|\mathbf{r}| - R_p}{\sigma}\right) * \operatorname{erf}\left(\frac{|\mathbf{r}| - R_p}{\sigma}\right),$$
 where the \* indicates a convolution operation.  
 Employs the `ramp` keyword
  - fieldphase [int type 1] [int type 2] [float prefactor] [int phase] [int dir] [int n periods]  
  
For the phase: 0 = Lamellar, 1 = BCC (NOT YET IMPLEMENTED), 2 = CYL, 3 = GYR.  
 Direction is the direction normal to lamellae, along the cylinders, and is ignore for GYR and BCC.  
 Employs the `ramp` keyword
  - gaussian [integer type 1] [integer type 2] [float prefactor] [float std deviation]  
  
This defines a non-bonded interaction between particles of type 1 and 2 of the form  

$$u_G(r_{ij}) = \frac{A}{(2\pi\sigma^2)^{D/2}} e^{-|r_{ij}|^2/2\sigma^2}$$
 Employs the `ramp` keyword.
  - gaussian\_erf [integer type 1] [integer type 2] [float prefactor] [float std deviation]  
  
This defines a non-bonded interaction between particles of type 1 and 2 of the form

$$u(r) = A \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-|r|^2/2\sigma^2} * \text{erf}\left(\frac{|r|-R_p}{\sigma}\right),$$

where the  $*$  indicates a convolution operation.

Employs the `ramp` keyword

- `maierSaupe [lc.input filename] [prefactor] [length scale]`

`lc.input` is an input file that defines the pairs to create orientation vectors.

`prefactor` is the ratio of  $\mu/2\rho_0$  used in the Maier-Saupe potential and `length scale` is the smearing length of the Gaussian. See Section 7 below for more details and the example “MaierSaupe” in `examples` folder.

- `rand_seed [integer]`

seed for the random number generator

- `read_data [string]`

Name of the input configuration file, should be in roughly LAMMPS data file format.

- `read_resume [string]`

Must be supplied along with the `read_data` command to read atom data and bonds.

Name of the resume file to set particle positions, assumed to be a `lammprj` frame format.

- `read_gsd [string]`

Allows you to read in a GSD file, which is a binary format for storing molecular dynamics trajectories. This is useful for reading in a trajectory from another simulation, or for reading in a configuration file. The GSD file format is written by the Glotzer lab.

Optional keywords: `frame [integer]` to read in a specific frame. `resume` to read in a GSD and only use the particle positions and timesteps. `read_config` to read in a GSD, using all information natively (including box size, particle types, bonds, angles, etc.) and use this as the initial configuration. `read_config` and `resume` are mutually exclusive.

Defaults: `frame` is -1, which will resume from the final frame of the GSD file. The code is set up to read resume by default. The code searches the inputted frame for the following information: particle positions, particle types, box size, timesteps, and bonds. If any of these are not found, the code will instead search the first frame for all that information. This behavior can be changed upon request.

- `set_timestep [intn]`

Sets the global timestep of the system; useful for keeping track of simulations that are resumed from a previous run.

- `threads [integer]`

Number of threads per GPU block to employ. Default is 512.

- `traj_freq [integer]`

Frequency for writing to `lammprj` text file. Use sparingly.

- `equil_traj_freq [integer]` (optional argument)

Frequency for writing equilibration data to `lammprj` text file. Requires equilibration data collection to be set using `equil_steps` or `prod_steps`. If not used, defaults to `traj_freq`.



#### 4.2.2 FT Commands

- boxLengths [float Lx] [float Ly] [optional float Lz]

The dimensions of the simulation box in each of the “Dim” directions.

- chemFieldFreq [integer number of steps]

Number of time integration steps between writing the chemical potential fields.

- densityFieldFreq [integer number of steps]

Number of time integration steps between writing the density fields.

- Dim [int dimension]

Dimensionality of the simulation box.

- grid [int Nx] [int Ny] [optional int Nz]

Number of grid points in each of the “Dim” directions. M is computed during this command as  $M = N_x N_y N_z$ .

- maxSteps [int max steps]

Maximum number of field updates to perform

- molecule [string “linear”] [float volume fraction] [int number of blocks] [block 1 length] [block 1 species] [block 2 length] [block 2 species]...

Creates a linear multiblock discrete Gaussian chain polymer

- Nr [integer reference chain length]

Length of the reference polymer chain used to non-dimensionalize the units of the simulation.

- potential [string potential style] [arguments]

The potential style must be either “Flory” or “Helfand” in the current version of the code.

##### **Flory potential:**

arguments = [string species I] [string species J] [float  $\chi_{IJ} N_r$ ] [float  $\delta t_+$ ] [float  $\delta t_-$ ] [optional arguments]

$\delta t_+$  and  $\delta t_-$  are the magnitude of the time steps used on the  $w_{IJ}^{(+)}$  and  $w_{IJ}^{(-)}$  fields, respectively.

##### **Helfand potential:**

arguments = [float  $\kappa N_r$ ] [float  $\delta t$ ] [optional arguments]

$\delta t$  is the size of the time step to be used on the corresponding potential field  $w_+$ .

##### **Optional arguments:**

- initialize [string style] [arguments]

String styles can be

- \* value [float  $w_r$ ] [float  $w_I$ ]

Initializes the field to a constant  $w = w_r + I w_I$ , where  $w_r$  and  $w_I$  are the real and imaginary parts of the field.

\* random [float  $w_{r,amp}$ ] [float  $w_{I,amp}$ ]

Initializes the field to random values with real and imaginary parts in the range  $\pm w_{r,amp}$  and  $w_{I,amp}$ , respectively.

\* sine [int direction] [float amplitude] [integer periods]

Initializes the potential fields as a sine wave in the specified integer direction  $\alpha = \{0, 1, 2\}$  with (0=x, 1=y, 2=z) and amplitude  $A$  and number of periods  $P$ . The functional form is  $w(\mathbf{r}) = A \sin \frac{2\pi r_\alpha P}{L_\alpha}$ .

– updateScheme [string update scheme]

Scheme to use to update the potential fields. Current options are *EM* and *1S* update schemes.

– modify [string modification]

Modifications to the potential. Current options are “zeroMean”, which zeros the mean of the chemical potential field.

• randSeed [int RNG seed]

Seed for the random number generator used to generate initial conditions.

• rho0 [float density]

Monomer density of the reference polymer melt in units of  $b^{-3}$ , where  $b$  is the statistical segment size of the reference polymer.

• tolerance [float tolerance]

Convergence tolerance for an SCFT calculation. Currently only compares the change in the real part of the Hamiltonian after logFreq number of steps for convergence criterion. Default value:  $10^{-4}$ .

• species [string species name]

Name of the chemical species, typically a letter (e.g., *A*, *B*, etc).

## 5 Utilities

In the directory “utils”, there are several files that can be used to post-process or generate initial configurations.

### 5.1 utils/post-proc

dump\_grid\_densities.cpp converts the given frames into .tec files for view in ParaView.

avg-grid-densities.cpp does not appear to actually average as promised...

calc\_sk/ subdirectory. Uses FFTW to calculate  $S(k)$  from the binary files.

dump\_particle\_posits.cpp dumps the binary particle positions into a lammpstrj file.

## 6 Interaction Potentials

### 6.1 Gaussians

A standard Gaussian governs the non-bonded interactions,

$$u(r) = \frac{A}{(2\pi\sigma^2)^{D/2}} \exp(-r^2/2\sigma^2) \quad (1)$$

$$= A u_G(r) \quad (2)$$

where  $A$  and  $\sigma$  are the amplitude and standard deviation of the Gaussian potential and  $u_G$  is a normalized Gaussian distribution in  $D$  dimensions. Entering the potential parameters in the input file is of the form

```
pair_style gaussian Itype Jtype A sigma
```

to calculate a potential energy of the form

$$U = A \int d\mathbf{r} \int d\mathbf{r}' \rho_I(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \rho_J(\mathbf{r}'). \quad (3)$$

#### 6.1.1 Relation to other field theory forms

Koski et al. [JCP 2013] and Villet and Fredrickson [JCP 2014] have shown the relationship between the “smearing functions” commonly used to regularize field-theoretic simulations (FTS) and non-bonded potentials,  $u(r) = A(h_I * h_J)(r)$ , where the  $*$  indicates a convolution, and  $h_I(r)$  and  $h_J(r)$  are the normalized smearing functions for species  $I$  and  $J$ . Smearing with a unit Gaussian is commonly used in FTS, which leads to an effective potential in Fourier space

$$\begin{aligned} \tilde{u}(k) &= A \tilde{h}_I(k) \tilde{h}_J(k) \\ &= A e^{-k^2 a_I^2/2} e^{-k^2 a_J^2/2} \\ &= A e^{-k^2 (a_I^2 + a_J^2)/2}. \end{aligned} \quad (4)$$

Or, in real space, the potential can be related to the smearing functions

$$u(r) = \frac{A}{(2\pi(a_I^2 + a_J^2))^{D/2}} e^{-r^2/2(a_I^2 + a_J^2)} \quad (5)$$

#### 6.1.2 Gaussian-Regularized Model A implementation

To implement a Gaussian regularized Edwards model (Model A from ETIP), the effective potential should simply be

$$u(r) = \frac{u_0}{2} u_G(r) \quad (6)$$

and so treating  $A = u_0/2$  in Eq. 2 gives the standard Model A.

#### 6.1.3 Compressible Blend (Model D)

The blend requires two components  $A$  and  $B$ , and for this section I’ll assume the range of the Gaussian potential ( $\sigma$ ) on all components is identical. Model D has the total interaction energy

$$\begin{aligned} U &= \frac{\kappa}{2\rho_0} \int d\mathbf{r} \int d\mathbf{r}' [\rho_+(\mathbf{r}) - \rho_0] u_G(|\mathbf{r} - \mathbf{r}'|) [\rho_+(\mathbf{r}') - \rho_0] \\ &\quad + \frac{\chi}{\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \rho_A(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \rho_B(\mathbf{r}'), \end{aligned} \quad (7)$$

where  $\rho_+ = \rho_A + \rho_B$ . Writing out  $\rho_+$  and collecting terms that are quadratic in the spatially-varying density, we arrive at the form typically used in TILD,

$$\begin{aligned} U = & \frac{\kappa + \chi}{\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \rho_A(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \rho_B(\mathbf{r}') \\ & + \frac{\kappa}{2\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \rho_A(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \rho_A(\mathbf{r}') \\ & + \frac{\kappa}{2\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \rho_B(\mathbf{r}) u_G(|\mathbf{r} - \mathbf{r}'|) \rho_B(\mathbf{r}') + u_{irr}, \end{aligned} \quad (8)$$

where  $u_{irr}$  are terms that are either constant or linear in the densities and thus have no thermodynamic relevance other than a shift in the energy scale. Thus, using two distinct prefactors in Eq. 2  $A_{II} = \kappa/2\rho_0$  and  $A_{IJ} = (\kappa + \chi)/\rho_0$ , we find

$$\kappa = 2\rho_0 A_{II} \quad (9)$$

$$\chi = \rho_0 (A_{IJ} - 2A_{II}). \quad (10)$$

### Example inputs

Binary mixture with a finite  $\chi$ :

```
gaussian 1 1 2.5 0.5
gaussian 2 2 2.5 0.5
gaussian 1 2 10.0 0.5
```

Two-component system that begins with  $\chi = 0$  in the above equations, then ramps  $\chi$  to larger values linearly over the simulation to a finite value:

```
gaussian 1 1 10.0 0.5
gaussian 1 2 20.0 0.5 ramp 20.4
gaussian 2 2 10.0 0.5
```

## 6.2 Erf Potential

This potential is used to generate smooth spheres within the simulation. The potential is defined as real-space as

$$u(r) = A \text{erfc} \left( \frac{|\mathbf{r}| - R_p}{\sigma} \right) \quad (11)$$

where  $A$  is the amplitude,  $R_p$  is the radius of the nanoparticles,  $\sigma$  controls the width of the nanoparticle. Entering the potential parameters in the input file is of the form

```
pair_style erf IType JType A Rp sigma ramp A_final
```

with `ramp` and `A_final` are optional parameters similar to the Gaussian parameters.

The potential in this case is the convolution between two spheres and thus should not be used between point particles and spheres (which is what “`gaussian_erfbelow`” is for).

The function is defined in kspace since the “erfc” function is the convolution between a Gaussian function and a box function. The kspace potential used is

$$U(k) = A u_G(k) \left( \frac{4\pi(\sin(R_p k) - R_p k \cos(R_p k))}{k^3} \right)^2 \quad (12)$$

To use the potential here

### 6.3 Gaussian-erfc cross potential

To model the interactions between the polymer and the spheres themselves, we use a modified version of the potential called `gaussian_erf`.

$$U(k) = A u_G(k) \frac{4\pi(\sin(R_p k) - R_p k \cos(R_p k))}{k^3} \quad (13)$$

### 6.4 Phase Fields

This potential is a static field that allows biasing into particular phases. The form of the pair style is

$$U = \int d\mathbf{r} [\rho_I(\mathbf{r}) - \rho_J(\mathbf{r})] A_o w(\mathbf{r}), \quad (14)$$

where  $w(\mathbf{r})$  takes values in  $\pm 1$  and has the symmetry of a desired phase. For example, one version might have  $w(\mathbf{r}) = \sin(2\pi x/L_x)$ , a sine wave to induce a single period of a lamellar structure in the  $x$ -direction. The ultimate amplitude is governed by  $A_o$ . One could imagine using such a field to initialize a system, then switching to the Gaussian potentials above to properly equilibrate and test the stability of a phase. While it won’t guarantee going into the equilibrium phase, it should enable the formation of defect-free structures.

This potential also includes `ramp` arguments like those of Gaussian and Erf.

NOTE: This should be used in conjunction with some kind of self-repulsion (e.g., a Gaussian potential between  $I - I$  and  $J - J$ ) to prevent large total density fluctuations.

## 7 Maier-Saupe Potential

The Maier-Saupe (MS) potential has been implemented using a modified form of the potential presented in Pryamitsyn and Ganesan, JCP V120 p5824 (2003). The form of the potential is

$$\beta U_{MS} = -\frac{\mu}{2\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \mathbf{S}(\mathbf{r}) : \mathbf{S}(\mathbf{r}') u_G(|\mathbf{r} - \mathbf{r}'|), \quad (15)$$

where  $\mathbf{S}(\mathbf{r})$  is a continuous tensor field that is constructed using information about the particle-level mesogen orientations and  $u_G(r)$  is a unit Gaussian to render the interactions non-local. Each particle  $i$  that represents a mesogenic center (i.e., one that interacts with the MS potential) carries an orientation that is defined using one of the neighboring particles,  $\mathbf{u}_i = \mathbf{r}_{ji}/|\mathbf{r}_{ji}|$  with  $\mathbf{r}_{ji} = \mathbf{r}_j - \mathbf{r}_i$ , and the index  $j$  represents the “partner” particle.

The particle-level tensor  $\mathbf{S}_i$  is defined as

$$\mathbf{S}_i = \mathbf{u}_i \mathbf{u}_i - \frac{\mathbf{I}}{\mathcal{D}}, \quad (16)$$

where  $\mathcal{D}$  is the dimensionality of the system and the product  $\mathbf{u}_i \mathbf{u}_i$  should be interpreted as an outer/tensor/dyadic product. The tensor field is defined on the grid as

$$\mathbf{S}(\mathbf{r}) = \sum_i^{n_{LC}} \delta(\mathbf{r} - \mathbf{r}_i) \mathbf{S}_i, \quad (17)$$

and usual particle-to-mesh schemes are used to discretize the delta function. Explicit differentiation of the force yields two contributions to the total force on particle  $j$  given as

$$\frac{\partial \beta U_{MS}}{\partial \mathbf{r}_j} = \frac{-\mu}{\rho_0} \int d\mathbf{r} \int d\mathbf{r}' \{ \delta(\mathbf{r} - \mathbf{r}_j) \mathbf{S}_j : \mathbf{S}(\mathbf{r}') \nabla_{\mathbf{r}} u(\mathbf{r} - \mathbf{r}') \} \quad (18)$$

$$\begin{aligned} & + \sum_i \delta(\mathbf{r} - \mathbf{r}_i) [\mathbf{I} \mathbf{u}_i + \mathbf{u}_i \mathbf{I}] \cdot \frac{\partial \mathbf{u}_i}{\partial \mathbf{r}_j} : \mathbf{S}(\mathbf{r}') u(\mathbf{r} - \mathbf{r}'), \\ & = \int d\mathbf{r} \int d\mathbf{r}' \{ \delta(\mathbf{r} - \mathbf{r}_j) \mathbf{S}_j : \mathbf{S}(\mathbf{r}') [-\frac{\mu}{\rho_0} \nabla_{\mathbf{r}} u(\mathbf{r} - \mathbf{r}')] \\ & - \sum_i \delta(\mathbf{r} - \mathbf{r}_i) [\mathbf{I} \mathbf{u}_i + \mathbf{u}_i \mathbf{I}] \cdot \frac{\partial \mathbf{u}_i}{\partial \mathbf{r}_j} : \mathbf{S}(\mathbf{r}') [\frac{\mu}{\rho_0} u(\mathbf{r} - \mathbf{r}')] \}. \end{aligned} \quad (19)$$

The first term (first line) represents how the potential changes as the particle carrying the unit vector is moved through space, and an integration by parts was used to get the derivative on the Gaussian potential. The second term represents how the unit vector changes as the particle moves. The sum over  $k$  is a sum over all MS interactions where particle  $j$  is involved in the definition of  $\mathbf{u}$ , including itself ( $k = j$ ). In practice, it is easier to accumulate the second term for  $k \neq j$  when the first term is evaluated for  $k$ . All of the double-dot product terms involve particle-level information on the left of the double dot and field info on the right.

Note: the prefactor read by the input file is the value of  $\mu/2\rho_0$  and a *positive* value should be given to promote nematic order.

## 8 Field-Theoretic Simulations

### 8.1 Overview

In the developing field theory code, many changes to the underlying architecture are being implemented. The simulation is going to live in a Box object, which will eventually lead to a scriptable input file. Within the Box object, an FTS simulation lives in an derived class FTS\_Box, and the simulation itself depends on three key objects:

1. **Species.** These are the number of unique chemical components, commonly given letters A, B, C, etc. in the field theory literature. Potentials below act on species classes.
2. **Potentials.** These govern the interactions and represent the various interaction types, such as the Helfand weak compressibility, Flory incompatibility, or the Edwards potential. Potentials act on species.
3. **Molecules.** Each element of this class is a particular molecule architecture. Initially, linear multiblock copolymers are the only implemented architecture, but stars, bottlebrushes, and grafted nanoparticles will all be part of the molecule class. Each block is defined as a species of some type, and the molecule object must calculate the propagators, densities, smeared densities if appropriate, and accumulate their density into the species objects for each type.

In the input script, Species must be defined before Molecules and Potentials since the molecules must be made of species and Potentials must act on Molecules. Species are simply given a name, typically a single letter (e.g. *A*, *B*, etc.), while molecules are comprised of one or more species.

### 8.2 FT Potentials

There are currently two potentials used in the simulations: a Flory incompatibility potential and the Helfand weak compressibility potential. The Flory potential takes the form of

$$\beta U_1 = \frac{\chi_{IJ} N_r}{C} \int d\mathbf{r} \check{\rho}_I(\mathbf{r}) \check{\rho}_J(\mathbf{r}), \quad (20)$$

where  $C$  is the potential concentration  $\chi_{IJ} N_r$  is the magnitude of repulsion between species  $I$  and  $J$ . The densities in Eq. 20  $\check{\rho}(\mathbf{r})$  are the smeared densities, not the particle-center densities, and the smearing is implemented in a way that is molecule specific. For example, species  $A$  could have two different molecules with different smearing functions that are “chemically identical”.

The Helfand compressibility potential takes the form

$$\beta U_2 = \frac{\kappa N_r}{2C} \int d\mathbf{r} [\check{\rho}_+(\mathbf{r}) - \rho_0]^2, \quad (21)$$

where  $\kappa N_r$  controls the strength of the repulsion and  $\check{\rho}_+(\mathbf{r})$  is the total smeared density of all molecules in the system.

Future versions of the code may allow for exclusion of specific species from this potential, which could be useful for neglected the excluded volume of ions or Drude particles, but this is not yet implemented.

### 8.3 Non-dimensionalization

The FT simulations and the results thereof are presented in dimensionless units. All of the potential fields  $w(\mathbf{r})$  are reduced by a reference polymer chain length  $N_r$ , which is specified in the input file. Furthermore, all length scales are reduced by the radius of gyration for this particular reference polymer,



$R_g = \sqrt{(N_r - 1)/6}$ . Note that  $N_r$  does *not* have to be equal to the length of any of the explicit species in the simulation box. Finally, one must specify *either* the polymer concentration of the reference melt of chains of length  $N_r$ ,  $C = n_r/V$ , where the volume here is already scaled by  $R_g^3$  or the monomer density of the same melt,  $\rho_0 = n_r N_r / (V R_g^3)$ .

## 8.4 Brief History

The GPU-TILD code evolved out of a series of programs originally referred to as dynamic mean-field theories (DMFT) developed by Chao, Koski, and Riggleman[?] and augmented by several students in subsequent years. Over time, it became clear that the approach was strongly connected to existing methods that went by the moniker theoretically-informed Monte Carlo (TIMC). The technical differences between the two is that our approach tends to use higher-order particle-to-mesh schemes to ensure UV-convergence and that time evolution is through Langevin dynamics in lieu of Monte Carlo dynamics. As a result we have subsequently referred to the method as theoretically-informed Langevin dynamics (TILD). Our group had several versions of an internal code that was developed using openMP and later openMPI. However, all of these codes suffered from a fairly rigid structure that was tied to simulations of specific systems associated with specific research projects and an restrictive input file format.

During the first summer of the Coronavirus pandemic, RAR began developing a basic TILD simulation package. The goal was to develop code intended to be used on GPUs from the ground up, creating a more modular, object-oriented structure. Once the utility of the code became clear, its use spread throughout our research group, with substantial additions to the code base being made by all of the co-authors of this work, most notably CT and ZMJ. The addition of the field-theoretic simulations portion of the code began in late 2022 by RAR.