NEEDS-A-NAME Code Documentation

This is a rough documentation of the CUDA-TILD code as it develops.

In V0.9, the code seems capable of handling the standard Gaussian chain polymer models. The input file is more flexible than other codes I have written, and the potentials use a class-based structure that should make extending the code more straightforward.

As of 3 May 2020, the only quantitative testing has been in 2D, so more testing needs to be done, particularly to make sure the magnitudes of χ work out as expected.

1 Interaction Potentials

1.1 Gaussians

A standard Gaussian governs the non-bonded interactions,

$$\begin{aligned} u(r) &= \frac{A}{(2\pi\sigma^2)^{D/2}}\ \exp(-r^2/2\sigma^2) && (1)\\ &= A\,u_G(r) && (2)\end{aligned}$$

where A and σ 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

```
gaussian Itype Jtype A sigma
```

to calculate a potential energy of the form

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

1.1.1 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

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

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

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

1.1.2 Gaussian-Regularized Model A implementation

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

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

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

1.1.3 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 (σ) on all components is identical. Model D has the total interaction energy

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

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

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

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

$$\kappa = 2\rho_0 A_{II} \tag{9}$$

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

Example inputs

Binary mixture with a finite χ:

```
n_gaussians 3
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 χ to larger values linearly over the simulation to a finite value:

```
n_gaussians 3
gaussian 1 1 10.0  0.5
gaussian 1 2 20.0 0.5 ramp 20.4
gaussian 2 2 10.0  0.5
```

1.2 Erf Potential

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

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

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

```
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_erf`below 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

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

To use the potential here

1.3 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`.

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

1.4 Phase Fields

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

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

where $w(\mathbf{r})$ takes values in ± 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.

2 Input Commands

2.1 Current input commands

- -in [file_name]

  Changes the input script from default `input`. Is used in the command line argument not in the input script.

- angle [int] [string] [float]

  Defines angle style to be used. First int: angle type. String: angle style, currently only “wlc” supported. Float: force constant.
  Only currently supported format is the WLC potential $u(\theta_{ijk}) = k_\theta(1 + \cos(\theta))$.

- 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

- 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).

- 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 [group] [style] [arguments]

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

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

  midpush [float magnitude]
  Adds a force of given magnitude pushing molecules towards the center of the simulation box in the highest-dimension direction (z in 3D, y in 2D, etc). 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.

- grid_freq [integer]

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

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

  Define a group with a given name (first string) defined according to the style. Currently only supports "type" based groups, which are defined with style "type" and should be followed by a single integer to define the atom type in the group.

- integrator [string]

  Integration scheme to employ; current options are EM (Euler-Maruyama) or GJF (Grønbech-Jensen and Farago)

- log_freq [integer]

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

- max_steps [integer]

  Total number of time steps

- n_gaussians [integer]

  Number of Gaussian non-bonded interaction pairs, n_G. This line **must** be immediately followed by n_G lines of the form:

  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
  $u_G(r_{ij}) = \frac{A}{(2\pi\sigma^2)^{\mathbb{D}/2}}\, e^{-|r_{ij}|^2/2\sigma^2}$

  Employs the `ramp` keyword.

- n_erfs [integer n_{erf}]

  Number of Erf non-bonded interaction pairs, n_{erf}. This line **must** be immediately followed by n_{erf} lines of the form:

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
$u(r) = A\ erf\left(\frac{|\mathbf{r}|-R_p}{\sigma}\right) * erf\left(\frac{|\mathbf{r}|-R_p}{\sigma}\right)$
Employs the `ramp` keyword

- n_gaussian_erfs [integer $n_{\text{G-erf}}$]

  Number of Gaussian-erf cross potential interaction pairs, $n_{\text{G-erf}}$. This line **must** be immediately followed by $n_{\text{G-erf}}$ lines of the form:

  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
  $u(r) = A\frac{1}{(2\pi\sigma^2)^{\mathbb{D}/2}}\,e^{-|r|^2/2\sigma^2} * erf\left(\frac{|\mathbf{r}|-R_p}{\sigma}\right)$
  Employs the `ramp` keyword

- n_fieldphases [integer n_f]

  Number of external applied fields with which the particles should interact, see section 1.4 above. This line **must** be immediately followed by n_f lines of the form:

  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

- 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.

- 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]

  Name of the resume file, assumed to be a lammpstrj frame format.

- threads [integer]

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

- traj_freq =[integer]

  Frequency for writing to lammpstrj text file. Use sparingly.

2.2 Creating new input commands

In addition to the variables that need to be defined, there are three places in read_input.cu that must be modified to create new commands read in “read_input()”.

1. Any default values need to be set in the set_defaults subroutine.
2. The code to parse the relevant line has to be in the main part of read input.
3. The right parts need to be added to write_runtime_parameters routine.

3 Creating a new potential

1. Assuming a normal pairstyle, one begins by creating a subclass of PairStyle with the desired name. The first real piece of code to write is the functional form of the potential and getting that on the device. After calling the subroutine `Initialize` which calls +Initialize_Potential+ (or whatever it gets named), the variables d_u, d_f, d_u_k, and d_f_k should all be defined. Both the real and Fourier space components are needed for convenient debugging (d_u) and required for proper setup of the virial (d_f). Calculations of the force and energy each use the Fourier-space versions.

   The initialize routine you write should begin with a call to the inherited function `Initialize\_PairSty` and end with a call to `Initialize\_Virial();`

   pair_style_gaussian.cu and .h are good examples to follow.

2. Implement the input file reading and writing, including writing the runtime parameters in read_input.cu. This generally requires defining a `std::vector` of your new class. The parameters of your pairstyle should be associated with either the `PairStyle` class or your new class, depending on how applicable it is for other potentials.

3. Implement the energy and virial calls in calc_properties.cu. Edits should be made to the subroutines calc_nbVirial, calc_nbEnergy.

   Separately, the forces should be called in forces.cu.

4. In main.cu, add the new potential's energy values to the output streams.

5. Test that it is behaving as expected.

4 Utilities

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

4.1 utils/py-gen-conf/

This contains a bunch of python routines to generate configurations. It works for generic linear architectures by editing the header in either main.py or linear-polymers.py. While commented, these routines are not meant to be as easily plug-and-play as the main source code, but rather to serve as a starting point to make a configuration.

4.2 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 actualy 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.

4.3 utils/gen-configs [deprecated]

The configuration generation utilities are not well maintained and could stand to be made into python and more uniform, but for now:
random_config.cpp must be edited in source and can be used to generate configurations of diblocks.

random_blend.cpp must be edited in source and can generate binary blends.

diblock_solvent.cpp must be edited in source, can create an AB diblock + a C solvent.

5 To-Do List

In no particular order of priority or difficulty.

1. Cosine harmonic angle potential
2. LC interactions via Maier-Saupe potential
3. Rigid body dynamics, requires figuring out quaternions...
4. DPD integrator
5. Create bond/polymerization functionality
6. Other bonded potentials (e.g., FENE) potentials.

Completed To-Do items:

1. Implement charges, which will require reading a new input format. This will necessitate an atom style flag of some sort.
2. ~~GJF Algorithm, requires allocating and storing old positions, noise for each particle.~~
3. ~~Calculate energies on the device, not the host. Will require figuring out how to do a reduction on the device.~~
   Attempted this using the routine in "CUDA reduction example.pdf", values didn't match..
   While this doesn't work on the device, it seems to be of negligible expense to do on the host. Ccalcuating reductions on host is not as terrible as it seemed? Is of negligible expense for the bonds anyway, which are a large loop over number of particles.
4. ~~Implement pressure calculations.~~
   Finished, seems to agree well with mean-field
5. ~~Binary output files and utilities to convert to text format~~
   Done, ~~though particle conversion utility still bugged..~~
6. ~~Ability to gradually ramp parameters (e.g., χ becomes time dependent)~~
   Done
7. ~~Resume functionality~~
   Takes a lammpstrj frame as a resume file
8. ~~Smooth, bare spherical nanoparticles~~
   Done
9. ~~Be able to replace cross-interactions with a static field to seed different microstructures. Keep particle-based self interactions a la κ potential, but cross-interactions are replaced with a static field that has the shape needed. Will need to figure out the basis functions required to make this work...~~
   Implemented, except for sphere phases

10. ~~Calculate $S(k)$ for a given species (maybe force this as a post-proc step)~~
    Implemented as post-processing

6 Scaling and Performance

More data needed, with figures.

Current version ran $nN = 175k$ particles on a 45^3 grid to 250k iterations in 25 minutes on my laptop, which runs an 8 GB NVIDIA GTX 1070. 10% of that time was spent communicating with the host and calculating the energy on the host, so there is still yet room for improvement.

Second calculation ran a system with 648k particles on a 63^3 grid at a density of 3.0 for 250k steps in 89 minutes.