README.md - Grip

Introduction

Concentration dynamics

The main field that evolves is relative concentration of the A phase at the mesoscale of a block copolymer, which occurs through a modified 2D Cahn Hilliard equation:

$$\frac{\partial c}{\partial t} = -\epsilon^2 \nabla^4(c) + \nabla^2(uc^3 - bc) - \sigma(c - m) + \sigma_\eta \eta$$

Domain: 2D rectangular

Boundary Conditions: periodic, Neumann, Dirichlet, or mixed

Polymer properties

If desired, some of the coefficients of this equation may be chosen to be a function of polymer properties (see, e.g., `Choksi & Ren, 2003`):

$$\epsilon^2 = \frac{l^2}{3(\frac{1-m}{2})(\frac{1+m}{2})\chi|\Omega|^{2/3}}$$

$$\sigma = \frac{36|\Omega|^{2/3}}{(\frac{1-m}{2})^2(\frac{1+m}{2})^2 l^2 \chi N^2}$$

Thermal dependence

Given that temperature is usually specified in experiment as a control, and that Flory-Huggins is inversely proportional to it, the user may optionally specify a temperature field and Flory-Huggins will be calculated as:

$$\chi(T) = \left(\frac{\chi_{max} - \chi_{min}}{\frac{1}{T_{min}} - \frac{1}{T_{max}}} \right) \left(\frac{1}{T} - \frac{1}{T_{max}} \right) + \chi_{min}$$

Thermal dynamics

If desired, temperature can be evolved as a spatial field through a thermal diffusion equation that is one-way coupled to the Cahn-Hilliard dynamics:

$$\frac{\partial T}{\partial t} = D_T \nabla^2 (T) + f$$

Boundary conditions: Neumann

Numerical method

Spatial discretization is uniform finite difference. Temporal evolution is handled by the `boost::numeric::odeint` library. A variety of explicit marching options are possible (e.g., RK4, with or without adaptive steps). The explicit timestep is set with reference to the biharmonic timescale of the problem (which should be the stiffest linear timescale present).

Building

The `cpp/` subdirectory contains all C++ source (`cpp/src/`) as well as Python Swig wrappers (`cpp/swig/`).

Building is done through `cmake`, using the included `cpp/CMakeLists.txt` as follows:

```
cd [/PATH/TO]/cahnhilliard_2d/cpp
mkdir build/
cd build/
cmake ../
make
```

This should (1) build all C++ source into the static library `libch_src.a` that one

can link any client code against, (2) build an example C++ driver into the executable `ch2d`, and (3) build Python wrappers to the code using Swig, and store them in `cpp/swig`. To test, the C++ executable can be run in the usual way:

```
./ch2d
```

The directory `cpp/swig/` provides example drivers demonstrating how to use the Swig Python wrappers. For example:

```
cd [/PATH/TO]/cahnhilliard_2d/cpp/swig
python driver.py
```

User Interface

The user of this code interfaces with the solver source through the following classes/functions:

- `SimInfo`: specifies general simulation properties (e.g., grid size, BC type)
- `CHparamsScalar`: specifies physical parameters, all of which are scalars
- `CHparamsVector`: specifies physical parameters, which can be field quantities
- `run_ch_solver( SimInfo& , CHparams[Scalar/Vector]& )`: function used to run the simulation

Please consult both the source code and example driver programs for more details on usage.

Visualization

Some lightweight Python scripts are provided in `visualization/` for convenience. You will have to edit the necessary options/filepaths to be consistent with the state files you are trying to read/display.

Parallel Scaling

The solver is parallelized using shared memory with OpenMP. Here are some small scaling studies, using no thermal behavior and constant CH coefficients. In what follows:

- `nx` = spatial resolution
- `epsilon` = value of biharmonic coefficient
- `n_dtbiharm` = temporal length of total simulation, referenced to the length of the biharmonic timescale
- `n_core` = number of CPU cores
- `time (sec)` = total execution time, in seconds

BloodMeridian (6 cores, Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz)

Pure C++

nx	epsilon	n_dtbiharm	n_core	time (sec)
128	0.01	300	1	305
128	0.01	300	3	41
128	0.01	300	6	36

C++/Swig

nx	epsilon	n_dtbiharm	n_core	time (sec)
128	0.01	300	1	255
128	0.01	300	3	54
128	0.01	300	6	51

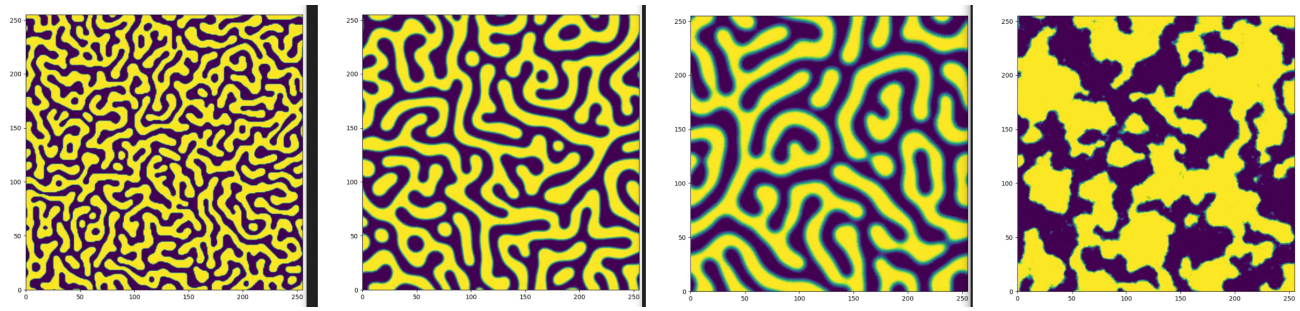
HPC1 (8 cores, Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz)

Pure C++

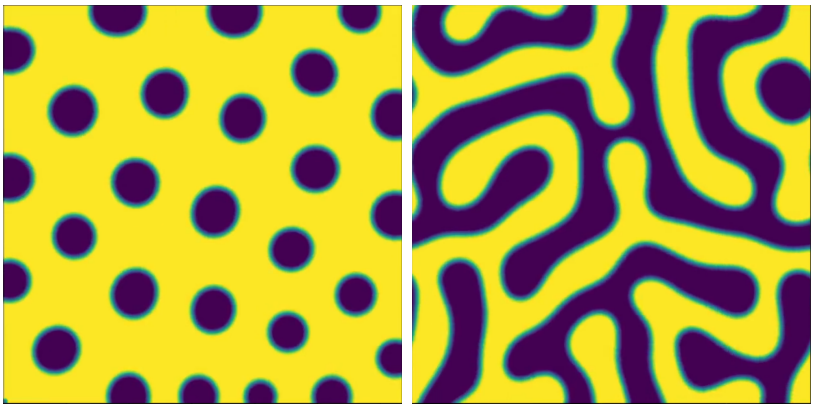
nx	epsilon	n_dtbiharm	n_core	time (sec)
128	0.01	300	1	309
128	0.01	300	4	52
128	0.01	300	8	34

Example Output

Here are some example state snapshots from 4 different simulations. The first three show the effect of increasing the biharmonic coefficient. The last snapshot is taken from the same system as that which produced the second snapshot, but with more noise added to the dynamics.



These are two steady-states achieved with differing values of `m` (the left has `m = 0.5`; the right has `m = 0`):



This is a temperature-dependent simulation with thermal diffusion present (the top is the concentration field; the bottom is the temperature field):