# Implementation of Cahn-Hilliard phase-field model coupled with elasticity

prepared by
Juan Michael Sargado

## 1 Derivation of governing equations

We consider a binary phase decomposition alloy system with a miscibility gap, whose composition $c'$ has two equilibrium phases, a matrix phase $c'_m$ and a precipitated phase $c'_p$. For convenience, we rescale the composition variable $c\left(\mathbf{x}, t\right)$ as

$$c = \frac{c' - c'_m}{c'_p - c'_m} \tag{1}$$

so that $c$ represents the equilibrium volume fraction of the precipitated phase. For such a system, the total free energy $F$ can be assumed to consist of chemical as well as elastic contributions, i.e.

$$F = F^{\text{ch}} + F^{\text{el}}. \tag{2}$$

The chemical contribution to the free energy may be written as

$$F^{\text{ch}} = N_v \int_\Omega \left[ f\left(c\right) + \kappa \nabla c \cdot \nabla c \right] \mathrm{d}\Omega, \tag{3}$$

$N_v$ denote the number of atoms per unit volume, $f\left(c\right) = Ac^2\left(1-c\right)^2$ is the standard Ginzburg-Landau double well potential with minima at 0 and 1, while $\kappa$ is a parameter that is assumed to be constant for a regular solution, and $A$ is a parameter that controls the height of the energy barrier between the two phases.

Meanwhile, the elastic contribution to the free energy is given by

$$F^{\text{el}} = N_v \int_\Omega \frac{1}{2} \sigma^{\text{el}}_{ij} \varepsilon^{\text{el}}_{ij} \, \mathrm{d}\Omega \tag{4}$$

in which $\varepsilon_{ij} = \dfrac{1}{2}\left(\dfrac{\partial u_i}{\partial x_j} + \dfrac{\partial u_j}{\partial x_i}\right)$ is the standard infinitesimal strain tensor, and

$$\varepsilon^{\text{el}}_{ij} = \varepsilon_{ij} - \beta\left(c\right)\bar{\varepsilon}\delta_{ij} \tag{5}$$

wherein the term $\beta\left(c\right)\bar{\varepsilon}\delta_{ij}$ is a spherical misfit strain. Here $\bar{\varepsilon}$ is assumed to be constant, and $\beta\left(c\right)$ is given by

$$\beta\left(c\right) = c^3\left(10 - 15c + 6c^2\right). \tag{6}$$

It is assumed that both matrix and precipitated phases may be described by linear elasticity, thus

$$\sigma^{\text{el}}_{ij} = C_{ijkl}\,\varepsilon^{\text{el}}_{kl} \tag{7}$$

where $C_{ijkl}$ is given by

$$C_{ijkl} = C^m_{ijkl} + \alpha\left(c\right)\left(C^p_{ijkl} - C^m_{ijkl}\right) \tag{8}$$

where in the reference paper (and adjusting for the definition of $C_{ijkl}$ given above), $\alpha(c) = \beta(c)$. Then, the elastic energy can be expressed as

$$F^{\text{el}} = N_v \int_\Omega \frac{1}{2} \left[\varepsilon_{ij} - \beta(c)\,\bar{\varepsilon}\delta_{ij}\right] \left[C^m_{ijkl} + \alpha(c)\left(C^p_{ijkl} - C^m_{ijkl}\right)\right] \left[\varepsilon_{kl} - \beta(c)\,\bar{\varepsilon}\delta_{kl}\right] \mathrm{d}\Omega. \tag{9}$$

For convenience, let $\Delta C_{ijkl} = C^p_{ijkl} - C^m_{ijkl}$. Then we can write $F^{\text{el}}$ more compactly as

$$F^{\text{el}} = N_v \int_\Omega \frac{1}{2} \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha(c)\,\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] \mathrm{d}\Omega \tag{10}$$

Then the variational derivative of the free energy with respect to $c$ is given by

$$\frac{\delta F}{\delta c} = \frac{\delta F^{\text{ch}}}{\delta c} + \frac{\delta F^{\text{el}}}{\delta c} \tag{11}$$

where

$$\frac{\delta F^{\text{ch}}}{\delta c} = N_v \left[A\left(4c^3 - 6c^2 + 2c\right) - 2\kappa\nabla^2 c\right] \tag{12}$$

and

$$\begin{aligned} \frac{\delta F^{\text{el}}}{\delta c} = \frac{N_v}{2} \{ & -\beta'(c)\,\bar{\varepsilon}\mathbf{I} : \left[\mathbb{C}^m + \alpha(c)\,\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] \\ & + \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'(c)\,\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] \\ & - \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha(c)\,\Delta\mathbb{C}\right] : \beta'(c)\,\bar{\varepsilon}\mathbf{I} \} \end{aligned} \tag{13}$$

Due to major symmetry of the elasticity tensor, we can simplify the 2nd result above to

$$\begin{aligned} \frac{\delta F^{\text{el}}}{\delta c} = \frac{N_v}{2} \{ & \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'(c)\,\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] \\ & -2\left[\boldsymbol{\varepsilon} - \beta(c)\,\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha(c)\,\Delta\mathbb{C}\right] : \beta'(c)\,\bar{\varepsilon}\mathbf{I} \} \end{aligned} \tag{14}$$

Assuming that the mobility $M$ is constant, the coupled system can be expressed as

$$\begin{cases} \dfrac{\partial c}{\partial t} = M\nabla^2 \left[AN_v\left(4c^3 - 6c^2 + 2c\right) - 2N_v\kappa\nabla^2 c + \dfrac{\delta F^{\text{el}}}{\delta c}\right] & \text{(15a)} \\ \nabla \cdot \boldsymbol{\sigma}^{\text{el}} = \mathbf{0} & \text{(15b)} \end{cases}$$

where we have assumed zero body forces for the stress equilibrium equation. In order to reduce the differentiability requirements on $c$, we define an auxiliary variable $\Psi$:

$$\Psi = AN_v\left(4c^3 - 6c^2 + 2c\right) - 2N_v\kappa\nabla^2 c + \frac{\delta F^{\text{el}}}{\delta c} \tag{16}$$

Then the modified system becomes

$$\begin{cases} \dfrac{\partial c}{\partial t} - M\nabla^2\Psi = 0 & \text{(17a)} \\ AN_v\left(4c^3 - 6c^2 + 2c\right) - \Psi + \dfrac{\delta F^{\text{el}}}{\delta c} - 2N_v\kappa\nabla^2 c = 0 & \text{(17b)} \\ \nabla \cdot \boldsymbol{\sigma}^{\text{el}} = \mathbf{0} & \text{(17c)} \end{cases}$$

# 2 Numerical approximation

To solve the above system, we employ a combined-discretization framework, wherein linear finite elements are used to approximate the weak form of the stress equilibrium equation, while the PDEs governing the evolution of $c$ and $\Psi$ are approximated using a cell-centered finite volume framework, using control volumes made up of simplex elements (triangles in 2D).

Since $\bar{\varepsilon}$ is assumed to be constant, $\delta\boldsymbol{\varepsilon}^{\mathrm{el}} = \delta\boldsymbol{\varepsilon}$ and the weak form of the stress equilibrium equation is given by

$$\int_{\Omega} \boldsymbol{\sigma}^{\mathrm{el}} : \delta\boldsymbol{\varepsilon}\, \mathrm{d}\Omega = \int_{\partial\Omega^N} \mathbf{t} \cdot \delta\mathbf{u}\, \mathrm{d}\partial\Omega. \tag{18}$$

In the finite element method utilizing standard Lagrange elements, the displacement vector is interpolated within each element $\Omega^e$ from its nodal values by means of shape functions, i.e. $\mathbf{u} = \{u, v\}^{\mathrm{T}} = [\mathbf{N}^e]\,\{\mathbf{u}^e\}$ where for a 3-node element,

$$[\mathbf{N}^e] = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & N_3^e & 0 \\ 0 & N_1^e & 0 & N_2^e & 0 & N_3^e \end{bmatrix} \qquad \{\hat{\mathbf{u}}\} = \begin{Bmatrix} u_1^e & v_1^e & u_2^e & v_2^e & u_3^e & v_3^e \end{Bmatrix}^{\mathrm{T}} \tag{19}$$

in which $N_i$ denote the FE shape function at node $i$, while $u_i$ and $v_i$ are respectively the $x$- and $y$-components of displacement at node $i$. In the current implementation, we assume 2D conditions so that the strain tensor may be expressed in Voigt form as

$$\{\boldsymbol{\varepsilon}\} = \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \gamma_{12} \end{Bmatrix} = [\mathbf{B}^e]\,\{\mathbf{u}^e\} \qquad [\mathbf{B}^e] = \begin{bmatrix} \dfrac{\partial N_1^e}{\partial x} & 0 & \dfrac{\partial N_2^e}{\partial x} & 0 & \dfrac{\partial N_3^e}{\partial x} & 0 \\ 0 & \dfrac{\partial N_1^e}{\partial y} & 0 & \dfrac{\partial N_2^e}{\partial y} & 0 & \dfrac{\partial N_3^e}{\partial y} \\ \dfrac{\partial N_1^e}{\partial y} & \dfrac{\partial N_1^e}{\partial x} & \dfrac{\partial N_2^e}{\partial y} & \dfrac{\partial N_2^e}{\partial x} & \dfrac{\partial N_3^e}{\partial y} & \dfrac{\partial N_3^e}{\partial x} \end{bmatrix}. \tag{20}$$

The weak form of the equilibrium equation within each element can then be written as

$$\int_{\Omega^e} \{\delta\mathbf{u}^e\}^{\mathrm{T}}\,[\mathbf{B}^e]^{\mathrm{T}}\left\{\boldsymbol{\sigma}^{\mathrm{el}}\right\}\,\mathrm{d}\Omega = \int_{\partial\Omega^e \cap \partial\Omega^N} \{\delta\mathbf{u}^e\}^{\mathrm{T}}\,[\mathbf{N}^e]^{\mathrm{T}}\,\{\mathbf{t}\}\,\mathrm{d}\Omega, \tag{21}$$

and as the above must hold for arbitrary $\delta\mathbf{u}$, this requires

$$\int_{\Omega^e} [\mathbf{B}^e]^{\mathrm{T}}\left\{\boldsymbol{\sigma}^{\mathrm{el}}\right\}\,\mathrm{d}\Omega = \int_{\partial\Omega^e \cap \partial\Omega^N} [\mathbf{N}^e]^{\mathrm{T}}\,\{\mathbf{t}\}\,\mathrm{d}\Omega. \tag{22}$$

Substituting in the expression for $\boldsymbol{\sigma}^{\mathrm{el}}$, we obtain

$$\int_{\Omega^e} [\mathbf{B}^e]^{\mathrm{T}}\left([\mathbf{C}^m] + \alpha\,(c)\,[\Delta\mathbf{C}]\right)\left([\mathbf{B}^e]\,\{\mathbf{u}^e\} - \beta\,(c)\,\bar{\varepsilon}\,\{\mathbf{I}\}\right)\mathrm{d}\Omega = \int_{\partial\Omega^e \cap \partial\Omega^N} [\mathbf{N}^e]^{\mathrm{T}}\,\{\mathbf{t}\}\,\mathrm{d}\Omega. \tag{23}$$

where $\{\mathbf{I}\} = \{1, 1, 0\}^{\mathrm{T}}$ is the 2nd order identity tensor in Voigt form. Note that for fixed $c$, the above equation is linear in $\mathbf{u}$. However, in the current implementation, we treat it as nonlinear in order to make use of the available solution methods in BROOMStyx.

The Cahn-Hilliard part of the system can be discretized using a cell-centered finite volume method utilizing the 2-point flux approximation scheme to model the normal gradients across control volume boundaries. First, recast the relevant equations in conservation form:

$$\begin{cases} \dfrac{1}{M}\dfrac{\partial c}{\partial t} - \nabla\cdot\nabla\Psi = 0 & \text{(24a)} \\ A\left(2c^3 - 3c^2 + c\right) - \dfrac{\Psi}{2N_v\kappa} + \dfrac{1}{2N_v\kappa}\dfrac{\delta F^{\mathrm{el}}}{\delta c} - \nabla\cdot\nabla c = 0 & \text{(24b)} \end{cases}$$

Integrating over the above equations over a control volume $\Omega^e$ and making use of the Gauss Divergence Theorem, we obtain

$$\begin{cases} \dfrac{1}{M}\displaystyle\int_{\Omega^e} \frac{\partial c}{\partial t} \mathrm{d}\Omega - \oint_{\partial\Omega^e} \nabla\Psi \cdot \mathbf{n}\, \mathrm{d}S = 0 & \text{(25a)} \\ \displaystyle\int_{\Omega^e} \left[ A\left(2c^3 - 3c^2 + c\right) + \frac{1}{2N_v\kappa}\left(\frac{\delta F^{\mathrm{el}}}{\delta c} - \Psi\right)\right] \mathrm{d}\Omega - \oint_{\partial\Omega^e} \nabla c \cdot \mathbf{n}\, \mathrm{d}S = 0. & \text{(25b)} \end{cases}$$

The primary variables $c$ and $\Psi$ are assumed to be piecewise constant over each control volume $\Omega^e$; as mentioned previously, the gradient terms are evaluated using the TPFA scheme. That is, for the $K$-th control volume,

$$\oint_{\partial\Omega_K^e} \nabla c \cdot \mathbf{n}\, \mathrm{d}S = \sum_{i=1}^{3} T_K^i \left(c_K^i - c_K\right) \tag{26}$$

where the transmissibility coefficient $M_K^i$ is given by

$$T_K^i = \frac{L_K^i}{d_K^i + \tilde{d}_K^i}. \tag{27}$$

Here, $d_K^i$ denotes the distance from the center of cell $K$ to the midpoint of edge $i$, $\tilde{d}_K^i$ is the distance from the center of the adjacent cell sharing edge $i$ with cell $K$ to the midpoint of edge $i$, and $L_K^i$ is the length of edge $i$. The same transmissibility coefficients are used to evaluate the surface integrals involving $\Psi$.

Meanwhile, we utilize a backward Euler scheme to discretize the time dertivative. Thus at time $t_n$,

$$\frac{\partial c}{\partial t} = \frac{c^n - c^{n-1}}{t^n - t^{n-1}} \tag{28}$$

where $c^{n-1}$ is the converged value of the concentration at the previous time step, and $c = c^n$ is the unknown value to be determined at the current time step. The backward Euler nature of the time integration is realized by taking all the instances of $c$ in the coupled system to be $c^n$ instead of $c^{n-1}$.

We can now summarize the discrete equations associated with each control volume/element $\Omega_K$. Writing these equations as residuals, we have

$$\begin{cases} r_K^{\Psi} = A_K^e \dfrac{M}{\Delta t}\left(c_K - c_K^{n-1}\right) + \displaystyle\sum_{i=1}^{3} T_K^i \left(\Psi_K - \Psi_K^i\right) & \text{(29a)} \\ r_K^{c} = A_K^e \left[ A\left(2c_K^3 - 3c_K^2 + c_K\right) + \dfrac{1}{2N_v\kappa}\left(\dfrac{\delta F^{\mathrm{el}}}{\delta c} - \Psi\right)\right] + \displaystyle\sum_{i=1}^{3} T_K^i \left(c_K - c_K^i\right) & \text{(29b)} \\ \mathbf{r}^{\mathbf{u}} = \displaystyle\int_{\Omega^e} \left[\mathbf{B}^e\right]^{\mathrm{T}} \left(\left[\mathbf{C}^m\right] + \alpha\left(c_K\right)\left[\Delta\mathbf{C}\right]\right)\left(\left[\mathbf{B}^e\right]\left\{\mathbf{u}^e\right\} - \beta\left(c_K\right)\bar{\varepsilon}\left\{\mathbf{I}\right\}\right) \mathrm{d}\Omega - \displaystyle\int_{\partial\Omega^e \cap \partial\Omega^N} \left[\mathbf{N}^e\right]^{\mathrm{T}} \left\{\mathbf{t}\right\} \mathrm{d}\Omega & \text{(29c)} \end{cases}$$

where we have incorporated the fact that a piecewise linear approximation of the displacement field results in constant strains throughout the element.

# 3 Linearization and staggered solution

The coupled system above is nonlinear, and in particular the free energy functional is non-convex. We adopt a splitting strategy, wherein we minimize $r^c$ and $r^{\Psi}$ together, alternating with $\mathbf{r}^{\mathbf{u}}$ in a staggered

scheme. Within each subsystem, we perform the actual minimization iteratively via the Newton-Raphson algorithm, wherein the next iterate is obtained from the previous one through the formula

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[\mathbf{J}\left(\mathbf{x}_n\right)\right]^{-1} \mathbf{r}\left(\mathbf{x}_n\right) \tag{30}$$

The relevant components of the Jacobian matrix are as follows:

$$\frac{\partial r_K^{\Psi}}{\partial c_K} = A_K^e \frac{M}{\Delta t} \tag{31}$$

$$\frac{\partial r_K^{\Psi}}{\partial \Psi_K} = \sum_{i=1}^{3} T_K^i \tag{32}$$

$$\frac{\partial r_K^{\Psi}}{\partial \Psi_K^i} = -T_K^i \tag{33}$$

$$\frac{\partial r_K^{c}}{\partial c_K} = A_K^e \left[ A\left(6c_K^2 - 6c_K + 1\right) + \frac{1}{2N_v\kappa} \frac{\partial}{\partial c}\left(\frac{\delta F^{\mathrm{el}}}{\delta c}\right)\bigg|_{c=c_K} \right] + \sum_{i=1}^{3} T_K^i \tag{34}$$

$$\frac{\partial r_K^{c}}{\partial c_K^i} = -T_K^i \tag{35}$$

$$\frac{\partial r_K^{c}}{\partial \Psi_K} = -\frac{A_K^e}{2N_v\kappa} \tag{36}$$

$$\frac{\partial \mathbf{r}^{\mathbf{u}}}{\partial \mathbf{u}^{\mathbf{e}}} = \int_{\Omega^e} \left[\mathbf{B}^e\right]^{\mathrm{T}} \left(\left[\mathbf{C}^m\right] + \alpha\left(c_K\right)\left[\Delta\mathbf{C}\right]\right)\left[\mathbf{B}^e\right] \mathrm{d}\Omega \tag{37}$$

where

$$\begin{aligned}
\frac{\partial}{\partial c}\left(\frac{\delta F^{\mathrm{el}}}{\delta c}\right)\bigg|_{c=c_K} = \frac{N_v}{2} \{ & \left[-\beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'\left(c_K\right)\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] \\
& + \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha''\left(c_K\right)\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] \\
& + \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'\left(c_K\right)\Delta\mathbb{C}\right] : \left[-\beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] \\
& - 2\left[-\beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha\left(c_K\right)\Delta\mathbb{C}\right] : \beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I} \\
& - 2\left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'\left(c_K\right)\Delta\mathbb{C}\right] : \beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I} \\
& -2\left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha\left(c_K\right)\Delta\mathbb{C}\right] : \beta''\left(c_K\right)\bar{\varepsilon}\mathbf{I}\}
\end{aligned} \tag{38}$$

which simplifies to

$$\begin{aligned}
\frac{\partial}{\partial c}\left(\frac{\delta F^{\mathrm{el}}}{\delta c}\right)\bigg|_{c=c_K} = \frac{N_v}{2} \{ & \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha''\left(c_K\right)\Delta\mathbb{C}\right] : \left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] \\
& - 4\left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\alpha'\left(c_K\right)\Delta\mathbb{C}\right] : \left[\beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] \\
& + 2\left[\beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha\left(c_K\right)\Delta\mathbb{C}\right] : \beta'\left(c_K\right)\bar{\varepsilon}\mathbf{I} \\
& -2\left[\boldsymbol{\varepsilon} - \beta\left(c_K\right)\bar{\varepsilon}\mathbf{I}\right] : \left[\mathbb{C}^m + \alpha\left(c_K\right)\Delta\mathbb{C}\right] : \beta''\left(c_K\right)\bar{\varepsilon}\mathbf{I}\}
\end{aligned} \tag{39}$$

# 4 Implementation in the BROOMStyx code

I have chosen to implement the above problem in a new branch of the software framework BROOMStyx. The link for the repository is `https://github.com/jmsargado/broomstyx`. The code runs on Linux (my operating system is Mint 22.1) and requires the `gcc` and g++ compilers, as well as `make` and `cmake`. The code also utilizes BLAS/LAPACK routines as well as the PARDISO solver from Intel MKL. You will also need to have `git` installed to navigate between the different branches of the code.

## 4.1 Code compilation and running simulations

The steps for code compilation are as follows:

1. Install Intel MKL. This can be done by following the instructions provided at this address: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-download.html?operatingsystem=linux&linux-install=apt`

2. Clone the BROOMStyx repository from `https://github.com/jmsargado/broomstyx`, and switch from the master branch to the one called `cahn_hilliard_elastic`. This can be done with the command

   ```
   git checkout -b cahn_hilliard_elastic origin cahn_hilliard_elastic
   ```

3. Clone the repository with the run files to another directory. In this new directory, you will find a `UserFunctions` folder.

4. Go back to the directory where you have cloned BROOMStyx, and make a copy of the file `runCMake.sh_example`. Name this new file `runCMake.sh`, and open it in a text editor. In the line

   ```
   export MKL_ROOT=<PATH_TO_MKL_INSTALLATION>
   ```

   replace the right side with the path where Intel MKL was installed on your machine. The default path is usually

   Next, comment out the line that starts with export VIENNACL_ROOT, since we won't be using it. Finally, in the line

   ```
   export USER_SOURCE_DIR=<PATH_TO_USER_SOURCE_DIRECTORY>
   ```

   replace the right side with the complete path of the `UserFunctions` folder that was mentioned in the previous step.

5. Open a terminal in the top level directory of the BROOMStyx repository, and execute the command

   ```
   bash run_CMake.sh
   ```

   This will output some messages regarding C compilers, CUDA, etc. Check that `USER_SOURCE_DIR` is correct, that MKL has been found, and that OpenMP is enabled.

6. Type

   ```
   cd build
   ```

   into the terminal and press enter; this brings you into the newly created `build` directory that has all the configuration files needed for code compilation. To compile the code, we invoke `make`, for example

   ```
   make -j15
   ```

The argument `-j15` signifies that `make` should use 15 threads to compile the code. More threads means faster compilation, but the amount should not exceed the actual number that is available in your machine.

7. For ease of running the executable, you can create a symbolic link:

```
sudo ln -s <BROOMSTYX_DIRECTORY>/build/broomstyx usr/bin/broomstyx
```

where `<BROOMSTYX_DIRECTORY>` is the full path to the directory in which you cloned the repository for BROOMStyx.

8. In the directory where you have cloned the run files, navigate to the MainSimulation folder. There should be at one input file in there, which has the file extension `.inp`, for example `CahnHilliard.inp`. To run a simulation using this input file, type

broomsyx CahnHilliard

in the terminal and press enter (you must omit the file extension when typing in the input file). The total runtime will depend on the number of cores in your machine as well the processor clock speed. **Note:** Make sure to modify the number of threads used by the linear solver to the actual number of threads you have in your machine. The relevant lines in the input file are

```
LinearSolver MKL_Pardiso nThreads 16 Unsymmetric
```

Change the number after `nThreads` to the actual number of hyperthreads that is available in your machine. Note that if you run the code in a machine that has multiple sockets/processors (or multiple NUMA nodes), the runtime maybe faster if your restrict the threads to a single NUMA node, depending on how slow the non-uniform memory access is when one processor accesses RAM that is connected to another processor.

## 4.2 Source files for the Cahn-Hilliard problem

BROOMStyx is a somewhat big code which implements various schemes for simulating different phenomena. For instance, the staggered nonlinear scheme that is used to solve the coupled system of equations is already available previously in the code and does not need to be re-implemented. The actual Cahn-Hilliard problem implementation can be found in the following class files in the `src/` directory:

- **Numerics/Mechanics/MicrostructureEvolution/CahnHilliard_Elas_FeFv_Tri3.hpp (.cpp)** – this implements the governing equations for the coupled Cahn-Hilliard equation with elasticity discretized using a combined linear FE and cell-centered FV framework, including the calculation of left and right hand sides, and local coefficient (tangent) matrices, both for static and transient terms.
- **Materials/Mechanics/CubicElasticity.hpp (.cpp)** – implements the constitutive law for cubic elasticity.
- **Materials/InterpolationFunctions/Interpolation_0_1.hpp (.cpp)** – implements the interpolating function used for $\alpha(c)$ and $\beta(c)$, including its derivatives.

In the `UserFunctions` folder found with the input files for running the simulations, we also find two classes used to implement initial conditions, which are used for the main simulation and also for tests:

- **RandomFunction.hpp (.cpp)** – implements a function that gives uniformly distributed values between a specified minimum and maximum, according to a user-chosed random seed. In the main simulation, we use it to initialize the concentration to a uniformly distribution with values between 0.49 and 0.51. The perturbations are necessary since a constant initial value of $c$ will not trigger spinodal decomposition.

- **LinearFunction.hpp (.cpp)** – implements a linear function with user-supplied coefficients, i.e. $f(x) = A + Bx + Cy$. It is used to initialize the concentration to a linear function during tests, in order to observe both nucleation/growth and spinodal decomposition. Note that the function does not check that the concentration is within bounds, i.e. $c \in [0, 1]$ so it is the user's reponsibility to supply meaningful coefficients.

# 5 Note on results

The code is able to model both spinodal decomposition in the unstable regime as well as nucleation in the metastable regime, however we can observe that the cuboid nature of the precipitates is somewhat suppressed. This might be due to the discrete formulation used, i.e. the finite volume scheme is very fast in terms of execution but seemingly struggles with anisotropy, as the TPFA scheme only considers normal gradients/fluxes through a cell face rather than the full gradient. In hindsight, it might have been better to use a pure finite element formulation based on 4-node quadrilaterals and bi-linear shape functions. However the latter implies possibly a substantial longer runtime since Q4 elements require 4 Gauss points (as opposed to one for linear FEM), and the size of local matrices is larger. For the current implementation, the total runtime was around 10 hours for 10,000 time steps on a machine equipped with a processor having 8 cores (16 threads) and base frequency of 3.0 GHz.

# 6 Tests

We conduct several tests to verify correct execution of the code, summarized as follows:

1. **Test of stress equilibrium equation**
   The run files for these tests can be found in the folder `/TESTS/ElasticityTest`. There are several input files, based on a domain consisting of a unit square and displacement boundary conditions such that the resulting uniform strain tensor is either $\boldsymbol{\varepsilon} = \{1, 0, 0\}^{\mathrm{T}}$, $\boldsymbol{\varepsilon} = \{0, 1, 0\}^{\mathrm{T}}$, or $\boldsymbol{\varepsilon} = \{0, 0, 1\}^{\mathrm{T}}$. This allows us to check that the stresses much the input values for $C_{11}, C_{12}$ and $C_{44}$. Only the stress equilibrium equations are solved, hence the concentration does not evolve away from its initial condition. By initializing the concentration to either 0 or 1, the stress should match the input values of the elasticity tensor components corresponding to either the matrix or the precipitate.

   To see the results of the these tests, open a terminal in the test folder (`/TESTS/ElasticityTest`) and run broomstyx on one input file at a time. After each run, open the resulting `.vtk` files (found in the `OutputParaview` folder) in `Paraview` or a similar program, and compare the resulting value of the stress components to the input parameters for the relevant elasticity tensor.

2. **Test of Cahn-Hilliard implementation**
   The run files for these tests can be found in the folder `/TESTS/CahnHilliardTest`. There are three tests in all:

   (a) Only the phase-field subsystem is solved. This is equivalent to having the system evolve only due to the chemical energy; the elastic component is zero all throughout.

(b) Both the phase-field subsystem as well as the stress equilibrium equations are solved (similar to the main simulation).

(c) Both the phase-field subsystem as well as the stress equilibrium equations are solved, but a higher misfit strain ($\bar{\varepsilon} = 0.02$) is used as input.

The tests are run up to $t = 1000$ with $\Delta t = 1$. In all cases, the concentration is initialized as linear function. This allows us to observe both nucleation in the meta-stable regime and spinodal decomposition in the unstable regime.

The tests may be run, but take some time. For brevity, the results are summarized in the file `Animation_Concentration.avi`, which shows side-by-side the evolution of test (a) at the left, test (b) in the center, and test (c) at the right. We can observe that in the case of tests (b) and (c) the nucleation and spinodal decompositions have preference for more horizontal-vertical evolutions compared to test (a) where only the chemical energy drives the evolution. Furthermore, for test (c), the larger misfit strain results in the unstable regime shifting noticeably to the right, thus spinodal decomposition occurs at higher initial concentrations compared to the case where the misfit strain is smaller.

Interestingly, for tests (b) and (c) we can observe that the nucleations and spinodal decompositions features more cuboidal shapes at early time steps, with corners becoming progressively rounder with time. I believe this is a numerical artefact related to the discretization framework I have adopted; instead it would be more advisable to adopt a quadrilateral mesh where the element edges are aligned with the directions of anisotropy. Such schemes could possibly be Q4-FE / cc-FV, or pure Q4-FE.

# 7 Simplifying assumptions

The main simulation run as well as the tests described above deviate from the reference paper with regard to boundary conditions. Rather than enforcing periodic boundary conditions, zero-flux conditions are imposed at the boundaries for the phase-field subsystem (i.e. $\nabla c \cdot \mathbf{n} = 0$ and $\nabla \Psi \cdot \mathbf{n} = 0$), and normal confinement constraints are imposed for the displacement field, i.e. $\mathbf{u} \cdot \mathbf{n} = 0$ where $\mathbf{n}$ denotes the outward unit normal vector at the boundary.