PX915-A Design Document

Dylan Morgan, Matyas Parrag, Anas Siddiqui, Ben Gosling, and Geraldine Anis

April 27, 2022

General Code Features and Stylistic Guidlines

- The code will be hosted on Github as open source, specifically licensed with the GNU Public License v3.0.
- The NetCDF4, Matplotlib, and f90wrap external libraries will be utilised.
- The code will be written to conform to the Fortran 2008 standard.
- Indentation will be with 2 spaces.
- All the code will be written in lower case.
- The Fortran `.f90` files will be organised as 1 module per file, and the main program in a separate file to these.

Documentation

User and developer documentation will be implemented and provided using the GitHub Pages tool. User documentation will include high-level descriptions of code blocks/functions, while developer documentation will contain a more developer-oriented code description. Both sets of documents will have detailed instructions on obtaining, building, and running the code. This will also be included in the `README.md` file on the GitHub repository. Details of the input parameters and the input file format will be provided, as well as a description of the output format.

Compilation

Users wishing to install and utilise the code will be able to clone it from the GitHub repository using standard git commands. This will be documented in a `README.md` as well as GitHub pages (as mentioned in the Documentation section above). A `Makefile` will be provided in the repository to automate the compilation of the code and run automated unit tests, as well as include specific rules for debugging. Details of the different Makefile options/rules will be included in the documentation. The Makefile will use GNU Fortran for compilation.

User Input

User input will be read from a text file. The text file will be structured be in the following format `VARIABLE = VALUE`. This will then be read in Fortran and the values stored in appropriate variables to be used later by functions. The user will also be able to input a custom polynomial. For this, the `VALUE` will be a comma-separated list of numbers $N, a_0, a_1, a_2, ..., a_{N-1}$, where N will be the number of coefficients and thus $N-1$ will be the order of the polynomial, $\{a_i\}$ which will correspond to the polynomial $\Sigma_i a_i x^i$. The format of the input file is outlined as follows:

$C0$ = double
$Cstd$ = double
Nx = integer ≥ 1
Ny = integer ≥ 1
$M1$ = double
$M2$ = double
k = double
A = double
C_int = integer ≥ -1
CPI = string
CPO = string
$f(c)$ = int ≥ 1, double, ..., double
T = double (Optional)
$Delta\_T$ = double (Optional)
dF_tol = double (Optional)

Where $C0$ is the initial average concentration, $Cstd$ is the standard deviation used for its random distribution, Nx, Ny specify the domain grid size, $M1, M2$ specify the mobility's of the two species, k is the free energy gradient parameter, A is the bulk free energy scaling factor (This is only required when $f(c)$ is not input), C_int it the checkpointing interval (this can be -1 for automatic checkpointing), CPO is the checkpoint output filename, CPI is the checkpoint input filename, and $f(c)$ is a user-defined polynomial for the bulk free energy.

There are also optional inputs if the user wants to specify the period for the spinodal decomposition that is simulated over (T), and the time-step ($Delta\_T$) between each forward iteration. There are also optional inputs for if the user wants to specify the period for the spinodal decomposition that is simulated (T), and the time-step ($Delta\_T$) between each forward iteration. The default time step will be set as per our stability analysis described later in the document. Suppose the user doesn't specify a period over which to simulate. In this case, the forward solver will keep iterating until the total free energy is determined to converge (i.e. when the change in F is close to zero). This tolerance can be set using dF_tol, which stops the code once the free energy is observed to converge within this tolerance (e.g less than $1e-6$). If T is set, the number of forward iterations is the ratio of T to the time step (the default value or one that the user has set).

User input values will be checked for correct data type and if the values conform with the constraints of the problem.

Checkpointing

The code will write a checkpoint file at regular (user-defined) intervals to a (user-defined) file. This will be done using a counter to measure the number of time steps elapsed after the previous

checkpoint. Once this counter reaches the defined value, it will reset, and a file will be written containing the current state of the concentration, and all metadata used to start the run. This file will be written using the NetCDF external library.

The user will be able to input a checkpoint file to restart the code from. This file will be read, and all data and metadata will be stored in the appropriate variables. The code will then continue as normal.

The user-defined interval should be an integer greater than 0. This will be checked on user input to avoid errors. The user-defined files will be input in the form of a filename, of which the file extension is defined in the code. When reading from a checkpoint file, the code will catch any errors relating to reading it and then prompt the user if they wish to continue without the checkpoint, or abort the run.

The user will be able to specify automatic checkpointing where the difference between two time steps will be evaluated. Checkpoints will be taken when the concentration changes by a large enough amount from the previous checkpoint.

Uncertainty Quantification and Validation

Python will be used to write parts of the code which will address uncertainty quantification. To do this, the Fortran code that will solve the equations will be wrapped using f90wrap. The resulting module can then be used as a solver, which will run multiple times, creating a distribution of the output with variance coming from the randomness in the initialisation of the grid.

The stability of various structural properties, such as the radial distribution function (RDF), has the option to be assessed under different starting conditions. This will be done by running the code several times for each set of initial conditions and looking at the resulting changes in the structural properties. First-order sensitivity analysis will be done by taking first-order differences for each input variable.

Further error analysis of the code will be performed by estimating the error in discretising the problem by n nodes and time steps Δt. Reference solutions will be used to represent the ideal solutions (i.e. solutions using large n and low Δt) with the discretisation error taken to be:

$$||e||_{L_2} = \frac{\sum_i \sum_j \left| c_{i,j} - c_{i,j}^{ref} \right|^2}{\sum_i \sum_j \left| c_{i,j}^{ref} \right|^2}, \tag{1}$$

where the concentration is written in shorthand notation such that $c(x, y) = c_{i,j}$. Convergence results of the spatial and temporal discretisation can then be observed for runs using various values of n and Δt.

Simple test cases will be used to validate our results, for example, setting the initial concentration field to be either constant, uniform 50/50 split or a step-like concentration field. These initial conditions will have an obvious, expected steady-state solution, enabling us to validate our results against the expected physics from the theory of spinodal decomposition.

Sensitivity Analysis (Only an idea, can be removed if it doesn't work)

Several input parameters are required to be inputted by the user, for example the mobility (M), gradient term strength constant (κ), initial average concentration (c_0) and fee energy scale factor

A. It would be nice to able to see which particular parameters effect the final results/physics and quantify them. One such idea would be to perform sensitivity analysis on these parameters using one of two methods:

- Finite Differences
  - Run the code at set input parameters (θ), and measure the quantity of interest (Q_I), and then vary the parameters individually by some small amount ($\epsilon = 1e-6$) and measure Q_I again.
  - Can determine first order sensitivity using finite differences:

$$\frac{Q_I(\theta+\epsilon) - Q_I(\theta)}{\epsilon} \tag{2}$$

  - Output quantity could be either final concentration or equilibrium time
- Gaussian Process Regression
  - Run the code at set input parameters (θ), and measure the quantity of interest (Q_I). Store the various simulation data to train a Gaussian process using GPy package.
  - Use either a square exponential or exponential kernel function:

$$k(x,x') \propto \exp\left(-\frac{(x-x')^2}{2l^2}\right) \text{ or } k(x,x') \propto \exp\left(-\frac{(x-x')}{l}\right). \tag{3}$$

  - Using automatic relevance detection in GPy, can use the GP's length scales to estimate the first order sensitivity (i.e inverse of the length scale).
  - Perhaps use the training data do do some MCMC using a GP to find the posterior distributions of the input parameters. Can set the presumed optimised parameters to those that match the work done in Y. H. Wang et al., J. Appl. 12 85102 (2019)). (Might be nice but not quite right)
  - Output quantity could be either final concentration or equilibrium time

Back-end Development

The code for the software back-end will be written in Modern Fortran. This includes initialisation of the 2D concentration grid, obtaining the bulk chemical potential grid, total chemical potential grid, and eventually solving the Cahn-Hilliard equation to get the evolution of concentration grid with time. The major modules that will be included within the back-end are detailed below.

Concentration Grid Initialisation

Once the average concentration ($C0$) and its standard deviation ($Cstd$) are read from the user input, samples will be generated randomly from its normal distribution; $c\,(x, y, t = 0) \sim \mathcal{N}(\mathrm{C0}, \mathrm{Cstd}^2)$ to initialise the (Nx, Ny) concentration grid.

Bulk Chemical Potential

The bulk chemical potential for each node of the 2D grid will be calculated simply by using the formula obtained after differentiating the bulk free energy density with respect to concentration. Given that the bulk free energy is taken to be a $N-1$ polynomial:

$$f\left(c_{i,j}\right) = \sum_{m=0}^{N-1} a_m \left(c_{i,j}\right)^m , \tag{4}$$

$$\mu\left(c_{i,j}\right) = \sum_{m=0}^{N-1} m\, a_m \left(c_{i,j}\right)^{m-1} . \tag{5}$$

Total Chemical Potential

The total chemical potential is the functional derivative of the total free energy with respect to the concentration (i.e. the Euler-Lagrange equation of F) needed for the Cahn-Hilliard equation. The finite difference method will solve this numerically for each node in the 2D grid. The Laplacian of the concentration will be discretised using the conventional central difference for the second derivative, which has second-order accuracy:

$$\frac{\delta F}{\delta c} = Q \approx \mu\left(c_{i,j}\right) - \kappa \left[\frac{c_{i+1,j} - 2c_{i,j} + c_{i-1,j}}{\left(\Delta x\right)^2} + \frac{c_{i,j+1} - 2c_{i,j} + c_{i,j-1}}{\left(\Delta y\right)^2} \right] + \mathcal{O}\left(\Delta x^2\right) + \mathcal{O}\left(\Delta y^2\right), \tag{6}$$

where $\Delta x, \Delta y$ are the spatial grid spacings in x and y axes and the shorthand $c_{i\pm1,j\pm1} = c\left(x \pm \Delta x, y \pm \delta y\right)$. It should be highlighted here that periodic boundary conditions will be followed to get the concentration Laplacians at the boundary. This will mean that out of bound encounters at the boundaries for finite difference like $c_{N+1,j}$ will be equivalent to $c_{1,j}$ and so on.

Cahn-Hilliard

The Cahn-Hilliard states how the order parameter c evolves with time. This will be solved numerically again using conventional central difference methods of second-order accuracy for the Laplacian term such that:

$$M\nabla^2\left(Q\right) = M \left[\frac{Q_{i+1,j} - 2Q_{i,j} + Q_{i-1,j}}{\left(\Delta x\right)^2} + \frac{Q_{i,j+1} - 2Q_{i,j} + Q_{i,j-1}}{\left(\Delta y\right)^2} \right] + \mathcal{O}\left(\Delta x^2\right) + \mathcal{O}\left(\Delta y^2\right). \tag{7}$$

To perform the time-evolution in accordance with the Cahn-Hilliard equation, implicit time stepping (forward Euler) methods will be used, giving the full discritised Cahn-Hilliard equation:

$$\frac{c_{i,j}^{k+1} - c_{i,j}^{k}}{\Delta t} \approx M \left[\frac{Q_{i+1,j}^{k} - 2Q_{i,j}^{k} + Q_{i-1,j}^{k}}{\left(\Delta x\right)^2} + \frac{Q_{i,j+1}^{k} - 2Q_{i,j}^{k} + Q_{i,j-1}^{k}}{\left(\Delta y\right)^2} \right] + \mathcal{O}\left(\Delta t\right) \tag{8}$$

$$c_{i,j}^{k+1} \approx c_{i,j}^{k} + \Delta t M \left[\frac{Q_{i+1,j}^{k} - 2Q_{i,j}^{k} + Q_{i-1,j}^{k}}{\left(\Delta x\right)^2} + \frac{Q_{i,j+1}^{k} - 2Q_{i,j}^{k} + Q_{i,j-1}^{k}}{\left(\Delta y\right)^2} \right] + \mathcal{O}\left(\Delta t\right) \tag{9}$$

where $c_{i,j}^{k}$ is the solution at the current time step and $c_{i,j}^{k+1}$ is the solution at the next time step. For explicit methods care needs to be taken with the choice of time step such that the method remains stable. The Cahn-Hilliard written in full has the form:

$$\frac{\partial c}{\partial t} = M\nabla^2\mu\left(c\right) - M\kappa\nabla^4 c. \tag{10}$$

By inspection of the derivatives, it is suggested that there is a constraint on the time step such that $\Delta t < \text{Min}\left(\Delta x^4, \Delta y^4\right)$. Von-Neumann analysis was performed on the expanded Cahn-Hilliard equation, with the assumption that we can neglect the discritised non-linear term $\left(\nabla^2\mu\left(c\right)\right)$ as it is assumed $(\Delta x^{-4}), (\Delta y^{-4}) >> (\Delta x^{-2}), (\Delta y^{-2})$. The time step was determined to be stable from this analysis if:

$$\frac{M\kappa\Delta t}{16}\left(\frac{1}{\left(\Delta x\right)^4} + \frac{1}{\left(\Delta y\right)^4}\right) \leq 1 \rightarrow \Delta t \leq \tau = \frac{16}{M\kappa}\left(\frac{1}{\left(\Delta x\right)^4} + \frac{1}{\left(\Delta y\right)^4}\right)^{-1}. \tag{11}$$

From this analysis, the default value of Δt will be set to $\tau/16$ (to have the constant co-factor as unity for simplicity) to ensure that the time-step is stable. Warnings in the code will be put in place in case the user defines a value for Δt that is unstable, mainly if the input value is not of the correct order of magnitude (i.e. $\Delta t < \text{Min}\left(\Delta x^4, \Delta y^4\right)$). If this is the case, then an error message will be displayed in which the user can only continue if they force the code to (similarly to typing `--force` for some command-line based programs) or restart with a different (i.e. stable) time-step.

Total Free Energy

The total free energy F will be used to define when equilibrium (i.e phase separation) has been achieved, as eluded to earlier in the document. The formulation of F is given by the integral shown below:

$$F(t) = \int_V \left[f\left(c(x,y,t)\right) + \frac{1}{2}\kappa\left(\nabla c(x,y,t)\right)^2\right] \mathrm{d}V, \tag{12}$$

which will be numerical solved using the trapezium rule. The discretised formulation is shown below:

$$F(t) = \sum_i \sum_j \left[f\left(c_{i,j}(t)\right) + \frac{1}{2}\kappa\left(P_{i,j}(t)\right)^2\right] \Delta x \Delta y, \tag{13}$$

$$P_{i,j}(t) = \left[\frac{c_{i+1,j}(t) - 2c_{i,j}(t) + c_{i-1,j}(t)}{\left(\Delta x\right)^2} + \frac{c_{i,j+1}(t) - 2c_{i,j}(t) + c_{i,j-1}(t)}{\left(\Delta y\right)^2}\right], \tag{14}$$

and the infinitesimal taken to be $\mathrm{d}V = \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = \Delta x \Delta y$, (where the out of plane dimension taken to have unit infinitesimal).

Parallelisation

Profiling

Once a successful serial implementation is achieved, gprof will be used to collect information on the performance of different parts of the code. This will help to determine performance bottlenecks in the code, which indicate where best to focus parallelisation efforts on.

MPI

MPI parallelism will be used for domain decomposition. A Cartesian topology will be implemented using the appropriate communicator to establish a 2D Cartesian coordinate system for the grid. The global and local grids will be initialised such that the use of domain composition is optional and is determined by the user.

OpenMP

OpenMP multi-threading will be used for each domain created by MPI to accelerate the grid loops, calculating the bulk chemical potential and total chemical potential.

Visualisation

The results from the code will be visualised using Python, specifically the Matplotlib library. The time-resolved concentration field will be visualised as an animated contour plot of the 2D discretised grid, for which the concentration will be plotted in the z-axis. Snapshots of this animation at certain times will also be visualised to validate our code result against previous work (e.g. Figure.4 Y. H. Wang et al., J. Appl. 12 85102 (2019)). Plots of the total free energy over simulation time will also be produced to gauge the timescale over which equilibrium is achieved in the modelled binary-alloy system. Results of the uncertainty quantification will be visualised, including the discretisation error for various time-steps and spatial nodes and the results from the stability and sensitivity analysis.

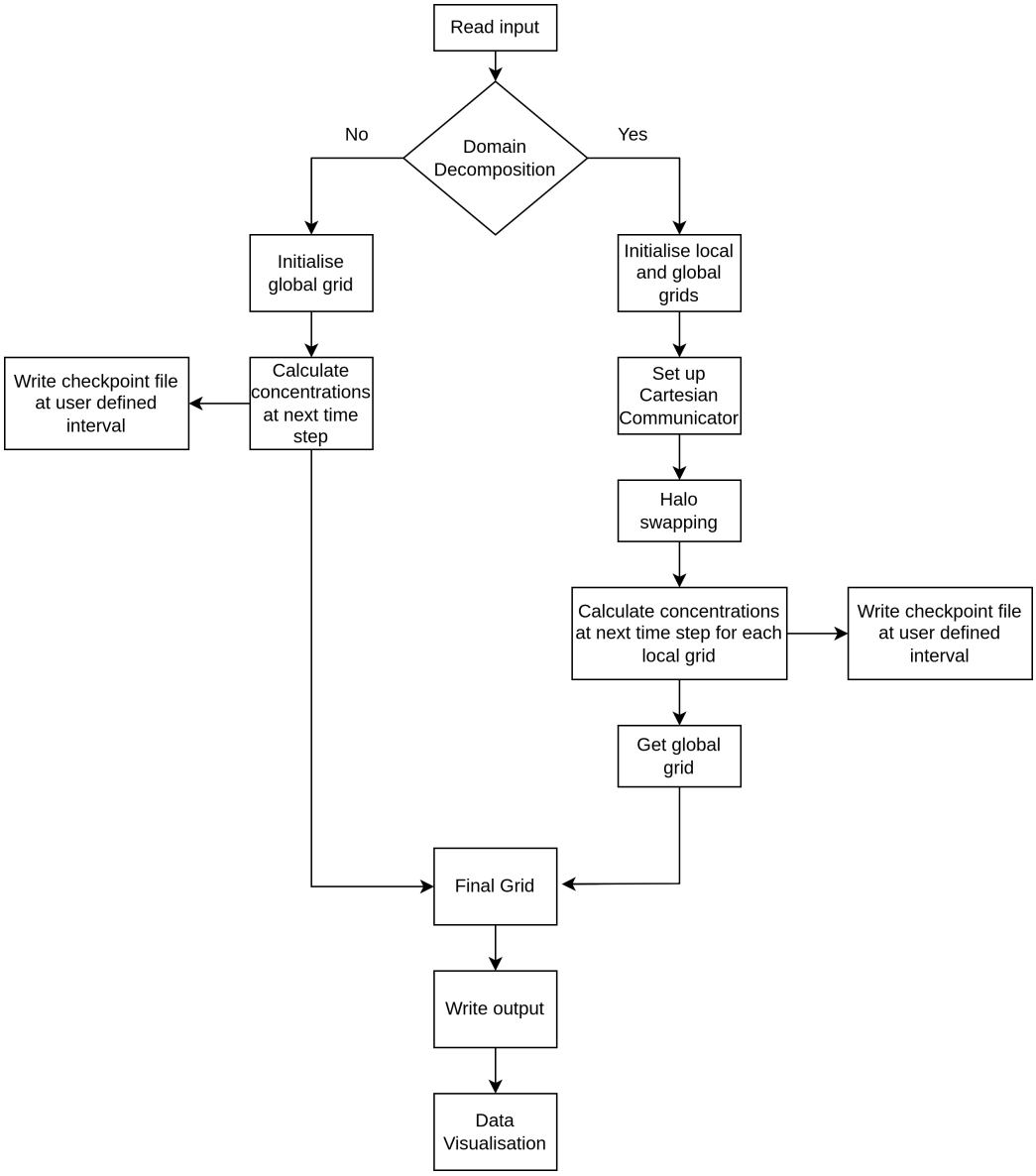

Figure 1: *Proposed workflow of the functionality of the final code.*