Manual for using the finite difference solver based on the grand-potential formalism

Abhik Choudhury

April 14, 2023

1 Introduction

This is a multi-phase multicomponent phase-field solver based on a regular-grid finite-difference discretization, with a simple Euler forward time marching scheme. It is based on the phase-field model presented in *Phys. Rev. E 85, 021602 (2012).* The solver is parallelized using a simple domain-decomposition in one direction using MPI interface. Before, you run the solver there are certain basic operations one needs to carry out.

2 Infile

The input parameters required for the solver are derived from an infile. This contains information about the domain geometry, the thermodynamic functions(free energy), material properties such as the interfacial energies and their anisotropies as well as boundary conditions. It could also contain special flags related to the running of the solver. The following is a basic description of the keys in the infile. **Each key must end with a semicolon**. Additionally, all lines beginning with "#" will be treated as comments in the infile.

```
##Geometrical dimensions of the simulation domain
DIMENSION = 2;
MESH_X = 200;
MESH_Y = 200;
MESH_Z = 1;
##Discretization, space and time
DELTA_X = 2.0;
DELTA_Y = 2.0;
DELTA_Z = 2.0;
DELTA_t = 0.08;
##Number of phases and composition
NUMPHASES = 2;
NUMCOMPONENTS = 2;
#Running and saving information
NTIMESTEPS = 1000;
NSMOOTH = 1;
SAVET = 100;
STARTTIME = 0;
```

```
RESTART = 0;
numworkers = 4;
## Component and Phase names
# COMPONENTS = {Al,Cu,B};
COMPONENTS = {Al, Cu};
PHASES = {alpha, beta};
##Material properties
##GAMMA={12, 13, 14, 23, 24...}
GAMMA = {1.0};
# Diffusivity = {Diagonal:0/1, phase, 11,22,33, 12, 13, 23...};
DIFFUSIVITY = {1, 0, 0};
DIFFUSIVITY = {1, 1, 1};
##Gas constant and molar volume
R = 1.0;
V = 1.0;
##Elasticity related parameters
#EIGEN_STRAIN = {0, 0.01, 0.01, 0.0, 0.0, 0.0, 0.0};
#EIGEN_STRAIN = {1, 0.01, 0.01, 0.0, 0.0, 0.0, 0.0};
#VOIGT_ISOTROPIC = {0, 270, 187.5, 125.0};
#VOIGT_ISOTROPIC = {1, 270, 187.5, 125.0};
ELASTICITY = 1;
EIGEN_STRAIN = {0, 0.01, 0.01, 0.0, 0.0, 0.0, 0.0};
EIGEN_STRAIN = {1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
VOIGT_ISOTROPIC = {0, 270, 187.5, 125.0};
VOIGT_ISOTROPIC = {1, 270, 187.5, 125.0};
rho = 100;
damping_factor = 0.4;
max_iterations = 5;
#VOIGT_CUBIC = {phase, c11, c12, c44};
#VOIGT_TETRAGONAL = {phase, c11, c12, c13, c33, c44, c66};
##Boundary conditions
#0: Free, 1: Neumann, 2: Dirichlet, 3: Periodic, 4: Complex
#Boundary = {phase, X+, X-, Y+, Y-, Z+, Z-}
BOUNDARY = {phi, 3, 3, 3, 3, 0, 0};
BOUNDARY = {mu, 3, 3, 3, 3, 0, 0};
BOUNDARY = {u, 3, 3, 3, 3, 0, 0};
BOUNDARY = {c, 3, 3, 3, 3, 0, 0};
BOUNDARY = {T, 3, 3, 3, 3, 0, 0};
# Boundary = {phi, 1, 1, 0};
# Boundary = {"u", 3, 3, 2, 2};
#Boundary_value = {Value X+, Value X-, Value Y+, Value Y-, Value
    ↪ Z+, Value Z-}
BOUNDARY_VALUE = {phi, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {mu, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {c, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {T, 0, 0, 0, 0, 0, 0};
##Type of simulation
ISOTHERMAL = 1;
BINARY = 1;
#TERNARY
```

```
DILUTE = 0;
T = 0.96;
##FILEWRITING and OUTPUTTING TO SCREEN
## WRITEFORMAT ASCII/BINARY/HDF5(Only for MPI)
##TRACK_PROGRESS: interval of writing out the progress of the
    ↪ simulation to stdout.
WRITEFORMAT = BINARY;
WRITEHDF5 = 1;
TRACK_PROGRESS = 10;
##Model-specific parameters: Grand-potential model
##Phase-field parameters; epsilon:interface width; it is not the
    ↪ gradient energy coefficient
epsilon = 8.0;
tau = 1.31;
Tau = {0.28};
##Anisotropy functions
##Anisotropy mode, FUNCTION_ANISOTROPY=0 is isotropic
Function_anisotropy = 1;
Anisotropy_type = 4;
dab = {0.02};
#Rotation_matrix = {0, 1, Euler_x(ang), Euler_y(ang), Euler_z(ang
    ↪ )};
Rotation_matrix = {0, 1, 0, 0, 0};
##Potential function
Function_W = 1;
Gamma_abc = {};
#Shifting of domain for infinite domain simulations
Shift = 0;
Shiftj = 30;
#Writing of composition fields along with the chemical potential
    ↪ fields
Writecomposition = 1;
#Noise
Noise_phasefield = 0;
Amp_Noise_Phase = 0.001;
##Temperature
Equilibrium_temperature = 1.0;
Filling_temperature = 1.0;
#TEMPGRADY={BASETEMP, DELTAT, DISTANCE, OFFSET, VELOCITY}
Tempgrady = {0.96, 0.06, 800.0, 0, 0.016};
##Function_F
Function_F = 1;
A = {0, 1};
A = {1, 1};
ceq = {0, 0, 0.78125};
ceq = {0, 1, 0.5};
ceq = {1, 1, 0.5};
ceq = {1, 0, 0.5};
cfill = {0, 0, 0.78125};
cfill = {0, 1, 0.5};
cfill = {1, 1, 0.5};
cfill = {1, 0, 0.5};
slopes = {0, 0, 0.45};
slopes = {0, 1, 0.45};
slopes = {1, 0, 0.45};
slopes = {1, 1, 0.45};
```

2.1 Simulation geometry, spatial and temporal discretization

```
##Geometrical dimensions of the simulation domain
DIMENSION = 2;
MESH_X = 100;
MESH_Y = 100;
MESH_Z = 1;
```

- DIMENSION: This can take values 2,3 for 2D and 3D simulations respectively. This is a required key in the solver and not mentioning this key might lead to unexpected results
- MESH_X,MESH_Y,MESH_Z: These are the number of grid points in the domain(and not the physical lengths) in the respective X, Y, Z directions. When DIMENSION is 2, the value of MESH_Z will redundant and will be taken as 1.

```
##Discretization, space and time
DELTA_X = 2.0;
DELTA_Y = 2.0;
DELTA_Z = 2.0;
DELTA_t = 0.08;
```

- The values DELTA_X, DELTA_Y, DELTA_Z correspond to the grid resolution in the X, Y, Z directions respectively. Similarly, DELTA_t corresponds to the temporal discretization(time-step).

2.2 Phases and Components information

```
##Number of phases and composition
NUMPHASES = 2;
NUMCOMPONENTS = 2;
```

- The keys are self-explanatory. NUMPHASES corresponds to the number of phases in the domain, while NUMCOMPONENTS corresponds to the number of components(2, for binary, 3 for ternary etc.)
- These keys are absolutely necessary, please fill carefully for successful running of your codes.

```
COMPONENTS = {Al, Cu};
PHASES = {alpha, beta};
```

- COMPONENTS refers to the tuple that consists of the names of the components. The names are separated by commas and the entire tuple needs to placed within {} followed by a semicolon.
- Similarly, PHASES refers to the names of the phases in the domain.

2.3 Boundary conditions

```
##0:FREE, 1: Neumann, 2: Dirichlet, 3: Periodic, 4: Complex
##BOUNDARY = {TYPE, X_LEFT, X_RIGHT, Y_FRONT, Y_BACK, Z_TOP,
    ↪ Z_BOTTOM}
BOUNDARY = {phi, 1, 1, 1, 1, 0, 0};
BOUNDARY = {mu, 1, 1, 1, 1, 0, 0};
BOUNDARY = {c, 1, 1, 1, 1, 0, 0};
BOUNDARY = {T, 1, 1, 1, 1, 0, 0};
```

- Any of the solvers in repository will consist of the following scalar default types. Type "phi" will represent the phase-field order parameters whose number is specified by the key, "NUMPHASES". Depending on the solver whether it is the grand-potential based solver, where "mu" will be the independent variable or if it is the Cahn-Hilliard or Kim-Kim-Suzuki based solvers, where "c" is the independent variable, Type "mu" or "c" will refer to the components in the key "COMPONENTS". Similarly, type "T" will refer to the temperature field. The BOUNDARY key will refer to the boundary condition for the respective Type of field. The following numbers in a given tuple refer to the boundary at the respective (X_LEFT, X_RIGHT, Y_FRONT, Y_BACK, Z_TOP, Z_BOTTOM) boundaries, that refer to the X, Y, Z boundaries in order. The boundary condition is specified by numbers, 0:FREE, 1: Neumann, 2: Dirichlet, 3: Periodic, 4: Complex, where for the DIRICHLET boundary condition the respective boundary can also take in a specified value which can be specified in the following key: BOUNDARY_VALUE. If the key is not present, the default remains the one that is initialized at the start of the simulation and is left unchanged. Further, if for any direction, for eg: X, one of the extremeties is specified as a PERIODIC boundary, then the other X-boundary will also be initialized as PERIODIC irrespective of the entry in the tuple.

```
#BOUNDARY_VALUE = {Type, Value X+, Value X-, Value Y+, Value Y-,
    ↪ Value Z+, Value Z-}
BOUNDARY_VALUE = {phi, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {mu, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {c, 0, 0, 0, 0, 0, 0};
BOUNDARY_VALUE = {T, 0, 0, 0, 0, 0, 0};
```

- This key corresponds to a specific value that needs to be specified on a boundary which will be held constant during the duration of the simulation. The definition of this key should follow the BOUNDARY specification and follows the same type of definition, except here the tuple Value X_LEFT, Value X_RIGHT, Value Y_FRONT, Value Y_BACK, Value Z_TOP, Value Z_BOTTOM, refer to values on the respective boundaries. The values in this tuple will only be utilized if the respective boundary condition on that boundary is DIRICHLET. By default, the value will be treated as the same as the one that is initialised at the start of the simulation, which will be utilized if this key is not present.

2.4 Number of iterations, smoothing time-steps and writing interval

```
#Running and saving information
NTIMESTEPS = 10000;
NSMOOTH = 10;
SAVET = 1000;
STARTTIME = 6000;
RESTART = 1;
numworkers = 4;
```

- NTIMESTEPS: Total number of iterations that you wish the solver to run. This is not the total time
- NSMOOTH: The number of pre-conditioning steps for smoothening sharp phase-field profiles that are initialised at the start of the simulation
- SAVET: Writing interval, i.e, frequency of writing files of the respective fields
- STARTTIME: The iteration number from which the simulation will start.
- RESTART : This key tells the solver to restart by reading in the files corresponding to STARTTIME. If either of the keys STARTTIME/RESTART are non-zero the code will restart after reading in the files corresponding to the value STARTTIME. If STARTTIME = 0 and RESTART = 1, it gives the possibility to start from an already initialized file. This is useful when the filling operations for initialization are time-consuming.
- Number of workers with which the simulation was executed previously. This is required only for the Grand-potential based MPI solvers.

2.5 Material parameters

```
##Gas constant and molar volume
R = 1.0;
V = 1.0;
```

- The values "R" and "V" will refer to the gas constant and the molar volume respectively

```
#DIFFUSIVITY={Diagonal:0/1, phase, 11,22,33..(K-1) diagonal
    ↪ elements, 12, 13, 23...(rest of the elements; rowwise)}
DIFFUSIVITY = {1, 0, 1};
```

- The DIFFUSIVITY key refers to the inter-diffusivity matrix which has the tuple in the following form. The first element can taken in values 0/1, "1" refers to as a diagonal matrix and "0" is a full matrix.
- The following element refers to the phase number referring to the phases in the list PHASES. This can take values from 0 to NUMPHASES-1.
- The following elements are the values in the inter-diffusivity matrix. The first elements are the diagonal terms in the matrix, while the following elements correspond rowwise to the off-diagonal terms in the diffusivity matrix.
- If the first element in the tuple is "1", i.e. the matrix is diagonal irrespective of the number of entries in the tuple only the entries corresponding to the diagonal elements in the matrix will be read in.

```
##GAMMA = {12, 13, 14, 23, 24...}
GAMMA = {1.0};
```

- The GAMMA key refers to the interfacial energy $\gamma_{\alpha\beta}$ between the phases $\alpha\beta$. The elements in tuple correspond to all combination of phases forming the interfaces from the list of phases in PHASES numbered from 0 to NUMPHASES-1.
- The elements are numbered in the order $12, 13, 14, 23, 24 \ldots N(N-1)$, $N = NUMPHASES$ where "12" corresponds to the value of the interfacial energy between phase 1 and 2; γ_{12}.
- In the tuple only combinations $\alpha\beta$ are included where $\alpha < \beta$ as the value of the interfacial energy of the $\alpha\beta$ interface is $\gamma_{\alpha\beta}$ which is the same as $\gamma_{\beta\alpha}$.
- Therefore, the total number of elements in the tuple is $\frac{N(N-1)}{2}$.

```
#EIGEN_STRAIN = {phase, exx, eyy, ezz, eyz, exz, exy};
EIGEN_STRAIN = {0, 0.01, 0.01, 0.0, 0.0, 0.0, 0.0};
EIGEN_STRAIN = {1, 0.01, 0.01, 0.0, 0.0, 0.0, 0.0};
```

- The EIGEN_STRAIN key refers to the eigen-strain tensor in a given phase. The information about the elements of the eigen-strain are derived from the elements in the tuple.

- The first element refers to the phase number in the list PHASES and can take in values from 0 to NUMPHASES-1.
- The following elements in tuple are mapped to the eigen-strain matrix in the following order $exx, eyy, ezz, eyz, exz, exy$.
- This is an important key required for problems where there are coherent interfaces and the phase transformation is coupled with the stress distribution arising due to coherency strains at the interface.

```
#VOIGT_ISOTROPIC = {phase, c11, c12, c44};
VOIGT_ISOTROPIC = {0, 270, 187.5, 125.0};
VOIGT_ISOTROPIC = {1, 270, 187.5, 125.0};
```

- VOIGT_ISOTROPIC is the key that refers to the elastic stiffness properties in the Voigt notation for an isotropic material.
- The first element in the tuple refers to the phase which will be a number from 0 to NUMPHASES-1.
- The following elements are the values $C11, C12, C44$ in order.

```
#VOIGT_CUBIC = {phase, c11, c12, c44};
VOIGT_CUBIC = {0, 270, 187.5, 125.0};
VOIGT_CUBIC = {1, 270, 187.5, 125.0};
```

- VOIGT_CUBIC is the key that refers to the elastic stiffness properties in the Voigt notation for a cubic material.
- The first element in the tuple refers to the phase which will be a number from 0 to NUMPHASES-1
- The following elements are the values $C11, C12, C44$ in order

```
#VOIGT_TETRAGONAL = {phase, c11, c12, c13, c33, c44, c66};
```

- VOIGT_TETRAGONAL is the key that refers to the elastic stiffness properties in the Voigt notation for a tetragonal material.
- The first element in the tuple refers to the phase which will be a number from 0 to NUMPHASES-1.
- The following elements are the values $C11, C12, C13, C33, C44, C66$ in order.

```
BINARY = 1;
DILUTE = 0;
```

- The keys "BINARY", "TERNARY", "DILUTE" are special flags to the solver allowing for simpler routines in the update of the chemical potential or composition fields.

```
##FILEWRITING and OUTPUTTING TO SCREEN
## WRITEFORMAT ASCII/BINARY
##TRACK_PROGRESS: interval of writing out the progress of the
    ↪ simulation to stdout.
WRITEFORMAT = ASCII;
WRITEHDF5 = 0/1;
TRACK_PROGRESS = 10;
```

- The key WRITEFORMAT mentions the type of the output files to be written. The possible values are ASCII or BINARY that are self-explanatory. When running in parallelized mode with MPI, there is a third possibility for writing files using the key WRITEHDF5 key. For MPI runs, if the WRITEHDF5 key is non-zero, then the value of the WRITEFORMAT key is ignored and files in hdf5 format are written for each time-step (files with extension .h5). Since, the hdf5 file library is linked to the solver by default, the solver needs to be compiled with h5pcc for parallel runs instead of mpicc. This is irrespective of the value of the key WRITEHDF5.
- For the case when the type chosen is BINARY, the format is BIG-ENDIAN such that it matches the BINARY type required for PARAVIEW.
- For MPI runs, in the event WRITEHDF5=0, the WRITEFORMAT key decides the format of the output file, where each processor writes its own file for each time given by the key saveT. In order to view the consolidated file, the tool (./reconstruct) needs to executed in the folder just outside the /DATA folder that is created. The following command needs to be executed for reconstructing,

  ```
  ./reconstruct name_of_infile name_of_output_file
      ↪ number_of_workers start_time end_time
  ```

- For the case when WRITEHDF5=1, all processors write collectively into a single .h5 file. For viewing the file in paraview, it needs to be put in .xml format, which can be performed using the following command, that needs to be executed from just outside the /DATA folder that is created using the run,

  ```
  ./write_xdmf name_of_infile name_of_output_file
      ↪ number_of_workers start_time end_time
  ```

- TRACK_PROGRESS is a key that will inform the solver about the frequency with which the progress of the simulation will be written to stdout.
- The number can be anything other than 0.

2.6 Model specific parameters

The following parameters are corresponding to the multi-phase, multi-component grand-potential model formalism as described briefly below: Order parameter fields ϕ_α are utilized for the describing the distribution of the N phases in the system. They follow the constraint.

$$\phi_\alpha \in [0,1] \quad \text{and} \quad \sum_{\alpha}^{N} \phi_\alpha = 1.$$

The grand potential functional reads,

$$\Omega(T,\mu,\boldsymbol{\phi}) = \int_\Omega \left[\psi(T,\mu,\boldsymbol{\phi}) + \epsilon a(\boldsymbol{\phi},\nabla\boldsymbol{\phi}) + \frac{w(\boldsymbol{\phi})}{\epsilon} + f_{el}\left(\boldsymbol{u},\boldsymbol{\phi}\right) \right] d\Omega. \tag{1}$$

Here, ϵ is the width of the diffuse interface, which is chosen such that the smallest feature in the resulting morphology is accurately resolved. $w(\boldsymbol{\phi})$ is the potential which can be a multi-obstacle or a multi-well potential. The formulation for a double obstacle potential is,

$$w(\boldsymbol{\phi}) = \begin{cases} \sum_{\substack{\alpha<\beta \\ \delta\neq\alpha\neq\beta}}^{N,N} \dfrac{16}{\pi^2}\gamma_{\alpha\beta}\phi_\alpha\phi_\beta + \gamma_{\alpha\beta\delta}\phi_\alpha\phi_\beta\phi_\delta & \text{if } \phi \in [0,1], \\ \infty & \text{otherwise,} \end{cases} \tag{2}$$

while the multi-well potential is of the form;

$$w(\boldsymbol{\phi}) = \begin{cases} \sum_{\substack{\alpha<\beta \\ \delta\neq\alpha\neq\beta}}^{N,N} 9\gamma_{\alpha\beta}\phi_\alpha^2\phi_\beta^2 + \gamma_{\alpha\beta\delta}\phi_\alpha\phi_\beta\phi_\delta \end{cases} \tag{3}$$

$\gamma_{\alpha\beta}$ is the isotropic surface energy of the the $\alpha - \beta$ interface. $\gamma_{\alpha\beta\delta}$ are third-order terms added in order to suppress the adsorption of a third phase at a binary interface. $a(\boldsymbol{\phi},\nabla\boldsymbol{\phi})$ is the gradient energy term and $\psi(T,\mu,\boldsymbol{\phi})$ is the grand potential. The anisotropy in the interface energy is incorporated in the gradient energy term as,

$$a(\phi,\nabla\phi) = \sum_{\alpha<\beta}^{N,N} \gamma_{\alpha\beta}\left[a_c(\vec{q}_{\alpha\beta})\right]^2 |\vec{q}_{\alpha\beta}|^2, \tag{4}$$

where a_c is the anisotropy function of the α - β interface normal vector $\vec{q}_{\alpha\beta} = \phi_\alpha\nabla\phi_\beta - \phi_\beta\nabla\phi_\alpha$. For imparting a particular rotation through an arbitrary angle; we have the following form for the anisotropy function;

$$a(\phi,\nabla\phi) = \sum_{\alpha<\beta}^{N,N} \gamma_{\alpha\beta}\left[a_c(\vec{q'}_{\alpha\beta})\right]^2 |\vec{q}_{\alpha\beta}|^2, \tag{5}$$

where $\vec{q'}_{\alpha\beta} = q'_x, q'_y, q'_z$ corresponds to the rotation of the $\vec{q}_{\alpha\beta}$ by an arbitrary angle given by the rotation matrix R.

The evolution equation for the phase-field parameters reads:

$$\tau\epsilon\frac{\partial \phi_\alpha}{\partial t} = \epsilon\left(\nabla\cdot\frac{\partial a(\phi,\nabla\phi)}{\partial\nabla\phi_\alpha} - \frac{\partial a(\phi,\nabla\phi)}{\partial\phi_\alpha}\right) - \frac{1}{\epsilon}\frac{\partial w(\phi)}{\partial\phi_\alpha} - \frac{\partial\Psi(T,\mu,\phi)}{\partial\phi_\alpha} - \frac{\partial f_{el}\left(\boldsymbol{u},\boldsymbol{\phi}\right)}{\partial\phi_\alpha} - \lambda, \tag{6}$$

where τ is the relaxation constant which controls the kinetics of the phase evolution equation and is interpolated at the interface as $\frac{\sum_{\alpha,\beta}^{N,N}\tau_{\alpha\beta}\phi_\alpha\phi_\beta}{\sum_{\alpha,\beta}^{N,N}\phi_\alpha\phi_\beta}$, λ is the Lagrange multiplier utilized for imposing the constraint $\sum_\alpha^N \phi_\alpha = 1$, i.e. the sum of volume fraction of all the phases is 1.

With this information, one can now follow the parameters in the file corresponding to the model

```
#Phase-field parameters; epsilon:interface width; it is not the
    ↪ gradient energy coefficient
epsilon = 8.0;
#Tau = {12, 13, 14, 23, 24...}
Tau = {0.28};
#tau: Constant value for points in the bulk
tau = 1.31;
```

- "epsilon" is the key that is proportional to the interface width as in Eqn.1.
- Tau is the key that is related to the relaxation of the phase-field equation as detailed in *Phys. Rev. E 85, 021602 (2012)* as described in Eqn.6. The value of the relaxation constants for each of the interfaces $\tau_{\alpha\beta}$ is derived from the tuple similar to the GAMMA key for the interfacial energy, where the elements are arranged in the order $12, 13, 14, 23, 24\ldots N(N-1)$, $N = NUMPHASES$, where "12" corresponds to the relaxation constant of the interface between the phases 1 and 2; τ_{12}.
- In the tuple only combinations $\alpha\beta$ are included where $\alpha < \beta$ as the value of the relaxation constant of the $\alpha\beta$, $\tau_{\alpha\beta}$ is the same as $\tau_{\beta\alpha}$.
- Therefore, the total number of elements in the tuple is $\frac{N(N-1)}{2}$.
- The value of τ in Eqn.6 is derived through a weighted average of the form $\frac{\sum_{\alpha,\beta}^{N,N}\tau_{\alpha\beta}\phi_\alpha\phi_\beta}{\sum_{\alpha,\beta}^{N,N}\phi_\alpha\phi_\beta}$.
- The values of $\tau_{\alpha\beta}$ in the tuple can be chosen according to the thin-interface asymptotic analysis performed in *Phys. Rev. E 85, 021602 (2012)*, such that diffusion-controlled growth may be achieved for the solid-liquid interfaces, while the lowest value among the two solid-liquid interfaces can be chosen for the solid-solid interface, such that the relaxation of the solid-solid interfaces is always faster than the solid-liquid interfaces.
- Also, at points where $\sum_{\alpha,\beta}^{N,N}\phi_\alpha\phi_\beta = 0$, the value of τ is set to the key "tau" in the infile.

```
Function_anisotropy = 1;
Anisotropy_type = 4;
```

- Function_anisotropy is the key to relate to the particular form of $a(\boldsymbol{\phi}, \nabla\boldsymbol{\phi})$ that is utilized as the gradient energy function.
- For a value of zero, the system is isotropic with respect to the change in the interfacial energy with orientation of the interface normal.
- A value of 1 corresponds to a particular function as described in Eqn.4 where the anistropy function is present only in the gradient energy term. There are other possibilities where the anisotropy function may also be incorporated in the potential function $w(\boldsymbol{\phi})$ or a combination of $a(\boldsymbol{\phi}, \nabla\boldsymbol{\phi})$ and $w(\boldsymbol{\phi})$ which will be added on later in the solver.
- The key Anisotropy_type refers to the particular symmetry of the anisotropy. For example a value of 4 implies a four fold anisotropy. This essentially sets the function $a_c(\vec{q}_{\alpha\beta})$ in Eqn.4, that for the value Anisotropy_type=4 reads:

$$a_c(\vec{q}_{\alpha\beta}) = 1 - \delta_{\alpha\beta}\left(3 - 4\frac{q_x^4 + q_y^4 + q_z^4}{|\vec{q}_{\alpha\beta}|^4}\right),$$

  where $\delta_{\alpha\beta}$ is the strength of the anisotropy, while $\vec{q}_{\alpha\beta} = q_x, q_y, q_z$, where q_x, q_y, q_z are the components of the vector $\vec{q}_{\alpha\beta}$.

```
##dab = {12, 13, 14, 23, 24...}
dab = {0.04};
```

- The ”dab” key in the infile is relevant in the presence of anisotropy in the interfacial energies when the key Anisotropy_type=4.
- The ”dab” key refers to the strength of the interfacial energy anisotropy $\delta_{\alpha\beta}$ between the phases $\alpha\beta$. The elements in tuple correspond to all combination of phases forming the interfaces from the list of phases in PHASES numbered from 0 to NUMPHASES-1.
- The elements are numbered in the order $12, 13, 14, 23, 24 \ldots N(N-1)$, $N = NUMPHASES$ where ”12” corresponds to the value of the strength of the interfacial energy anisotropy between phases 1 and 2; δ_{12}.
- In the tuple only combinations $\alpha\beta$ are included where $\alpha < \beta$ as the value of the anisotropy strength of the $\alpha\beta$ interface is $\delta_{\alpha\beta}$ which is the same as $\delta_{\beta\alpha}$.
- Therefore, the total number of elements in the tuple is $\frac{N(N-1)}{2}$.

```
#Rotation_matrix = {0, 1, Euler_x(ang), Euler_y(ang), Euler_z(ang
    ↪ )};
Rotation_matrix = {0, 1, 0, 0, 0};
```

- The "Rotation_matrix" key refers to the rotation of the anisotropy function by a given Euler angle description given by $\theta_x, \theta_y, \theta_z$, where θ_x refers to the angle of rotation about X-axis, θ_y refers to the angle of rotation about Y-axis and θ_z is the angle of rotation about Z-axis.
- The values are taken in as a tuple, where the first two elements correspond to numbers from 0 to NUMPHASES-1 corresponding to combinations of phases that can form an interface.
- This rotation operation will be reflected in the interfacial energy anisotropy of the relevant interface. For example if the first two elements are 0 and 1 then the anisotropy function of the interface between the phases 0 and 1 will be influenced.
- The following three elements represent the three Euler angles $\theta_x, \theta_y, \theta_z$, where θ_x as described before.

```
##Potential function
Function_W = 1;
Gamma_abc = {123, 124, 234 ...(N)(N-1)(N-2)};
```

- The "Function_W" key represents the potential function $w\left(\boldsymbol{\phi}\right)$ in Eqn.1.
- For a value of Function_W=1, the multi-obstacle potential as given in Eqn.2 is selected.
- While for Function_W=2, the multi-well potential as given in Eqn.3 is selected.
- The tuple Gamma_abc={ } is a list of third-order terms $\gamma_{\alpha\beta\delta}$ as described in the Eqns.2,3.
- Only combinations $\alpha\beta\delta$ are considered where $\alpha < \beta < \delta$.
- The total number of terms are (N(N-1)(N-2)/6).

```
Function_F = 1;
A = {0, 1};
A = {1, 1};
ceq = {0, 0, 0.78125};
ceq = {0, 1, 0.5};
ceq = {1, 1, 0.5};
ceq = {1, 0, 0.5};
cfill = {0, 0, 0.78125};
cfill = {0, 1, 0.5};
cfill = {1, 1, 0.5};
cfill = {1, 0, 0.5};
slopes = {0, 0, 0.45};
slopes = {0, 1, 0.45};
slopes = {1, 0, 0.45};
slopes = {1, 1, 0.45};
```

ψ is the grand potential density in Eqn.1, which is obtained at any point as a weighted sum of the the grand potential densities of each phase present at that point,

$$\psi = \sum_{\alpha}^{N} h_\alpha(\boldsymbol{\phi})\psi^\alpha, \tag{7}$$

where, $h_\alpha(\boldsymbol{\phi})$ is a third order interpolation function that reads,

$$h_\alpha(\boldsymbol{\phi}) = 3{\phi_\alpha}^2 - 2{\phi_\alpha}^3 + 2\phi_\alpha \sum_{\substack{\beta,\gamma\neq\alpha \\ \beta<\gamma}}^{N,N} \phi_\beta\phi_\gamma. \tag{8}$$

$\psi^\alpha = \psi^\alpha(\mu, T)$ is the grand potential density of phase α and can be written as

$$\psi^\alpha(T,\mu) = f^\alpha(c^\alpha(T,\mu)) - \mu c^\alpha(T,\mu), \tag{9}$$

where f^α is the Helmholtz free energy density per unit volume. Under constant pressure and volume, f^α, the Helmholtz free energy density differs from the Gibbs free energy density by a constant.

- The key "Function_F" refers to the selection of the particular free-energy density per unit volume.
- For the value of "Function_F=1" it refers to the case where a parabolic free-energy density is chosen of the type described in (Trans Indian Inst Met (2015) 68(6):1137–1143), (Current Opinion in Solid State and Materials Science 19 (2015) 287–300), (PHYSICAL REVIEW E 91, 022407 (2015)).
- The free-energy density is assumed of the form:

$$f^\alpha(\boldsymbol{c}) = \frac{1}{V_m}\Big(\sum_{i<j}^{K,K} A_{ij}^\alpha c_i c_j + \sum_{j}^{K} B_j^\alpha c_j + C^\alpha\Big), \tag{10}$$

- where the coefficients A^α, B^α and C^α are the respective coefficients, V_m is the molar volume that is assumed equal for all the phases in the present discourse and K is the number of independent components.
- The coefficients A_{ij} are linked to the curvatures of the free-energy curves and hence are related to an effective susceptibility. This matrix can be directly obtained from the databases for real material simulations by using the information of the free-energies G^α for each of the phases as $\left(\dfrac{\partial^2 G^\alpha}{\partial c_i \partial c_j}\right)_{\boldsymbol{c}_{eq}^\alpha}$ and $\boldsymbol{c}_{eq}^\alpha$ are the equilibrium compositions of the components in the $\alpha-$phase that is in equilibrium with a given phase, which is presently chosen to be the liquid.
- Returning to keys as you see in Eqn.10, the key "A" in the infile relates to the matrix A_{ij}^α in Eqn.10.

- The first element of the tuple $A = \{\}$, is the number of the phase which can be any number between 0 an NUMPHASES-1, referring to a phase in the list PHASES.
- This is followed by diagonal values of the matrix A_{ij}
- Thereafter, the tuple contains the off-diagonal values of the matrix A_{ij} in the format $i < j$.
- Symmetry is utilized for populating the other values $j > i$.
- Assuming B^l_j and C^l to be zero for the liquid phase, the conditions of equilibrium (equal diffusion potential and equal grand-potentials) allows to determine the other coefficients B^α_j and C^α for any of the solid-phases, such that the equilibrium between the solid phase α and the liquid phase is reproduced at the respective equilibrium compositions. Thus, one can derive for a given temperature T_{eq},

$$B^\alpha_j = 2(A^l_{jj}c^l_{eq,j} - A^\alpha_{jj}c^\alpha_{eq,j}) + \sum_{j\neq i}\left(A^\alpha_{ij}c^\alpha_{eq,i} - A^l_{ij}c^l_{eq,i}\right) \tag{11}$$

$$C^\alpha = \sum_{i\leq j}\left(A^\alpha_{ij}c^\alpha_{eq,i}c^\alpha_{eq,j} - A^l_{ij}c^l_{eq,i}c^l_{eq,j}\right) \tag{12}$$

- Therefore, since the coefficients B_j and C_j may be determined once the coefficients A_{ij} and the equilibrium compositions $\boldsymbol{c}_{eq,j}$ are specified, we take in the information about the equilibrium compositions in the infile.
- The key "ceq" refers to the equilibrium compositions of the phases. The considered equilibrium is between the last phase NUMPHASES-1 and any of the other phases at the temperature provided in the key "Equilibrium_temperature" which is the same as "T_{eq}" in Eqn.11,12.
- The first two elements of the tuple "ceq={}" are numbers between 0 and N = NUMPHASES-1, which correspond to the list of N phases.
- Thereafter, the tuple consists in order, the compositions of the K = NUMCOMPONENTS-1 components.
- The required values are corresponding to all combinations "ceq=$\{a, a, 0, 1 \ldots K\}$" and secondly "ceq=$\{a, N-1, 0, 1 \ldots K\}$" where $a \in [0, N-1]$. The former corresponds to the composition of the "a" phase in equilibrium with the "N-1" phase, and the latter set corresponds to the composition of the "N-1" phase in equilibrium with the "a" phase.
- The remaining possibilities can be skipped.
- The key "cfill" is of the same type as "ceq" and is utilized while initializing the domain. It is safe to start with having the "cfill" the same as "ceq", unless one wants to start with a supersaturation.
- For describing the driving force as a result of small undercoolings, we utilize the slopes of the liquidus m^l_i and the solidus m^α_i , which can be introduced from known thermodynamics, and can be incorporated in the

construction of the respective free-energy densities of the phases. The equilibrium co-existence lines and their variation as a function of temperature are reproduced, according to the approximation that the tie-lines are preserved for smaller undercoolings, i.e. the change in the phase-coexistence is assumed to occur in a self-similar manner, such that the resultant equilibrium compositions at a given undercooling, also exist along the same initial tie-line. This is described in the following image,

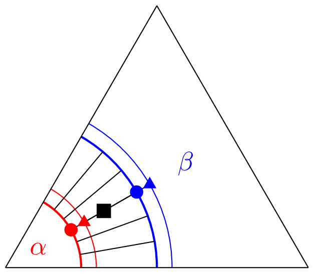


Figure 1: Two-phase equilibrium in a ternary alloy. The phase-co-existence lines at a given temperature are drawn as solid curves and the corresponding tie-lines are drawn as solid black lines between the co-nodes on the co-existence curves. For a particular alloy composition indicated by the solid square, the fitting of the free-energies is performed using the equilibrium compositions comprising the tie-line passing through this point in composition space. For small undercoolings, the change in the phase-co-existence lines shown as the thin solid lines can be assumed to occur self-similar, such that the resultant equilibrium compositions at the undercooled temperature, also exist along the same initial tie-line.

This allows to write the variation of equilibrium phase concentrations of both phases α, l as,

$$\frac{c^{\alpha,l}_{eq,i}\left(T\right) - c^{\alpha,l}_{eq,i}\left(T_{eq}\right)}{c^{\alpha}_{eq,i}\left(T_{eq}\right) - c^{l}_{eq,i}\left(T_{eq}\right)} = \frac{c^{\alpha,l}_{eq,j}\left(T\right) - c^{\alpha,l}_{eq,j}\left(T_{eq}\right)}{c^{\alpha}_{eq,j}\left(T_{eq}\right) - c^{l}_{eq,j}\left(T_{eq}\right)} \qquad \forall \quad i,j \in 1\ldots K. \tag{13}$$

The extent of extrapolation $\Delta T = (T - T_{eq})$ can be expressed as the departure of the equilibrium compositions from the chosen set at temperature T_{eq},

$$\sum_i m^{\alpha,l}_i \left(c^{\alpha,l}_{eq,i}\left(T\right) - c^{\alpha,l}_{eq,i}\left(T_{eq}\right)\right) = (T - T_{eq}).$$

Combining the two preceding relations, we derive the equilibrium concentrations of either phase as functions of temperature along a given tie-line by,

$$c^{\alpha,l}_{eq,i}\left(T\right) = c^{\alpha,l}_{eq,i}\left(T_{eq}\right) + \frac{\left(T - T_{eq}\right)\left(c^{\alpha}_{eq,i}\left(T_{eq}\right) - c^{l}_{eq,i}\left(T_{eq}\right)\right)}{\Delta T^{\alpha,l}_{f}}, \tag{14}$$

where $\Delta T_f^{\alpha,l}$ is given by,

$$\Delta T_f^{\alpha,l} = \sum_i m_i^{\alpha,l} \left(c_{eq,i}^{\alpha} \left(T_{eq}\right) - c_{eq,i}^{l} \left(T_{eq}\right) \right).$$

Using the above relations for the temperature variations of the equilibrium compositions of the phases along a chosen tie-line, one can determine the coefficients $B_j^{\alpha}(T)$ and $C^{\alpha}(T)$ using Eqns. 11, 12, such that the equilibrium co-existence lines are reproduced for small variations of both the composition and the temperature around the chosen equilibrium compositions $\boldsymbol{c}_{eq}$ and temperature T_{eq}.

- The required slopes are m_i^l and m_i^{α} corresponding to the equilibrium between the α and the phase $l = N-1$ (the last phase) for the i^{th} component. The information is taken from the infile similar to the "ceq" key and the description is as follows;
- The key "slopes" refers to the slopes of the co-existence curves between phases. The equilibrium is between the last phase NUMPHASES-1 and any of the other phases at the temperature provided in the key "Equilibrium_temperature" which is the same "T$_{eq}$".
- The first two elements of the tuple "ceq={}" are numbers between 0 and N = NUMPHASES-1, which correspond to the list of N phases.
- Thereafter, the tuple consists in order, the slopes for the K = NUMCOMPONENTS-1 components.
- The required values are corresponding to all combinations "slopes=$\{a, a, 0, 1 \dots K\}$" and secondly "slopes=$\{a, N-1, 0, 1 \dots K\}$" where $a \in [0, N-1]$. The former corresponds to the slopes of the "a" phase in equilibrium with the "N-1" phase, and the latter set corresponds to the slope of the "N-1" phase in equilibrium with the "a" phase. For solidification reactions, this would correspond to solidus and liquidus slopes respectively.
- The remaining possibilities can be skipped.

```
Function_F = 2;
num_thermo_phases = 2;
tdbfname = alzn_mey.tdb;
tdb_phases = {FCC_A1, LIQUID};
phase_map = {FCC_A1, LIQUID};
c_guess = {0, 0, 0.92133};
c_guess = {0, 1, 0.80354};
c_guess = {1, 1, 0.80354};
c_guess = {1, 0, 0.80354};
```

- This function chooses the option that the thermodynamic functions from the databases will be utilized for the computation of the driving forces.
- There are additional Infile keys corresponding to this selection.

- "tdbfname" is the name of the thermodynamic database from which the thermodynamic functions will be extracted. The extraction occurs using a separate python code "GE_data_writer.py" which extracts the required thermodynamic functions and places them in the folder "tdbs" which will be utilized by the solvers. The generated files are "Thermo.c" and "Thermo.h" which are included in the solver.

- "num_thermo_phases" is the number of thermodynamic distinct phases that the user wishes to consider for the simulation.

- "tdb_phases" is the tuple mentioning the names of the thermodynamic phases that will be considered from the .tdb file. It is important that the names of these phases be exactly the same as in the database. This can be pre-checked by running the "GE_data_writer.py" separately by the user, which interfaces with the .tdb. The length of this tuple must be exactly the same as "num_thermo_phases".

- "phase_map" is the tuple that mentions the association of the phases in the tuple "PHASES" with the thermodynamic phases mentioned in "tdb_phases". It is important that the length of the tuple "phase_map" be exactly the same as the length of the tuple "PHASES" as this ensures that each phase has a thermodynamic association and the corresponding thermodynamics will be utilized for its evolution.

- The tuples "tau" and "Tau" can be avoided as the values are self-consistently calculated for diffusion controlled growth.

- Since, the thermodynamic functions may not be convex through the interval and in general are non-linear, it is important to evaluate the function $\mathbf{c}(\mu)$ through a numerical inversion route as an analytical inversion may not be possible.

- For this the functions in gsl/gsl_multiroots.h are utilized as these are heavily optimized for the solution of these non-linear problems. In addition, processor specific optimization such as the intel-oneapi-mkl libraries may be linked for superior performance.

- The numerical evaluation of the $\mathbf{c}(\boldsymbol{\mu})$ allows the function to be locally treated in a convex fashion and thereby the driving forces can be calculated.

- In order to limit the evaluation of the function $\mathbf{c}(\boldsymbol{\mu})$ to the local equilibrium valley of a a given phase, it is necessary to provide guess for the iteration routines called from the gsl library. These guesses are provided in the tuple "c_guess" which can be interpreted in the same manner as "c_eq" or "c_fill".

- Different iterative routines from the gsl_multiroots library may be chosen based on convergence and the particular problem. The hybridj and hybridsj solvers work the best when it comes to global convergence behavior.

- Since, these are non-linear functions, in contrast to the parabolic functions the matrices $\left[\dfrac{\partial c_i}{\partial \mu_j}\right]$ are composition dependent. Therefore, although the

$\boldsymbol{\mu}$ update equation is derived from the condition $c_i = \sum_\alpha \boldsymbol{c}_i^\alpha(\boldsymbol{\mu})\, h_\alpha(\phi)$ along with the mass conservation equation, the diffusion potential evolution equation linearizes the dependencies on each of the variables. This can introduce errors in the composition constraint for certain cases.

- In order to avoid this, an iterative scheme is devised for correcting the diffusion potential update if required. The iterative scheme in pseudo-code is written as follows and is detailed in FunctionF_02.h. The conservation equation for each of the components follows

$$\frac{\partial c_i}{\partial t} = \nabla \cdot \left(M_{ij} \nabla \mu_j\right), \tag{15}$$

where $M_{ij} = \sum_{\alpha=1}^{N} \left[D_{ij}^\alpha\right] \left[\frac{\partial c_i^\alpha}{\partial \mu_{ij}}\right] h_\alpha(\boldsymbol{\phi})$. This can be converted into a diffusion potential update equation by linearising based on the dependencies on the individual variables as,

$$\left\{\frac{\partial \mu_i}{\partial t}\right\} = \left[\sum_{\alpha=1}^{N} h_\alpha(\boldsymbol{\phi}) \frac{\partial c_i^\alpha(\boldsymbol{\mu}, T)}{\partial \mu_j}\right]_{ij}^{-1} \Bigg\{\nabla \cdot \sum_{j=1}^{K} M_{ij}(\boldsymbol{\phi}) \nabla \mu_j - \sum_{\alpha}^{N} c_i^\alpha(\boldsymbol{\mu}, T) \frac{\partial h_\alpha(\boldsymbol{\phi})}{\partial t} - \frac{\partial T}{\partial t} \sum_{\alpha}^{N} \left(\frac{\partial c_i^\alpha(\boldsymbol{\mu}, T)}{\partial T}\right)_{\boldsymbol{\mu}} h_\alpha(\boldsymbol{\phi}) \Bigg\}, \tag{16}$$

using the composition constraint $c_i = \sum_{\alpha=1}^{N} c_i^\alpha(\boldsymbol{\mu})\, h_\alpha(\boldsymbol{\phi})$. However, as mentioned when the matrices $\left[\frac{\partial c_i^\alpha}{\partial \mu_j}\right]$ become composition dependent, there can be errors in the composition constraint, depending on the changes in the other variables in a given time step. In order to account for this, a hybrid approach is devised, in which the composition is updated using the mass conservation equation in Eqn.15, along with the explicit diffusion potential update in Eqn.16 and the following correction scheme is implemented.

  1. Step 1: Update composition using Eqn.15 and diffusion potential using Eqn.16.
  2. Step 2: Calculate the error in each composition of each component as $\{\Delta c_i\} = c_i - \sum_{\alpha=1}^{N} c_i^\alpha(\boldsymbol{\mu})\, h_\alpha(\boldsymbol{\phi})$. For this you need to call the functions that compute the phase compositions using the gsl functions.
  3. Step 3: Compute matrices $\left[\frac{\partial c_i^\alpha}{\partial \mu_j}\right]$ for each phase and calculate the matrix $\left[\frac{\partial c_i}{\partial \mu_j}\right] = \sum_{\alpha=1}^{N} \left[\frac{\partial c_i^\alpha}{\partial \mu_j}\right] h_\alpha(\boldsymbol{\phi})$.

4. Step 3: Compute correction in $\boldsymbol{\mu}$ by $\{\Delta\mu_i\} = \left[\frac{\partial c_i}{\partial \mu_j}\right]^{-1} \{\Delta c_i\}$.
5. Update diffusion potential as $\{\mu_i\} = \{\mu_i\} + \{\Delta\mu_i\}$.
6. Check tolerance and continue from Step 2 if not satisfied.

- One might recognize that the procedure illustrated is a Newton iteration in the variable $\boldsymbol{\mu}$, starting from the explicit update from Eqn.16 as the initial guess. This greatly simplifies the iterations from the classical version in typical Kim-Kim-Suzuki models as the number of iterations are greatly reduced. Because of this, it is also prudent for most cases to avoid this iteration altogether, as the magnitude of errors are already very small, and it avoids the computation of the matrices $\left[\frac{\partial c_i^{\alpha}}{\partial \mu_j}\right]$ that involve costly matrix inversion operations.

```
Function_F = 3;
num_thermo_phases = 2;
tdbfname = alzn_mey.tdb;
tdb_phases = {FCC_A1, LIQUID};
phase_map = {FCC_A1, LIQUID};
ceq = {0, 0, 0.78125};
ceq = {0, 1, 0.5};
ceq = {1, 1, 0.5};
ceq = {1, 0, 0.5};
c_guess = {0, 0, 0.92133};
c_guess = {0, 1, 0.80354};
c_guess = {1, 1, 0.80354};
c_guess = {1, 0, 0.80354};
```

- This function chooses the option to couple with information from thermodynamic databases using an approximate parabolic formulation.
- This function is built on the Function_F=1, where parabolas are utilized to describe the free-energies of the phases.
- In this function however, the thermodynamic information from the databases is utilized to construct the coefficients A_{ij}, B_j and C_j.
- The function requires the same keys as Function_F=2, where the number of thermodynamic phases and the phase association with the phases needs to be explicitly put in.
- For this function the tuples "slopes" and "A" can be avoided as the required information is constructed from the thermodynamic database.
- The tuples "tau" and "Tau" can be avoided as the values are self-consistently calculated for diffusion controlled growth.
- The matrices A_{ij}^{α} for each phase α are evaluated using the matrices as $\left[\frac{\partial c_i}{\partial \mu_j}\right]$ that are computed from the thermodynamic functions in the .tdb

files at the compositions mentioned in tuple "ceq" at the temperature T. For components $i = j$, $A^{\alpha}_{ii} = \dfrac{1}{2}\dfrac{\partial c^{\alpha}_i}{\partial \mu_i}$ and $A^{\alpha}_{ij} = \dfrac{\partial c^{\alpha}_i}{\partial \mu_j}$, for $i \neq j$. The $\left[\dfrac{\partial c_i}{\partial \mu_j}\right]$ is computed by first evaluating $\left[\dfrac{\partial \mu_i}{\partial c_j}\right]$ from the thermodynamic functions in "Thermo.c" generated from the databases and then inverting them.

- For the determination of the slopes the following scheme is utilized. For any solid α phase in equilibrium with the liquid, equilibrium co-existence is maintained if the following holds;

$$\left(\frac{\partial \psi^{\alpha}}{\partial T}\right)_{\boldsymbol{\mu}} \Delta T + \sum_{i=1}^{K-1} \left(\frac{\partial \psi^{\alpha}}{\partial \mu_i}\right)_T \Delta \mu_i = \left(\frac{\partial \psi^{l}}{\partial T}\right)_{\boldsymbol{\mu}} \Delta T + \sum_{i=1}^{K-1} \left(\frac{\partial \psi^{l}}{\partial \mu_i}\right)_T \Delta \mu_i.$$

At constant diffusion potential, it can be shown that $\left(\dfrac{\partial \psi^{\alpha}}{\partial T}\right)_{\boldsymbol{\mu}} = \left(\dfrac{\partial f^{\alpha}}{\partial T}\right)_{\boldsymbol{c}}$. Similarly, $\dfrac{\partial \psi^{\alpha}}{\partial \mu_i} = -c^{\alpha}_i$. With this, the preceding relation may be re-written as,

$$\left(\frac{\partial f^{\alpha}}{\partial T}\right)_{\boldsymbol{c}^{\alpha}} - \left(\frac{\partial f^{l}}{\partial T}\right)_{\boldsymbol{c}^{l}} = \sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \Delta \mu_i.$$

The preceding relation has infinite solutions for components more than 2. However, since we are interested in the determination of slopes, we need to determine co-existence points upon variation of one component(along with solvent) at a given time. Thus, the changes in the diffusion potential may be written as a function of change in the composition of a component j along with change in T. With this, the preceding relation becomes;

$$\left(\left(\frac{\partial f^{\alpha}}{\partial T}\right)_{\boldsymbol{c}^{\alpha}} - \left(\frac{\partial f^{l}}{\partial T}\right)_{\boldsymbol{c}^{l}}\right) \Delta T = \sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\frac{\partial \mu_i}{\partial c_j}\right)_T \Delta c_j + \sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\frac{\partial \mu_i}{\partial T}\right)_{\boldsymbol{c}} \Delta T.$$

Re-arranging, we derive;

$$\frac{\Delta T}{\Delta c_j} = \frac{\sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\dfrac{\partial \mu_i}{\partial c_j}\right)_T}{\left(\dfrac{\partial f^{\alpha}}{\partial T}\right)_{\boldsymbol{c}^{\alpha}} - \left(\dfrac{\partial f^{l}}{\partial T}\right)_{\boldsymbol{c}^{l}} - \sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\dfrac{\partial \mu_i}{\partial T}\right)_{\boldsymbol{c}}}$$

In the limit $\Delta c_j \to 0$, we have for each solid phase, the solidus slope as,

$$\frac{\partial T}{\partial c^{\alpha}_j} = \frac{\sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\dfrac{\partial \mu^{\alpha}_i}{\partial c^{\alpha}_j}\right)_T}{\left(\dfrac{\partial f^{\alpha}}{\partial T}\right)_{\boldsymbol{c}^{\alpha}} - \left(\dfrac{\partial f^{l}}{\partial T}\right)_{\boldsymbol{c}^{l}} - \sum_{i=1}^{K-1} \left(c^{\alpha}_i - c^{l}_i\right) \left(\dfrac{\partial \mu^{\alpha}_i}{\partial T}\right)_{\boldsymbol{c}^{\alpha}}}$$

and the corresponding liquidus slope as,

$$\frac{\partial T}{\partial c_j^l} = \frac{\sum_{i=1}^{K-1}\left(c_i^\alpha - c_i^l\right)\left(\dfrac{\partial \mu_i^l}{\partial c_j^l}\right)_T}{\left(\dfrac{\partial f^\alpha}{\partial T}\right)_{\boldsymbol{c}^\alpha} - \left(\dfrac{\partial f^l}{\partial T}\right)_{\boldsymbol{c}^l} - \sum_{i=1}^{K-1}\left(c_i^\alpha - c_i^l\right)\left(\dfrac{\partial \mu_i^l}{\partial T}\right)_{\boldsymbol{c}^l}}$$

- Utilizing the information about the slopes and the coefficients A_{ij} the coefficients B_j and C_j are determined just as in Function_F=1.

```
Function_F = 4;
num_thermo_phases = 2;
tdbfname = alzn_mey.tdb;
tdb_phases = {FCC_A1, LIQUID};
phase_map = {FCC_A1, LIQUID};
ceq = {0, 0, 0.78125};
ceq = {0, 1, 0.5};
ceq = {1, 1, 0.5};
ceq = {1, 0, 0.5};
```

- This function chooses to couple with thermodynamic databases using the parabolic approximation, however, the coefficients of the free-energy expressions are derived not from the functional evaluation of the expressions from the databases, rather they are computed using information read in from files present in the folder "tdbs_encrypted".

- The folder contains files "Composition_'Phase_name'.csv" that contains information about compositions of the phase 'Phase_name' for each of the components at different temperatures in the solidification interval, followed by the compositions of the last thermodynamic phase that is in equilibrium with the phase 'Phase_name'. The folder also contains files "HSN_'Phase_name'.csv" for each thermodynamic phase. This file contains the components of the Hessian matrix for each of the thermodynamic phases for each of the temperatures in the solidification interval for a given alloy composition.

- The phase compositions in "Composition_'Phase_name'.csv" are fitted with csplines using the gsl libraries. Similarly each of the components of the Hessian matrix present in the files "HSN_'Phase_name'.csv" are fitted as a function of temperatures with csplines using the gsl libraries.

- The information from the Hessian as well as the phase compositions are utilized to create the coefficients A, B and C just like in functions F=1,3.

- This function provides the option of incorporating thermodynamic information from encrypted databases by utilizing information relating to Hessian and the equilibrium compositions in a temperature interval that can be computed using softwares like Pandat and ThermoCalc.

```
##Temperature
ISOTHERMAL = 1;
#Tempgrady={BASETEMP, DELTAT, DISTANCE, OFFSET, VELOCITY}
Tempgrady = {0.96, 0.06, 800.0, 0, 0.016};
Equilibrium_temperature = 1.0;
Filling_temperature = 1.0;
T = 0.96;
```

- The key "ISOTHERMAL" is flag for the solver setting it for simulations where the temperature "T" will fixed to the value given by the key "T". Setting it to 1 sets the solver up for isothermal simulations.
- In the event "ISOTHERMAL=0" the temperature is now not fixed. In the present solver, temperature evolution is not solved yet. However, the option exists to have a non-uniform temperature distribution such as giving a uniform gradient distribution that moves in a particular direction at a perscribed velocity V. This mimics directional solidification conditions. By default the gradient has been defined in the y-direction.
- The tuple Tempgrady={BASETEMP, DELTAT, DISTANCE, OFFSET, VELOCITY} is read in for setting the properties of thermal gradient. The first value sets the BASETEMP which is the value at a given location, DELTAT is the temperature increment at a distance given by DISTANCE from the given location(where the temperature was BASETEMP), which sets the magnitude of the thermal gradient. Typically, the BASETEMP is given for the location at y=0. If this is not the case, y=OFFSET is utilized to mention the value of BASETEMP corresponding to y=OFFSET. VELOCITY sets the rate of advance of the linear isotherms in the y-direction corresponding to the imposed value of the directional solidification velocity.
- The key "T" sets the value of the temperature for isothermal simulations when the key "ISOTHERMAL = 1".
- The key "Equilibrium_temperature" is the value of "T_{eq}" for the equilibrium compositions given in the key "ceq" and the "slopes" that are defined in the infile. It is this value that is used for determining the phase-compositions as a function of temperature as described in Eqn.14.
- The key "Filling_temperature" allows one to use a different temperature for the filling operations concerning the computation of the chemical potential. This is useful for non-isothermal simulations when re-starting the simulation from an arbitrary time, by reading from a file.

```
Shift = 0;
ShiftJ = 50;
Writecomposition = 0;
Noise_phasefield = 0;
Amp_Noise_Phase = 0.001;
```

- The key "Shift" refers to the simulation setting where in the case Shift = 1 we have all the fields shifted in the negative y-direction periodically, as and when the phase boundaries reach the grid-position specified by the key "ShiftJ".
- This allows for simulations of infinite growth in the y-direction, relevant for directional solidification, wherein the dynamics in the solid-state is zero because of no diffusivity.
- "Writecomposition" is a specific key that tells the file-writing function to convert the chemical potential fields into the composition fields and write them along with the chemical potential fields.
- "Noise_phasefield" is a key to impose noise in the phase-field evolution equations at the interface. This is typically utilized for dendritic simulations.
- When "Noise_phasefield=1", the amplitude of the noise is specified by the key "Amp_Noise_Phase".

```
rho = 10;
damping_factor = 0.4;
damping_factor = 0.8;
max_iterations = 5;
```

- The preceding parameters are relevant for the grand-potential based MPI solver when the key "ELASTICITY=1" or when elasticity is turned on
- The elasticity component in the functional Eqn.1, i.e. the term $f_{el}\left(\boldsymbol{u}, \boldsymbol{\phi}\right)$ is activated when the key "ELASTICITY=1" in the infile.
- The present implementation is restricted for cubic systems.
- We follow the Khachaturyan interpolation scheme in the interpolation of the elastic properties.
- In this interpolation method, the elastic energy density writes as,

$$f_{el}(\boldsymbol{u}, \phi) = \frac{1}{2}C_{ijkl}(\boldsymbol{\phi})(\epsilon_{ij} - \epsilon^*_{ij}(\boldsymbol{\phi}))(\epsilon_{kl} - \epsilon^*_{kl}(\boldsymbol{\phi})), \tag{17}$$

where the total strain can be computed from the displacement $\boldsymbol{u}$ as,

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \tag{18}$$

while the elastic constants C_{ijkl} and eigenstrain ϵ^*_{ij} can be expressed as:

$$C_{ijkl}(\boldsymbol{\phi}) = \sum_{\alpha=1}^{N} C^{\alpha}_{ijkl}\phi_\alpha \tag{19}$$

$$\epsilon^*_{ij}(\boldsymbol{\phi}) = \sum_{\alpha=1}^{N} \epsilon^{*\alpha}_{ij}\phi_\alpha.$$

To simplify the equations, without any loss of generality we additionally impose that the eigenstrain exists only in precipitate phases which makes the eigenstrain in the matrix phases $\epsilon_{ij}^{*\beta} = 0$.

- Mechanical equilibrium is imposed for the computation of the displacement fields as a function of the spatial distribution of the order parameter. This is done iteratively by solving the damped wave equation written as,

$$\rho \frac{d^2 \boldsymbol{u}}{dt^2} + b \frac{d\boldsymbol{u}}{dt} = \nabla \cdot \boldsymbol{\sigma}, \tag{20}$$

  that is solved until the equilibrium is reached, i.e $\nabla \cdot \boldsymbol{\sigma} = \mathbf{0}$.

- The terms ρ and b are chosen such that the convergence is achieved in the fastest possible time.
- The key "rho" corresponds to ρ in the iterative equation.
- The key "damping_factor" corresponds to b in the iterative equation.

3 Filling

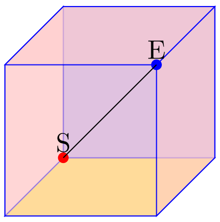
The instructions about filling the domain with phases present in the file "filling.in", which is detailed in

```
#FILLCUBE = {0, 0, 0, 0, 5, 5, 5};
#FILLCUBE = {1, 25, 0, 0, 50, 5, 0};
##FILLCYLINDER = {phase, x_centre, y_centre, z_start, z_end,
    ↪ radius}
#FILLCYLINDER = {0, 49, 49, 0, 0, 12};
# FILLCYLINDERRANDOM = {0, 5, 0.02, 4, 0.1};
# FILLCYLINDERRANDOM = {1, 5, 0.02, 4, 0.1};
# FILLCYLINDERRANDOM = {2, 5, 0.1, 4, 0.1};
# FILLCYLINDERRANDOM = {4, 5, 0.1, 4, 0.1};
# FILLCYLINDERRANDOM = {5, 5, 0.1, 4, 0.1};
# FILLCYLINDER = {0, 0, 0, 0, 0, 10};
# FILLCYLINDER = {1, 35, 40, 0, 0, 5};
# FILLCYLINDER = {2, 180, 40, 0, 0, 5};
# FILLCYLINDER = {3, 75, 125, 0, 0, 5};
# FILLCYLINDER = {4, 50, 100, 0, 0, 5};
#FILLCUBE = {0, 0, 0, 0, 200, 15, 0};
#FILLCUBE = {1, 16, 0, 0, 33, 10,0};
#FILLCUBE = {2, 33, 0, 0, 50, 10,0};
###FILLELLIPSE = {phase, x_center, y_center, major_axis,
    ↪ eccentricity, rotation_angle_deg}
#FILLELLIPSE = {0, 50, 50, 0, 10, 0.1, 10};
FILLSPHERE = {0, 49, 49, 49, 10};
```

. In the following, the possible filling routines are described. The filling instructions may be repeated, that allows for the filling of complicated shapes with overlapping.

```
#FILLCUBE = {phase, x_start, y_start, z_start, x_end, y_end,
    ↪ z_end}
FILLCUBE = {0, 10, 10, 0, 20, 20, 0};
```

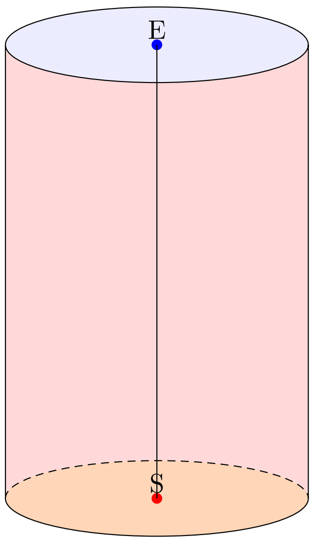
- The "FILLCUBE" key can be utilized for initializing a cube.
- The input about the dimensions of the cube are given in the form of a tuple FILLCUBE = {phase, x_start, y_start, z_start, x_end, y_end, z_end}, where the first element is a number between 0 and NUMPHASES-1, which indicates a phase in the list PHASES.
- The following elements are the co-ordinates(x,y,z) of the end-points of the diagonal in the cube. In the following figure, (x_start, y_start, z_start) corresponds to point S and (x_end, y_end, z_end) corresponds to point E in the cube.
- This operation leads to the filling of the phase given by the number "phase" in the shape of a cube. All other phases are initialised as zero in this region while the rest of the domain is filled with the last phase NUMPHASES-1.
- For 2D object, the z dimensions should be initialised as 0. This would initialize a rectangle.



```
##FILLCYLINDER = {phase, x_centre, y_centre, z_start, z_end,
    ↪ radius}
FILLCYLINDER = {0, 50, 50, 0, 0, 20};
```

- The "FILLCYLINDER" key allows one to initialize a phase in the shape of a cylinder.
- The dimensions are provided in the form of a tuple {phase, x_centre, y_centre, z_start, z_end, radius}.
- The first element "phase" is a number between 0 and NUMPHASES-1, which indicates a phase in the list PHASES.
- The next elements (x_centre, y_centre, z_start) refer to the point S in the following figure.
- The next dimension is z_end which sets the length of the cylinder, i.e the co-ordinate of point E in the following figure is (x_end, y_end, z_end).

- Finally, the radius of the cylinder is specified by the final element in the tuple.
- This operation leads to the filling of the phase given by the number ”phase” in the shape of a cylinder. All other phases are initialised as zero in this region while the rest of the domain is filled with the last phase NUMPHASES-1.
- For 2D object, the z dimensions should be initialised as 0. This would initialize a circle.
- Presently, the FILLCYLINDER is designed to fill a cylinder oriented in the Z-direction.



```
#FILLSPHERE = {phase, x_center, y_center, z_center, radius};
FILLSPHERE = {0, 50, 50, 0, 10};
```

- The ”FILLSPHERE” key allows the filling of a phase in the form of a sphere.
- The dimensions are provided in the form of a tuple {phase, x_center, y_center, z_center, radius} where the first element ”phase” is a number from 0 to NUMPHASES-1, which corresponds to a phase in the list PHASES.
- The center of the sphere is provided in the next three elements of the tuple (x_center, y_center, z_center).
- The last element is the radius of the sphere.
- This operation leads to the filling of the phase given by the number ”phase” in the shape of a sphere. All other phases are initialised as zero in this region while the rest of the domain is filled with the last phase NUMPHASES-1.
- This object can only be used when the DIMENSION = 3 in the infile.

4 Compiling and running

- The solver can be compiled using the command "make" in the command line in a terminal that is opened in the folder which contains the "Makefile".
- After compilation the executable "microsim_gp" will be created.
- The serial solver can be run as ./microsim_gp name_of_infile name_of_filling_file name_of_output_file.
- The MPI version of the solver can be run using the command

```
mpirun -np num_processors Input_file Filling_file Output
    ↪ workers_x, workers_y
```

- The solver takes in three command line arguments, the first is the name of the Input file, the second the name of the filling file containing the information about how the domain should be initialized and thirdly the name of the output file, followed by the number of workers in the x,y dimensions for a 2D solver and x,y,z for a 3D solver.
- The output of the program will be written in a folder "DATA" with the filename "name_of_output_file_timestep.vtk" for the separate timesteps for the serial solver.
- For the MPI version of the solver the output depends on the WRITEFORMAT and WRITEHDF5 keys. For MPI runs, if the WRITEHDF5 key is non-zero, then the value of the WRITEFORMAT key is ignored and files in hdf5 format are written for each time-step (files with extension .h5). Since, the hdf5 file library is linked to the solver by default, the solver needs to be compiled with h5pcc for parallel runs instead of mpicc. This is irrespective of the value of the key WRITEHDF5.
- For the case when the type chosen is BINARY, the format is BIG-ENDIAN such that it matches the BINARY type required for PARAVIEW.
- For MPI runs, in the event WRITEHDF5=0, the WRITEFORMAT key decides the format of the output file, where each processor writes its own file for each time given by the key saveT. In order to view the consolidated file, the tool (./reconstruct) needs to executed in the folder just outside the /DATA folder that is created. The following command needs to be executed for reconstructing,

```
./reconstruct name_of_infile name_of_output_file
    ↪ number_of_workers start_time end_time
```

- For the case when WRITEHDF5=1, all processors write collectively into a single .h5 file. For viewing the file in paraview, it needs to be put in .xml format, which can be performed using the following command, that needs to be executed from just outside the /DATA folder that is created using the run,

```
./write_xdmf name_of_infile name_of_output_file start_time
   ↪ end_time
```

- The files at all the timesteps can be opened as a group in the software "paraview" and the fields according to the names given in the COMPONENTS and PHASES keys can be visualized.
- As a check the solver also outputs two files "name_of_infile.out" and "name_of_infile.bd" that prints out the parameters that have been read in by the solver as well as the boundary conditions initialized respectively.
- Note, these instructions are presently solver specific. After integration of the software stack, the instructions will get modified slightly.

5 Details about parallelization

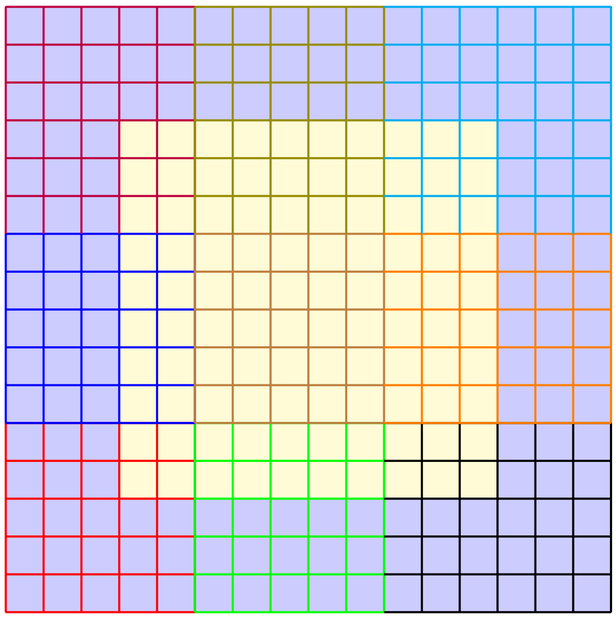


Figure 2: Schematic showing the domain decomposition among the different processors.

- The domain decomposition scheme is illustrated in the Fig.2.
- The original domain consists of the real domain points colored in yellow surrounded by 3 layers of boundary buffer cells in each direction colored in blue.
- The decomposition of the domain results in processors that contain boundary cells and processors that consist of only real domain points.

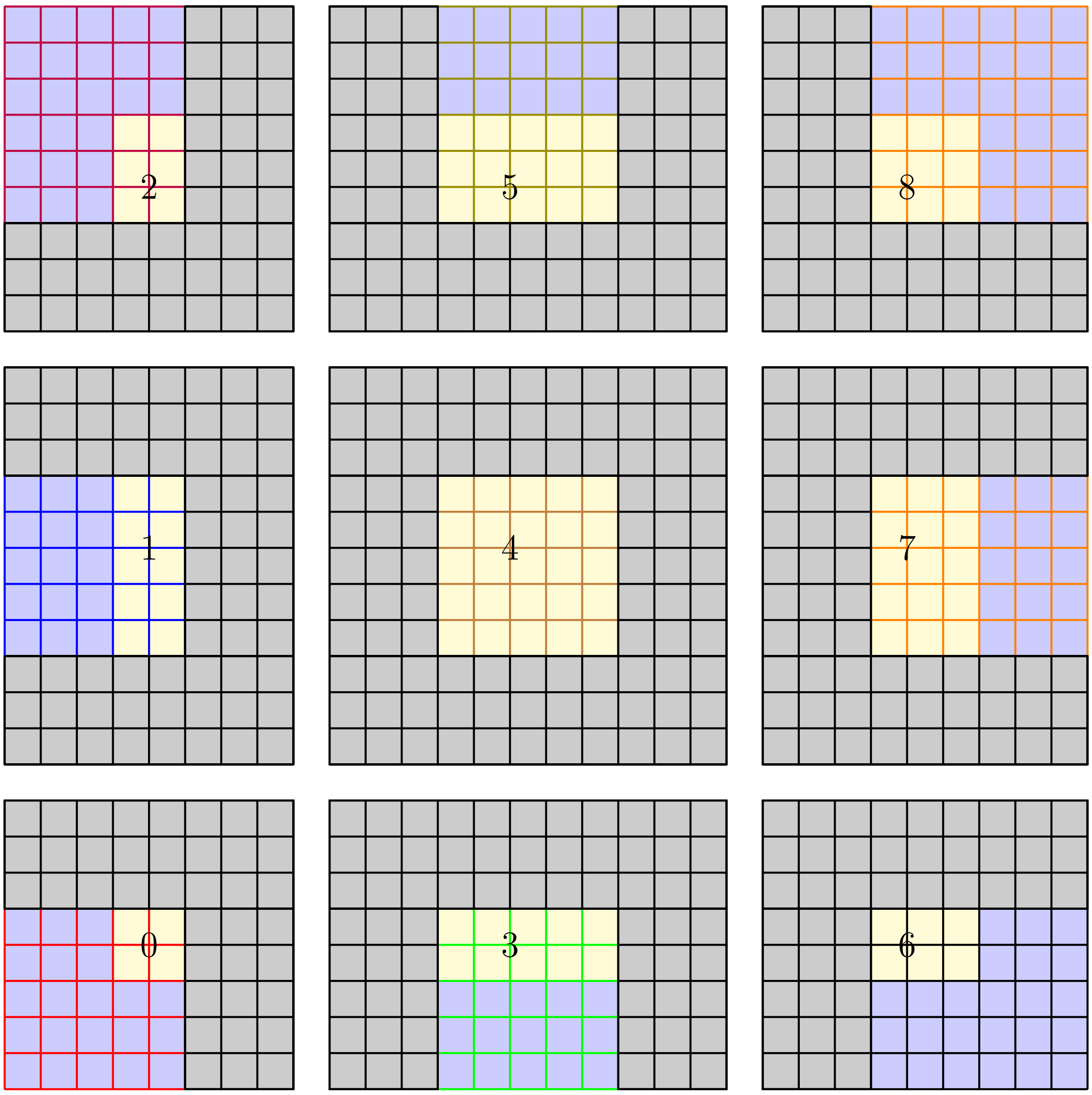
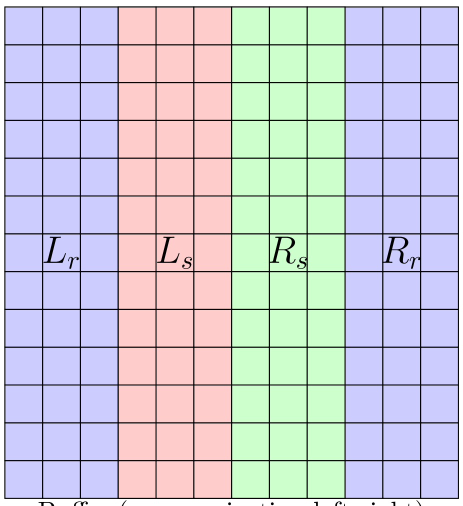
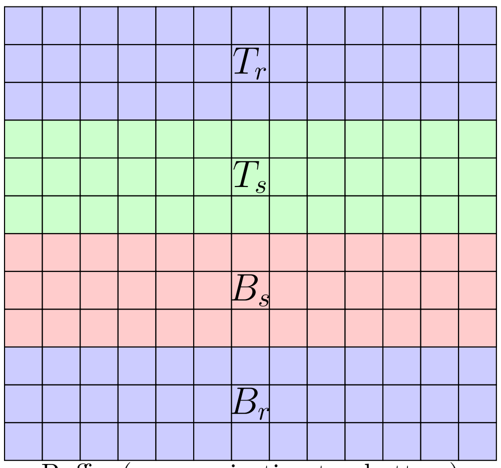


Figure 3: Figure showing the domains with each processor along with the buffer points added for communication between processors.

- Fig.3 shows the compute domains of each processor along with the buffer points.
- The colored parts are corresponding to the grid-points in the original domain.
- The grids colored in black are buffer points added for inter-processor communication.
- The numbering scheme of the processors is also highlighted where for a given number of workers, the rank of the processors follows a linear-indexed scheme where the processor count along the y-direction increases the fastest.(aka column-major ordering)



Buffer (communication left-right)



Buffer (communication top-bottom)

Figure 4: Temporary buffers for single pass communication between the processors.

- Fig.4 shows the structure of buffer created for transfer of information between processors as a single chunk of memory.
- Since, the data structure is in the column-major format, the communication between processors in the x-direction (Left-Right communication) is performed by creating contiguous MPI_derived_type containing all the elements of the structure (struct fields) and the amount of memory corresponding to the grids colored in red and green in the left figure of Fig.4 are sent to the left and right processors respectively.
- For the communication between the top and bottom processors, similarly a buffer is created as shown in the right figure of Fig.4.
- Since, the data to be transferred is now in terms of rows instead of columns, a separate MPI_derived_type using the MPI_Vector type is created with the appropriate data-stride, such that the entire chunks of memory as

shown in colors red and green can be transferred to the bottom and top processors respectively in a single communication.

- The colors depicted in blue in the left and right figures of Fig.4 are utilized for receiving information from the left/right and top/bottom processors respectively.
- The letters L_r R_r stand for Left-receive and Right-receive respectively. Similarly for B_r and T_r that are labels for grids that will receive data from the the bottom and top processors respectively.
- A similar terminology is used for the letters L_s,R_s,B_s and T_s that stand for the sending operations corresponding to the subscript ”s”.