

Phase-field Solver for Eutectic Transformation

Tanmay Dutta

November 10, 2021

Contents

1	2D eutectic growth	1
1.1	Problem specification	1
1.2	Pre-processing	5
1.3	Running the application	17
1.4	Post-processing	18
2	Directional solidification	20
2.1	Problem specification	20
2.2	Pre-processing	20
2.3	Running the case	27
2.4	Post-processing	28
	References	28

This is a phase-field solver pertaining to three phases (α , β and liquid) for OpenFOAM based on finite-volume discretization. The solver can be decomposed into multiple domains for parallel runs. To carry out the simulations, the steps mentioned below can be followed.

1 2D eutectic growth

The first case is conducted by subjecting the melt to a positive moving thermal gradient ahead of the solid-liquid interface in 2D. The imposed temperature field as a function of thermal gradient in the x direction, G and pulling velocity, v is taken as:

$$T(x, t) = T_{initial} + G(x - vt) \quad (1)$$

1.1 Problem specification

The problem is defined as follows:

Solution domain The domain is considered to be two-dimensional in this problem. The rectangular domain consists of three phases with boundaries as shown in Figure 1 (Umate [(2021)]).

Governing equations The basic idea of the phase-field model consists in the description of the evolution of individual phases using the corresponding set of order parameters (the phase-field variables). For each phase, the associated phase-field variable is equal to 1 inside the phase and vanishes to 0 outside the area occupied by the considered phase. The transition of phase fields from 1 to 0 at the phase boundaries is continuous and smooth, i.e., the phase boundaries are represented by diffuse interfaces with a finite thickness. The three phases, α , β and liquid phase are described by three phase fields ϕ_i , with $i = \alpha, \beta, \text{liq}$, representing the volume fractions. The state of the binary eutectic system is described by the order parameter $\Phi = \{\phi_\alpha, \phi_\beta, \phi_{liq}\}$, the chemical potential μ , and the temperature T . The constraints on individual order parameters are:

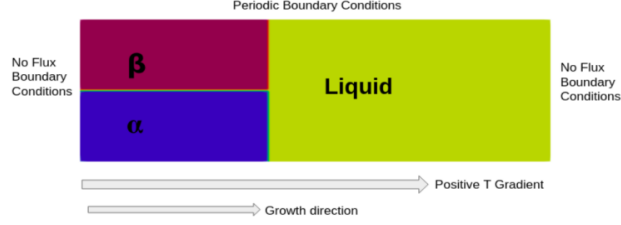


Figure 1: Geometry of the melt with an undercooled region.

$$\phi_i \in [0, 1] \quad \forall i \quad \sum_i \phi_i = 1 \quad (2)$$

- Phase evolution is determined by the phenomenological minimization of the grand-potential functional (Choudhury and Nestler [(2012)]). The evolution equations for three phases can be written as (Folch and Plapp [(2005)]):

$$\begin{aligned} \tau \epsilon \frac{\partial \phi_i}{\partial t} = & -\frac{2}{3} \left[-\epsilon \left(\nabla \cdot \frac{\partial a(\nabla \Phi)}{\partial \nabla \phi_i} \right) + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_i} + \frac{\partial \Psi(T, \mu, \Phi)}{\partial \phi_i} \right] \\ & + \frac{1}{3} \left[-\epsilon \left(\nabla \cdot \frac{\partial a(\nabla \Phi)}{\partial \nabla \phi_j} \right) + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_j} + \frac{\partial \Psi(T, \mu, \Phi)}{\partial \phi_j} \right] \\ & + \frac{1}{3} \left[-\epsilon \left(\nabla \cdot \frac{\partial a(\nabla \Phi)}{\partial \nabla \phi_k} \right) + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_k} + \frac{\partial \Psi(T, \mu, \Phi)}{\partial \phi_k} \right] \end{aligned} \quad (3)$$

where i, j and k are all different and $\epsilon \in \{\alpha, \beta, liq\}$; τ is the relaxation constant which influences the kinetics of the phase transformation at the interfaces.

- The double well potential $w(\Phi)$ can be written as:

$$w(\Phi) = \sum_i 9\gamma \phi_i^2 (1 - \phi_i)^2 \quad (4)$$

- The gradient energy density, $a(\nabla \Phi)$ has the form:

$$a(\nabla \Phi) = \sum_i \gamma |\nabla \phi_i|^2 \quad (5)$$

where γ controls the interfacial energy of the system and is known as surface energy density.

- The grand potential $\Psi(T, \mu, \phi)$ can be expressed as sum of grand potential of phases present in the system:

$$\Psi(T, \mu, \Phi) = \sum_i \Psi_i(T, \mu) h_i(\Phi) \quad (6)$$

where $\Psi_i(T, \mu)$ for the three phases are the following:

$$\Psi_\alpha(T, \mu) = -A \left(\frac{\mu - B_\alpha}{2A} \right)^2 + D_\alpha(T) \quad \Psi_\beta(T, \mu) = -A \left(\frac{\mu - B_\beta}{2A} \right)^2 + D_\beta(T) \quad \Psi_{liq}(T, \mu) = -A \left(\frac{\mu}{2A} \right)^2 \quad (7)$$

where A, B_i and D_i are derived following the consequence of equilibrium between the different phases described by the common tangent construction with parabolic approximation for free energy. These are given below:

$$B_\alpha(T) = 2A_{liq}c_{liq}^{eut} - 2A_\alpha c_\alpha^{eut} + (T - T^{eut}) \left(\frac{2A_{liq}}{m_{liq-\alpha}} - \frac{2A_\alpha}{m_{\alpha-liq}} \right) B_\beta(T) = 2A_{liq}c_{liq}^{eut} - 2A_\beta c_\beta^{eut} + (T - T^{eut}) \left(\frac{2A_{liq}}{m_{liq-\beta}} - \frac{2A_\beta}{m_{\beta-liq}} \right) B \quad (8)$$

where c_i^{eut} are the α , β and liquid phase compositions at the eutectic temperature, T^{eut} . The liquidus slopes for $\alpha - liq$ phase equilibrium is denoted by $m_{liq-\alpha}$ while the corresponding solidus slope by $m_{\alpha-liq}$ and similar notation is followed for the $\beta - liq$ phase equilibrium.

- In equation (6), $h_i(\Phi)$ is an interpolation function given as:

$$h_i = \frac{\phi_i^2}{4} [15(1 - \phi_i) \{1 + \phi_i - (\phi_k - \phi_j)^2\} + \phi_i(9\phi_i^2 - 5)] \quad (9)$$

These interpolation functions make sure that at the interface of phases i and j , the phase field for phase k remains zero.

- The phase-field equations for three phases after incorporating above equations can be given as:

$$\begin{aligned} \tau \epsilon \frac{\partial \phi_i}{\partial t} = & -\frac{2}{3} \left[-2\gamma \epsilon \nabla^2 \phi_i + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_i} + \Psi_i(T, \mu) \frac{\partial h_i(\Phi)}{\partial \phi_i} \right] \\ & + \frac{1}{3} \left[-2\gamma \epsilon \nabla^2 \phi_j + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_j} + \Psi_j(T, \mu) \frac{\partial h_j(\Phi)}{\partial \phi_j} \right] \\ & + \frac{1}{3} \left[-2\gamma \epsilon \nabla^2 \phi_k + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_k} + \Psi_k(T, \mu) \frac{\partial h_k(\Phi)}{\partial \phi_k} \right] \end{aligned} \quad (10)$$

- These three equations for the phase fields corresponding to the liquid, α and β phases are solved in a coupled manner along with mass balance equation which takes the form:

$$\begin{aligned} & \left[\sum_i \frac{\partial c_i}{\partial \mu} h_i(\Phi) \right] \frac{\partial \mu}{\partial t} + \sum_i c_i \left(\sum_n \frac{\partial h_i}{\partial \phi_n} \frac{\partial \phi_n}{\partial t} \right) \\ & = \nabla \cdot \left(D \phi_{liq} \frac{\partial c_{liq}(T, \mu)}{\partial \mu} \nabla \mu \right) - \nabla \cdot j_{at} \end{aligned} \quad (11)$$

The slope of $c-\mu$ curves is expressed according to the parabolic approximation for free energy in the solver source code:

$$\frac{\partial c_i(T, \mu)}{\partial \mu} = \frac{1}{2A} \quad (12)$$

- In this case diffusivity in solid phase is assumed to be negligible. This then becomes the case of one sided diffusion. For the case of one-sided diffusion, it has been shown in various previous works that there exists a thin-interface defect called solute trapping when simulations are performed with interface thicknesses, orders of magnitude larger than those of a real interface. The methodology proposed to correct this effect is the incorporation of an anti-trapping current j_{at} in the evolution equation of the chemical potential. The anti-trapping term is incorporated as an additional flux of solute from the solid to the liquid in the normal direction to the interface (**Karma [(2001)]**). To make sure that the anti-trapping current appears in the first-order correction to the chemical potential, the anti-trapping current is formulated into the following form:

$$j_{at} = j_{at}^{\alpha \Rightarrow liq} \left(-\frac{\nabla \phi_\alpha}{|\nabla \phi_\alpha|} \cdot \frac{\nabla \phi_{liq}}{|\nabla \phi_{liq}|} \right) + j_{at}^{\beta \Rightarrow liq} \left(-\frac{\nabla \phi_\beta}{|\nabla \phi_\beta|} \cdot \frac{\nabla \phi_{liq}}{|\nabla \phi_{liq}|} \right) \quad (13)$$

where $j_{at}^{i \Rightarrow liq}$ is:

$$j_{at}^{i \Rightarrow liq} = -\frac{\epsilon}{2\sqrt{2}}[c_{liq}(\mu) - c_i(\mu)] \frac{\partial \phi_i}{\partial t} \frac{\nabla \phi_i}{|\nabla \phi_i|} \quad (14)$$

All terms in the above equation are used in the nondimensional form, so ϵ is the nondimensional parameter related to the interface width and t is the nondimensional time.

- As the simulation progresses, the solid-liquid interface will choose an undercooling and will traverse with the temperature gradient with the imposed velocity. Since the simulation of solidification begins from an infinite reservoir of liquid, it becomes necessary to implement shifts in all the fields in the domain to ensure that the far field composition of the liquid remains the same as eutectic composition. This is analogous to having a camera frame at the solid liquid interface which is moving along with it into a virtually infinite liquid in the growth direction. This is implemented simply by ‘rigidly convecting’ all the fields. This requires adding the flux term $v \cdot \nabla \phi$, where v is the imposed velocity and ϕ is phase field parameter of all phases and also the chemical potential field. Since, v is a constant, all these fields will simply be translated as it is in the entire domain. This shifts are required to be performed in the growth direction, which is chosen to be x direction. So, each equation becomes:

$$\begin{aligned} \tau \epsilon \frac{\partial \phi_i}{\partial t} - v \frac{\partial \phi_i}{\partial x} = & -\frac{2}{3} \left[-2\gamma \epsilon \nabla^2 \phi_i + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_i} + \Psi_i(T, \mu) \frac{\partial h_i(\Phi)}{\partial \phi_i} \right] \\ & + \frac{1}{3} \left[-2\gamma \epsilon \nabla^2 \phi_j + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_j} + \Psi_j(T, \mu) \frac{\partial h_j(\Phi)}{\partial \phi_j} \right] \\ & + \frac{1}{3} \left[-2\gamma \epsilon \nabla^2 \phi_k + \frac{1}{\epsilon} \frac{\partial w(\Phi)}{\partial \phi_k} + \Psi_k(T, \mu) \frac{\partial h_k(\Phi)}{\partial \phi_k} \right] \end{aligned} \quad (15)$$

and the chemical diffusion evolution equation becomes:

$$\begin{aligned} & \left[\sum_i \frac{\partial c_i}{\partial \mu} h_i(\Phi) \right] \frac{\partial \mu}{\partial t} - v \frac{\partial c}{\partial x} + \sum_i c_i \left(\sum_n \frac{\partial h_i}{\partial \phi_n} \frac{\partial \phi_n}{\partial t} \right) \\ & = \nabla \cdot \left(D \phi_{liq} \frac{\partial c_{liq}(T, \mu)}{\partial \mu} \nabla \mu \right) - \nabla \cdot j_{at} \end{aligned} \quad (16)$$

Since each of the fields is being moved in the direction opposite to growth direction, the temperature gradient is kept stationary and solid-liquid interface is expected to find its equilibrium undercooling.

- Using the relation $\frac{\partial \Psi(T, \mu, \Phi)}{\partial \mu} = -c$ and equations (6–9), ∇c is evaluated.

Boundary conditions

- Cyclic boundary condition is specified at the top and bottom planes, and zero-flux is specified at the remaining planes.

Initial conditions

- $\phi_i = 1.0$ inside the phase i and 0 inside phases j and k (with $i, j, k \in \{\alpha, \beta, liq\}$) (see Figure 1).
- $\mu = 1.0$ in the entire domain.

Physical properties The nondimensionalized physical properties are:

- Slope liquidus of $liq - \alpha$ ($m_{liq-\alpha}$) = Slope solidus of $\alpha - liq$ ($m_{\alpha-liq}$) = -0.5
- Slope liquidus of $liq - \beta$ ($m_{liq-\beta}$) = Slope solidus of $\beta - liq$ ($m_{\beta-liq}$) = 0.5
- Eutectic composition of liquid phase (c_{liq}^{eut}) = 0.5
- Eutectic composition of α phase (c^α) = 0.2
- Eutectic composition of β phase (c^β) = 0.8
- Thermal gradient (G) = 1.7e-5

- Velocity (v) = 0.001
- Diffusivity in liquid (D) = 1
- Eutectic temperature (T_{eut}) = 1.0
- Constant value from temperature profile ($T_{initial}$) = 0.93
- Relaxation coefficient (τ) = 0.144
- Interface energy parameter (γ) = 1
- Interface width parameter (ϵ) = 4

Solver name

- *eutectic*: An implementation for phase-field method to model eutectic transformation of two solid phases from liquid phase.

Case name

- *eutecticGrowth2D*, located in the `$FOAM_RUN/PhaseFieldSolverEutectic` directory.

The problem is solved using *eutectic*, located in the `$FOAM_RUN/PhaseFieldSolverEutectic` directory. The code-guide for this solver is given in the `$FOAM_RUN/PhaseFieldSolverEutectic/codeGuide` directory. Before proceeding further, it may be helpful for the user to become familiar with [OpenFOAM documentation](#).

1.2 Pre-processing

To prepare case files and running the case, the user can change to the case directory:

```
cd $FOAM_RUN/PhaseFieldSolverEutectic/eutecticGrowth2D
```

1.2.1 Mesh generation

OpenFOAM solves the case in three dimensions by default but can be instructed to solve in two dimensions. Here, the mesh must be 1 cell layer thick. The domain consists of a square of side length of 12000 in the x-y plane. Large variations in fields can be expected near the left and bottom boundaries, so the mesh will be graded to be smaller in this region. A graded mesh of 250 by 250 cells will be used initially. The block structure is shown in Figure 2.

The entries in *blockMeshDict* located in the *system* directory for this case are as follows:

```
/*-----*- C++ -*-----*\
| =====|
| \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O p e r a t i o n | Version: 4.0 |
| \ \ / A n d | Web: www.OpenFOAM.org |
| \ \ / M a n i p u l a t i o n |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       blockMeshDict;
}
// * * * * *
convertToMeters 48;
x_min 0;
x_max 250; //direction of growth
```

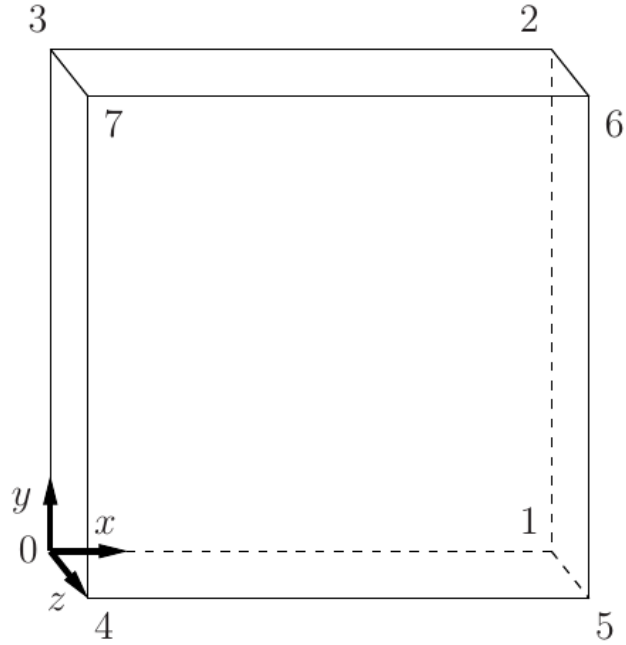


Figure 2: Figure 2: Block structure of the mesh.

```

y_min 0;
y_max 250;
z_min 0;
z_max 2;

// should all be integers
lx #calc "$x_max - $x_min";
ly #calc "$y_max - $y_min";
lz #calc "$z_max - $z_min"; // should all be integers
vertices
(
    ($x_min $y_min $z_min)
    ($x_max $y_min $z_min)
    ($x_max $y_max $z_min)
    ($x_min $y_max $z_min)
    ($x_min $y_min $z_max)
    ($x_max $y_min $z_max)
    ($x_max $y_max $z_max)
    ($x_min $y_max $z_max)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) ($lx $ly 1) simpleGrading (13 13 1)
    // for dx = 0.5: simpleGrading (2 2 2) or convertToMeters = 0.5
);

edges
(

```

```

);

//===== FOR ZERO-FLUX BOUNDARY CONDITIONS =====
boundary
(
    floor
    {
        type cyclic;
        faces
        (
            (0 1 5 4)
        );
    }

    ceiling
    {
        type cyclic;
        faces
        (
            (2 3 7 6)
        );
    }
    sideSolid
    {
        type wall;
        faces
        (
            (0 3 2 1)
            (0 4 7 3)
            (4 5 6 7)
            (1 2 6 5)
        );
    }
);

mergePatchPairs
(

);

// *****

The mesh is generated after accessing blockMesh entry within $FOAM_RUN/PhaseFieldSolverEutectic/eutecticGrowth2D/Allrun

#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

application=`getApplication`

runApplication blockMesh

```

```

cp 0/phi.orig 0/phi_alpha
cp 0/phi.orig 0/phi_beta
cp 0/phi.orig 0/phi_liq
cp 0/phi.orig 0/T
cp 0/mu.orig 0/mu

runApplication setFields -dict system/setFieldsDict

runApplication decomposePar

#runApplication `getApplication`
runParallel `getApplication`

#runApplication reconstructPar

#-----
The logs are stored into log.blockMesh.

```

1.2.2 Boundary and initial conditions

Once the mesh generation is complete, the user can look at this initial conditions set up for this case. The case is set up to start at time $t = 0$ s, so the initial field data is stored in a *0* sub-directory of the *freeGrowth* directory. The *0* sub-directory contains *phi_alpha.orig*, *phi_beta.orig*, *phi_liq.orig*, and *mu.orig*, one for each of the phase parameter ($\phi_\alpha, \phi_\beta, \phi_{liq}$) and chemical potential (μ) fields whose initial values and boundary conditions must be set. First, the file *phi_alpha.orig* can be examined:

```

/*-----*- C++ -*-----*\
| =====|
| \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O p e r a t i o n | Version: 4.x |
| \ \ / A n d | Web: www.OpenFOAM.org |
| \ \ / M a n i p u l a t i o n | |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       phi_alpha;
}
// *****

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

//===== NO-FLUX BOUNDARY CONDITIONS =====//

```



```

boundaryField
{
    floor
    {
        type            cyclic;
    }
    ceiling
    {
        type            cyclic;
    }
    sideSolid
    {
        type            zeroGradient;
    }
}

```

For this case, the boundary consists of planes split into patches named: (1) floor and ceiling for cyclic boundary at top and bottom planes; (2) sideSolid patch includes other planes of the 2D case and is given a zeroGradient boundary condition for ϕ_{alpha} , meaning “the normal gradient of ϕ_{α} is zero.” The initial fields are set to uniform zero.

The user can similarly examine *0/phi_beta.orig*, *0/phi_liq.orig*, and the chemical potential field in the *0/mu.orig* file. The non-dimensionalized internal field is initialised as uniform zero. The boundary fields require the same boundary condition.

```

/*-----*- C++ -*-----*\
| =====|
| \\\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\\ / O peration | Version: 4.x |
| \\\ / A nd | Web: www.OpenFOAM.org |
| \\\ / M anipulation | |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       phi_beta;
}
// * * * * *

dimensions      [0 0 0 0 0 0 0];

internalField    uniform 0;

//===== NO-FLUX BOUNDARY CONDITIONS =====//

boundaryField
{
    floor
    {
        type            cyclic;
    }
}

```

```

        ceiling
        {
            type            cyclic;
        }
    sideSolid
    {
        type            zeroGradient;
    }
}

/*-----*- C++ -*-----*\
| =====|
| \\\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\\ / O peration | Version: 4.x |
| \\\ / A nd | Web: www.OpenFOAM.org |
| \\\ / M anipulation |
\*-----*- C++ -*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       phi_liq;
}
// *****

dimensions      [0 0 0 0 0 0 0];

internalField    uniform 0;

//===== NO-FLUX BOUNDARY CONDITIONS =====//

boundaryField
{
    floor
    {
        type            cyclic;
    }
    ceiling
    {
        type            cyclic;
    }
    sideSolid
    {
        type            zeroGradient;
    }
}

/*-----*- C++ -*-----*\
| =====|
| \\\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\\ / O peration | Version: 4.x |

```

```

|  \ \  /  A nd      | Web:      www.OpenFOAM.org      |
|  \ \ /  M anipulation | | | | | | | | | | | | | | | | |
\*-----*
FoamFile
{
    version      2.0;
    format        ascii;
    class         volScalarField;
    location      "0";
    object        mu;
}
// * * * * *

dimensions      [0 0 0 0 0 0 0];

internalField    uniform 0;

//===== NO-FLUX BOUNDARY CONDITIONS =====//

boundaryField
{
    floor
    {
        type      cyclic;
    }
    ceiling
    {
        type      cyclic;
    }
    sideSolid
    {
        type      zeroGradient;
    }
}
}

```

1.2.3 Setting initial field

A non-uniform initial condition is specified for the phase parameter, ϕ_i , where $\phi_i = 1.0$ inside the phase i and 0 inside phases j and k (with $i, j, k \in \{\alpha, \beta, liq\}$). And for chemical potential, μ , where $\mu = 1.0$ in the entire domain.

This is achieved by running the setFields utility. It requires a *setFieldsDict* dictionary, located in the *system* directory, whose entries for this case are shown below:

```

\*-----*- C++ -*-----*\
| ===== |
| \ \  /  F ield      | OpenFOAM: The Open Source CFD Toolbox | | | | | | | | | | | | | | | |
| \ \ /  O peration   | Version:  4.0                        |
| \ \  /  A nd        | Web:      www.OpenFOAM.org           |
| \ \ /  M anipulation | | | | | | | | | | | | | | | | |
\*-----*
FoamFile
{
    version      2.0;

```

```

        format      ascii;
        class       dictionary;
        location     "system";
        object       setFieldsDict;
    }
// * * * * *

defaultFieldValues
(
    volScalarFieldValue phi_alpha 0.0
    volScalarFieldValue phi_beta 0.0
    volScalarFieldValue phi_liq 0.0
    volScalarFieldValue T 0.93
    volScalarFieldValue mu 1.0

);

regions
(

    boxToCell // boxToCell: work on all cells in a rectangular box defined with starting and end points
    {
        box (0 0 -0.12) (500 24000 0.12); // Box can be larger than domain; in general this avoids edge effects
        fieldValues // NB: no semicolons at the end of the dictionary entries below!
        (
            volScalarFieldValue phi_alpha 1.0
            volScalarFieldValue phi_beta 0.0
            volScalarFieldValue phi_liq 0.0
            volScalarFieldValue T 0.93
            volScalarFieldValue mu 1.0
        );
    }

    boxToCell // boxToCell: work on all cells in a rectangular box defined with starting and end points
    {
        box (0 0 -0.12) (500 24000 0.12); // Box can be larger than domain; in general this avoids edge effects
        fieldValues // NB: no semicolons at the end of the dictionary entries below!
        (
            volScalarFieldValue phi_alpha 0.0
            volScalarFieldValue phi_beta 1.0
            volScalarFieldValue phi_liq 0.0
            volScalarFieldValue T 0.93
            volScalarFieldValue mu 1.0
        );
    }

    boxToCell // boxToCell: work on all cells in a rectangular box defined with starting and end points
    {
        box (0 0 -0.12) (500 24000 0.12); // Box can be larger than domain; in general this avoids edge effects
        fieldValues // NB: no semicolons at the end of the dictionary entries below!
        (
            volScalarFieldValue phi_alpha 0.0

```

```

        volScalarFieldValue phi_beta 0.0
        volScalarFieldValue phi_liq 1.0
        volScalarFieldValue T 0.93
        volScalarFieldValue mu 1.0
    );
}

);

// *****

The user can execute setFields through $FOAM_RUN/PhaseFieldSolverEutectic/eutecticGrowth2D/Allrun.
The logs are stored into log.setFields.

```

1.2.4 Physical properties

The physical properties for the case are specified in *constant/transportProperties* dictionary. The entries of *transportProperties* dictionary are shown below:

```

/*-----*- C++ -*-----*\
| =====|
| \ \ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O peration | Version: 2.1.x |
| \ \ / A nd | Web: www.OpenFOAM.org |
| \ \ / M anipulation | |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "constant";
    object       transportProperties;
}
// * * * * *

dimx      dimx [0 1 0 0 0 0 0] 1; //Dimension of position
dimt      dimt [0 0 1 0 0 0 0] 1; //Dimension of time
ms_alpha  ms_alpha [0 0 0 0 0 0 0] -0.5; //Slope liq-alpha and alpha-liq
ms_beta   ms_beta [0 0 0 0 0 0 0] 0.5; //Slope liq-beta and beta-liq
c_eq_liq  c_eq_liq [0 0 0 0 0 0 0] 0.5; //Eutectic composition of liquid phase
c_eq_alpha c_eq_alpha [0 0 0 0 0 0 0] 0.2; //Eutectic composition of alpha phase
c_eq_beta c_eq_beta [0 0 0 0 0 0 0] 0.8; //Eutectic composition of beta phase
G         G [0 0 0 0 0 0 0] 1.7e-5; //Thermal gradient
v         v [0 0 0 0 0 0 0] 0.001; //Velocity
A         A [0 0 0 0 0 0 0] 1.0;
D         D [0 0 0 0 0 0 0] 1.0; //Diffusivity in liquid
T_eut     T_eut [0 0 0 0 0 0 0] 1.0; //Eutectic temperature
initial   initial [0 0 0 0 0 0 0] 0.93; //Constant value from temperature profile
tau       tau [0 0 0 0 0 0 0] 0.144; //Relaxation coefficient
gamma     gamma [0 0 0 0 0 0 0] 1.0; //Interface energy parameter
epsilon   epsilon [0 0 0 0 0 0 0] 4.0; //Interface width parameter
// *****

```

The physical properties are read by *readTransportProperties.H* while running the case.

1.2.5 Time step control

Input data relating to the control of time, reading and writing of the solution data are read from the *controlDict* dictionary located in the *system* directory.

The run is started at time $t = 0$ to read field data from a directory named θ . Therefore, the *startFrom* keyword is set to *startTime* and then the *startTime* keyword to be 0.

The end time can be considered to be the time taken for the primary dendrites to reach the boundaries, which is found to be 3000000 for this case. To specify this end time, the *stopAt* keyword is set to *endTime* and then the *endTime* keyword to 3000000.

Next, the time step must be fixed which is represented by the keyword *deltaT*. To achieve temporal accuracy and numerical stability while reducing computational effort, *deltaT* is set to 200.

As the simulation progresses, results written at certain intervals of time can later be viewed with a post-processing package. The *writeControl* keyword can be set to the *runTime* option, where the time interval for writing is specified under the *writeInterval* keyword. The *writeInterval* can be set to 100000. OpenFOAM creates a new directory named after the current time, e.g. 500000, on each occasion that it writes a set of data containing the results for each field, ϕ_i and μ , into the time directories. For this case, the entries in the *controlDict* are shown below:

```

/*-----*- C++ -*-----*\
| =====|
| \ \ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O peration | Version: 4.0 |
| \ \ / A nd | Web: www.OpenFOAM.org |
| \ \ / M anipulation | |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       controlDict;
}
// * * * * *

application    eutectic;

startFrom      startTime;

startTime      0; // the actual time i.e. runtime.value() or no. of iterations*deltaT

stopAt         endTime;

endTime        6000000; //actual time = total_iterations*deltaT

deltaT         200;

writeControl    runTime;

writeInterval   200000;

```

```

purgeWrite      0;

writeFormat      ascii;

writePrecision   6;

writeCompression off;

timeFormat       general;

timePrecision    6;

runTimeModifiable true;

```

```
// ***** //
```

1.2.6 Discretisation schemes

In this case, the term $\nabla \cdot \left(D\phi_{liq} \frac{\partial c_{liq}(T, \mu)}{\partial \mu} \nabla \mu \right)$ in the chemical potential equation includes laplacian(D*phi_liq,mu). The laplacian is solved by fixing laplacianSchemes to Gauss linear corrected. The gradient and divergence are solved by fixing gradSchemes and divSchemes to Gauss linear.

The other discretised terms use commonly employed schemes so that the *fvSchemes* dictionary entries should therefore be:

```

/*-----*- C++ -*-----*\
| ===== |
| \ \      / F ield      | OpenFOAM: The Open Source CFD Toolbox |
| \ \      / O peration  | Version: 4.0 |
| \ \      / A nd        | Web: www.OpenFOAM.org |
| \ \ /    M anipulation | |
\*-----*-*/

```

```

FoamFile
{
    version      2.0;
    format        ascii;
    class         dictionary;
    location      "system";
    object        fvSchemes;
}
// ***** //

```

```

ddtSchemes
{
    default      Euler;
}

gradSchemes
{
    default      Gauss linear;
    grad(phi)    Gauss linear;
}

divSchemes

```

```

{
    default          Gauss linear;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
}

snGradSchemes
{
    default          corrected;
}

// ***** //

```

1.2.7 Solver settings

In the *fvSolution*, the solver tolerance should be set to 10^{-6} for this case. The solver relative tolerance, denoted by *relTol*, is set to 0.

```

/*-----*- C++ -*-----*\
| ===== |
|  \ \      /  F ield      | OpenFOAM: The Open Source CFD Toolbox |
|  \ \      /  O peration  | Version: 4.0 |
|  \ \      /  A nd        | Web: www.OpenFOAM.org |
|  \ \      /  M anipulation | |
\*-----*-*/

FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       fvSolution;
}

// ***** //

```

```

solvers
{
    mu
    {
        solver          smoothSolver;
        preconditioner   DIC;
        smoother         symGaussSeidel;
        tolerance        1e-6;
        relTol           0.0;
    }
}

```



```

phi_alpha
{
    solver                smoothSolver;
        preconditioner    DIC;
        smoother          symGaussSeidel;
        tolerance         1e-6;
        relTol             0.0;
}

phi_beta
{
    solver                smoothSolver;
        preconditioner    DIC;
        smoother          symGaussSeidel;
        tolerance         1e-6;
        relTol             0.0;
}

phi_liq
{
    solver                smoothSolver;
        preconditioner    DIC;
        smoother          symGaussSeidel;
        tolerance         1e-6;
        relTol             0.0;
}
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}

```

```
// ***** //
```

The *fvSolution* dictionary contains a sub-dictionary, SIMPLE that contains a control parameter `nNonOrthogonalCorrectors` set to 0. The description of other options can be found in the OpenFOAM userguide.

1.3 Running the application

The first step to run a parallel case is to decompose the domain using the `decomposePar` utility for assigning to different processors. The dictionary associated with `decomposePar`, *decomposeParDict* is located in the *system* directory. The first entry is `numberOfSubdomains` corresponding to the number of processors available for the case. The method of decomposition can be simple. The domain is split into subdomains, in the x, y and z directions, the number of subdomains in each direction being given by the vector `n`. As this geometry is two dimensional, the 3rd direction, z, cannot be split, hence n_z must be 1. The n_x and n_y components of `n` split the domain in the x and y directions following $n_x n_y = \text{numberOfSubdomains}$. To minimize the communication time between the processors, the number of cell faces adjoining the subdomains are kept to a minimum. For growth direction parallel to x, the domain is split in the y direction to have optimum load balance. The `delta` keyword is set to 0.001.

For this case, `numberOfSubdomains` = 4 and `n` = (1, 4, 1). When executing *Allrun*, `decomposePar` is run. The logs are stored into *log.decomposePar*.

```

/*-----*-- C++ -*-----*\
| ===== |
| \ \      / F ield      | OpenFOAM: The Open Source CFD Toolbox |
| \ \      / O peration  | Version: 4.0 |
| \ \      / A nd        | Web:      www.OpenFOAM.org |
|  \ \     M anipulation | |
\*-----*--\
FoamFile
{
    version      2.0;
    format        ascii;
    class         dictionary;
    note          "mesh decomposition control dictionary";
    location      "system";
    object        decomposeParDict;
}

// * * * * *

numberOfSubdomains 4;

//- Keep owner and neighbour on same processor for faces in zones:
// preserveFaceZones (heater solid1 solid3);

//method          scotch;
method            simple;

simpleCoeffs
{
    n              (1 4 1);
    delta          0.001;
}

```

After compiling the solver (see *codeGuide*), it is executed parallelly using `runParallel` within *Allrun*. The progress of the job is stored into *log.phaseFieldDynamic*. It includes the current time, initial and final residuals for all fields.

1.4 Post-processing

Once the case has completed running, the decomposed fields and mesh from processor directories can be reconstructed for post-processing. Using `reconstructPar`, the decomposed mesh with fields can be reassembled. The results reconstructed to time directories can be viewed using ParaView by creating and opening *eutecticGrowth2D.foam* case module:

```

cd $FOAM_RUN/PhaseFieldSolverEutectic/eutecticGrowth2D
touch eutecticGrowth2D.foam
paraview eutecticGrowth2D.foam

```

The evolution of phase-field profile corresponding to three time steps are shown in Figure 4-6.

As can be seen from the figures, in absence of thermal gradient, the solidification happens in the form of freely growing dendrites ([Choudhury and Nestler \[\(2012\)\]](#)). Also, the chemical potential plot at time 2500000 is shown in Figure 7.



Figure 3: Figure 4: Phase-field profile at time 100000.

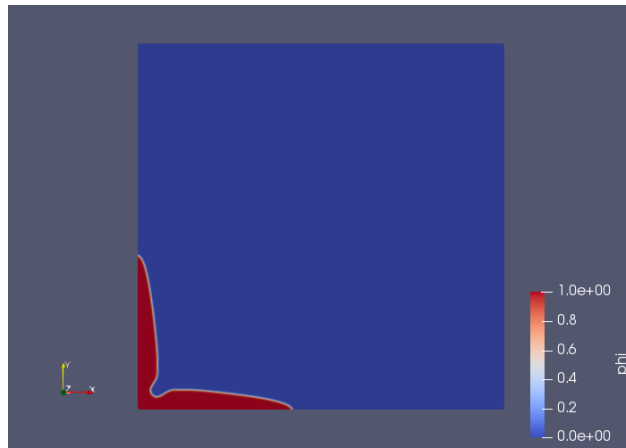


Figure 4: Figure 5: Phase-field profile at time 1000000.

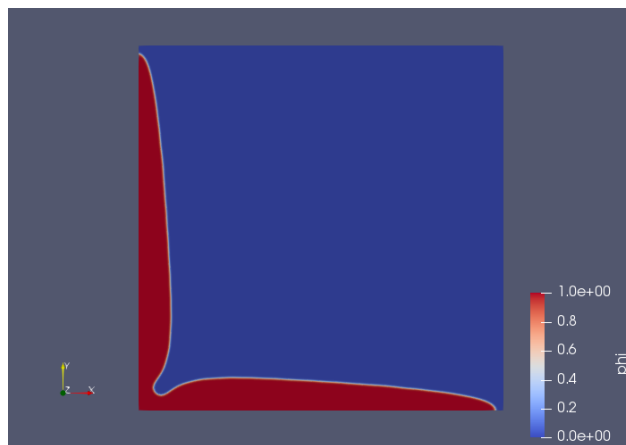


Figure 5: Figure 6: Phase-field profile at time 2500000.

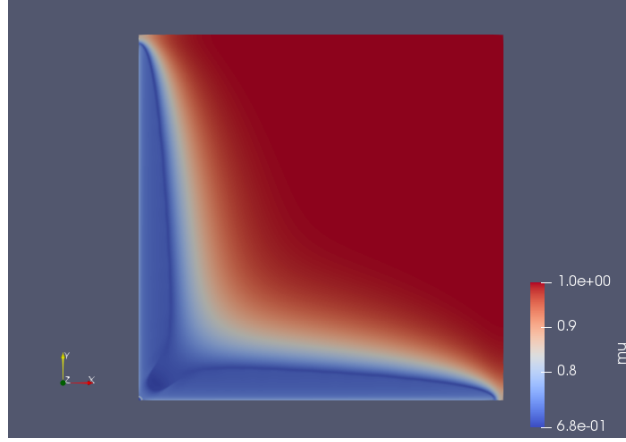


Figure 6: Figure 7: Chemical potential at time 2500000.

2 Directional solidification

The second case is performed by subjecting the melt to a positive thermal gradient ahead of the solid-liquid interface. The imposed temperature field as a function of thermal gradient in the x direction, G and pulling velocity, v is taken as:

$$T(x, t) = T_{initial} + G(x - vt) \quad (167)$$

2.1 Problem specification

Solution domain The domain is considered to be two-dimensional in this problem. The rectangular domain consists of an undercooled region as shown in Figure 8.

Governing equations The governing equations used are same as in section 1.1.

Boundary conditions Zero-flux is specified at all planes.

Initial conditions The initial conditions are same as in section 1.1.

Physical properties The physical properties are same as in section 1.1 except the following (in order to impose thermal gradient and velocity in the x direction):

- Thermal gradient (G) = 1.7e-5
- Velocity (v) = 0.001

Solver name *phaseFieldDynamic*: An implementation with dynamic interface refinement for phase-field method to model solidification of two-phase two-component system.

Case name *directionalSolidification*, located in the `$FOAM_RUN/PhaseFieldSolverDynamic` directory.

2.2 Pre-processing

To prepare case files and running the case, the user can change to the case directory:

```
cd $FOAM_RUN/PhaseFieldSolverDynamic/directionalSolidification
```

2.2.1 Mesh generation

The overall process of generating the mesh remains same as in section 1.2.1. However, this case is a symmetric problem. So, only the top half of the whole geometry (shown in Figure 8) is considered. Similarly, graded

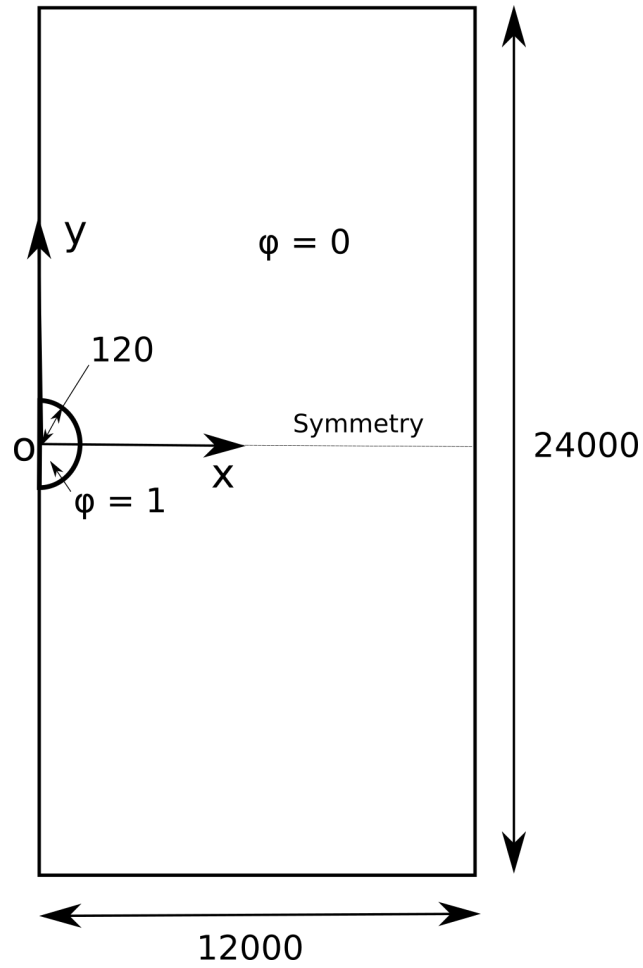


Figure 7: Figure 8: Geometry of the melt with an undercooled region.

mesh of 250 by 250 cells (with smaller cells near the left and bottom boundaries) will be used. The entries in *blockMeshDict* located in the *system* directory for this case are as follows:

```

/*-----*- C++ -*-----*/
| =====|
| \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O p e r a t i o n | Version: 4.0 |
| \ \ / A n d | Web: www.OpenFOAM.org |
| \ \ / M a n i p u l a t i o n |
/*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       blockMeshDict;
}
// * * * * *
convertToMeters 48;
x_min 0;
x_max 250; //direction of growth
y_min 0;
y_max 250;
z_min 0;
z_max 2;

// should all be integers
lx #calc "$x_max - $x_min";
ly #calc "$y_max - $y_min";
lz #calc "$z_max - $z_min"; // should all be integers
vertices
(
    ($x_min $y_min $z_min)
    ($x_max $y_min $z_min)
    ($x_max $y_max $z_min)
    ($x_min $y_max $z_min)
    ($x_min $y_min $z_max)
    ($x_max $y_min $z_max)
    ($x_max $y_max $z_max)
    ($x_min $y_max $z_max)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) ($lx $ly 1) simpleGrading (13 13 1)
// for dx = 0.5: simpleGrading (2 2 2) or convertToMeters = 0.5
);

edges
(
);

//===== FOR ZERO-FLUX BOUNDARY CONDITIONS =====
boundary

```

```

(
    floor
    {
        type cyclic;
        faces
        (
            (0 1 5 4)
        );
    }

    ceiling
    {
        type cyclic;
        faces
        (
            (2 3 7 6)
        );
    }
    sideSolid
    {
        type wall;
        faces
        (
            (0 3 2 1)
            (0 4 7 3)
            (4 5 6 7)
            (1 2 6 5)
        );
    }
);

mergePatchPairs
(

);

// *****

```

2.2.2 Boundary and initial conditions

Similar to section 1.2.2, the file *phi.orig* in *0* sub-directory of the *directionalSolidification* directory can be examined:

```

/*-----*- C++ -*-----*\
| =====|
| \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O p e r a t i o n | Version: 4.x |
| \ \ / A n d | Web: www.OpenFOAM.org |
| \ \ / M a n i p u l a t i o n | |
\*-----*/
FoamFile
{
    version 2.0;
    format ascii;

```

```

        class      volScalarField;
        location    "0";
        object      phi_alpha;
    }
    // * * * * *

    dimensions      [0 0 0 0 0 0 0];

    internalField    uniform 0;

    //===== NO-FLUX BOUNDARY CONDITIONS =====//

    boundaryField
    {
        floor
        {
            type      cyclic;
        }
        ceiling
        {
            type      cyclic;
        }
        sideSolid
        {
            type      zeroGradient;
        }
    }
}

```

For this case, the boundary consists of symmetryPlane, only for patch named floor for the bottom symmetric boundary. The remaining planes of the 2D case are given a zeroGradient boundary condition.

The boundary field for chemical potential requires the same boundary condition.

```

/*-----*- C++ -*-----*\
| ===== |
| \ \      / F i e l d      | OpenFOAM: The Open Source CFD Toolbox |
| \ \      / O p e r a t i o n | Version: 4.x |
| \ \      / A n d             | Web:      www.OpenFOAM.org |
| \ \      / M a n i p u l a t i o n | |
\*-----*-*/

FoamFile
{
    version      2.0;
    format        ascii;
    class        volScalarField;
    location      "0";
    object        mu;
}
// * * * * *

dimensions      [0 0 0 0 0 0 0];

internalField    uniform 0;

```



```
//===== NO-FLUX BOUNDARY CONDITIONS =====//
```

```
boundaryField
{
    floor
    {
        type            cyclic;
    }
    ceiling
    {
        type            cyclic;
    }
    sideSolid
    {
        type            zeroGradient;
    }
}
```

2.2.3 Setting initial field

The entries of *setFieldsDict* dictionary for this case are kept same as section 1.2.3.

2.2.4 Physical properties

The entries of *transportProperties* dictionary are shown below:

```
/*-----*- C++ -*-----*\
| ===== |
| \\\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\\ / O peration | Version: 2.1.x |
| \\\ / A nd | Web: www.OpenFOAM.org |
| \\\ / M anipulation | |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "constant";
    object       transportProperties;
}
// * * * * *

dimx      dimx [0 1 0 0 0 0 0] 1; //Dimension of position
dimt      dimt [0 0 1 0 0 0 0] 1; //Dimension of time
ms_alpha  ms_alpha [0 0 0 0 0 0 0] -0.5; //Slope liq-alpha and alpha-liq
ms_beta   ms_beta [0 0 0 0 0 0 0] 0.5; //Slope liq-beta and beta-liq
c_eq_liq  c_eq_liq [0 0 0 0 0 0 0] 0.5; //Eutectic composition of liquid phase
c_eq_alpha c_eq_alpha [0 0 0 0 0 0 0] 0.2; //Eutectic composition of alpha phase
c_eq_beta c_eq_beta [0 0 0 0 0 0 0] 0.8; //Eutectic composition of beta phase
G         G [0 0 0 0 0 0 0] 1.7e-5; //Thermal gradient
v         v [0 0 0 0 0 0 0] 0.001; //Velocity
A         A [0 0 0 0 0 0 0] 1.0;
D         D [0 0 0 0 0 0 0] 1.0; //Diffusivity in liquid
```

```

T_eut      T_eut [0 0 0 0 0 0 0] 1.0; //Eutectic temperature
initial    initial [0 0 0 0 0 0 0] 0.93; //Constant value from temperature profile
tau        tau [0 0 0 0 0 0 0] 0.144; //Relaxation coefficient
gamma      gamma [0 0 0 0 0 0 0] 1.0; //Interface energy parameter
epsilon    epsilon [0 0 0 0 0 0 0] 4.0; //Interface width parameter
// *****

```

It must be noted that, thermal gradient, G and velocity, v are nonzero for directional solidification. The remaining entries are same as section 1.2.4.

2.2.5 Time step control

Similar to section 1.2.5, the startTime and deltaT are specified. The endTime keyword is set to 6000000 to observe the growth of secondary dendrites from the primary dendrite, growing normal to the direction of imposed thermal gradient. The writeInterval can be set to 200000. For this case, the entries in the *controlDict* are shown below:

```

/*-----*- C++ -*-----*\
| =====|
| \ \ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \ \ / O peration | Version: 4.0 |
| \ \ / A nd | Web: www.OpenFOAM.org |
| \ \ / M anipulation |
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       controlDict;
}
// *****

application    eutectic;

startFrom      startTime;

startTime      0; // the actual time i.e. runtime.value() or no. of iterations*deltaT

stopAt         endTime;

endTime        6000000; //actual time = total_iterations*deltaT

deltaT         200;

writeControl    runTime;

writeInterval   200000;

purgeWrite     0;

writeFormat     ascii;

writePrecision  6;

```

```

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// ***** //

```

2.2.6 Discretisation schemes and solver settings

The *fvSchemes* and *fvSolution* dictionary entries are kept same as section 1.2.6 and 1.2.7.

2.3 Running the case

For this case, the primary dendrite can be expected to grow in the y direction since the thermal gradient is applied in the x direction. Consequently, the secondary dendrites will grow almost uniformly in the x direction (as will be seen later in section 2.5). In order to balance the load between the processors, in the presence of dynamic interface refinement and without stopping the run unlike section 1.3, the subdomains are divided in the y direction. Hence, `numberOfSubdomains = 4` and `n = (1, 4, 1)` are set.

```

/*-----*- C++ -*-----*\
| ===== |
| \ \      / F ield      | OpenFOAM: The Open Source CFD Toolbox |
| \ \      / O peration  | Version: 4.0 |
| \ \      / A nd        | Web: www.OpenFOAM.org |
| \ \      / M anipulation | |
\*-----*-*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    note         "mesh decomposition control dictionary";
    location     "system";
    object       decomposeParDict;
}

// * * * * * //

numberOfSubdomains 4;

//- Keep owner and neighbour on same processor for faces in zones:
// preserveFaceZones (heater solid1 solid3);

//method      scotch;
method        simple;

simpleCoeffs
{
    n          (1 4 1);
    delta      0.001;
}

```

}

Similar to section 1.3, the solver for this case is run parallelly using `runParallel` within *Allrun*.

2.4 Post-processing

Similar to section 1.4, the results reconstructed to time directories can be viewed by creating and opening the *directionalSolidification.foam* case module using ParaView. The evolution of phase-field profile corresponding to three time steps are shown in Figure 9-11.

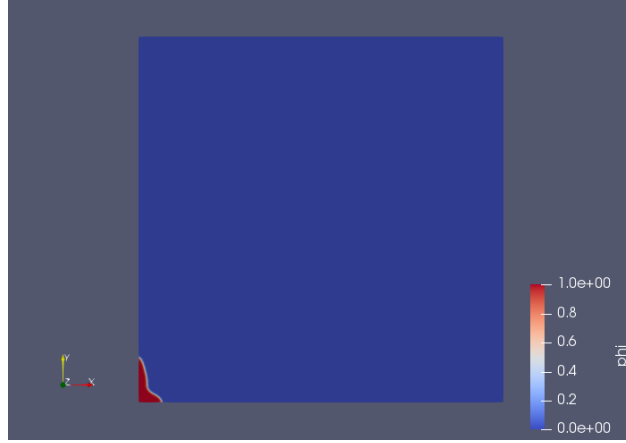


Figure 8: Figure 9: Phase-field profile at time 200000.

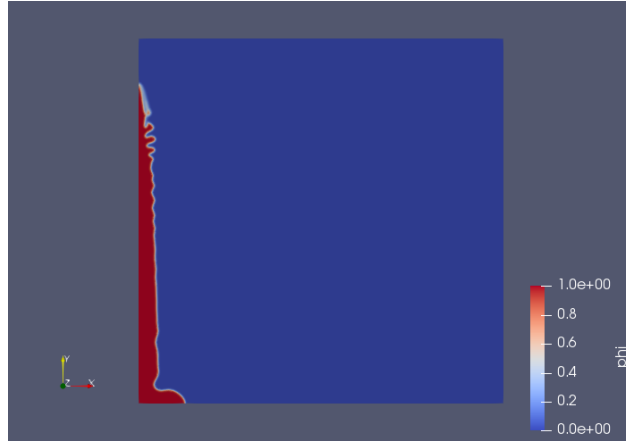


Figure 9: Figure 10: Phase-field profile at time 800000.

As can be seen from the figures, in the presence of thermal gradient and velocity in the x direction, the primary dendrite grows in the y direction. Later, secondary dendrites start to grow from the primary dendrite unlike the *freeGrowth* case in section 1.4. Also, the chemical potential plot at time 3600000 is shown in Figure 12.

References

[Choudhury and Nestler(2012)] A Choudhury and B Nestler. Grand-potential formulation for multicomponent phase transformations combined with thin-interface asymptotics of the double-obstacle potential. *Physical review. E, Statistical, nonlinear, and soft matter physics*. 85:021602, 2012.



Figure 10: Figure 11: Phase-field profile at time 3600000.

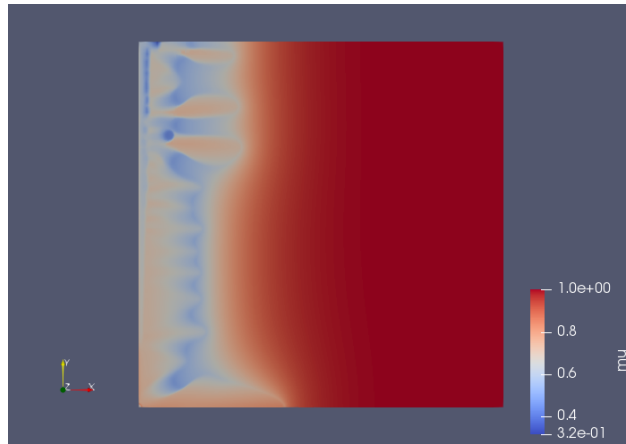


Figure 11: Figure 12: Chemical potential at time 3600000.

- [Folch and Plapp(2005)] R Folch and M Plapp. Quantitative phase-field modeling of two-phase growth. *Phys. Rev. E*. 72:011602, 2005.
- [Karma(2001)] A Karma. Phase-field formulation for quantitative modeling of alloy solidification. *Phys. Rev. Lett.* 87:115701, 2001.
- [Umate(2021)] K Umate. Development of framework for eutectic solidification in OpenFOAM. *IISc Thesis*.