NUMERICAL SIMULATION OF DENDRITES GROWTH IN CONTINUOUS CASTING BY USING OPEN SOURCE SOFTWARE

by

Xiang Zhou

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering

Department of Mechanical and Civil Engineering

Hammond, Indiana

August 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Chenn Q. Zhou, Chair

Department of Mechanical and Civil Engineering

Dr. Hansung Kim

Department of Mechanical and Civil Engineering

Dr. Xiuling Wang

Department of Mechanical and Civil Engineering

Approved by:

Dr. Chenn Q. Zhou

To the people I love

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to those who encouraged me to complete this thesis. First, a greatest gratitude I give to my academic advisor, Prof. Chenn Q Zhou. She offered the opportunity for me to study at Purdue. Only with her careful guidance and encouragement, I can successfully complete the thesis. I would like to thank Prof. Hansung Kim and Prof. Xiuling Wang for serving on my advisory committee and providing valuable suggestions and research experiences during the courses and regular meetings.

I would like to thank the colleagues in the Center for Innovation through Visualization and Simulation (CIVS) at Purdue University Northwest for their kind help and encourage to me. A special gratitude I would like to give to my CIVS mentor, Haibo Ma, whose professional advice and a wealth of knowledge help me a lot during the study of the thesis. Also, I would like to thank him for his co-supervision and forgiving me for many mistakes. Without his help, I will not be on the right track in this project.

I would like to thank Prof. David Montiel, doctoral student Zhenjie Yao from University of Michigan and one of my best friends Liyang Qin for helping me a lot in the field of computer programming and phase field method.

I would like to acknowledge support from the Steel Manufacturing Simulation and Visualization Consortium (SMSVC) throughout the course of this research.

I would like to thank my parents. Thank them for always supporting me to complete my studies on both physical and mental aspects.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

φ	phase field variable
τ	interface relaxation time
W	interface layer width
$\bar{W}$	average interface layer width
m	free energy driving force
θ	angle between interface normal direction and positive x-axis direction
T	temperature
T_0	initial temperature
T_e	equilibrium temperature
α	coupling constant
γ	coupling constant
a_1	coupling coefficient
a_2	coupling coefficient
ε	anisotropy strength
K	dimensionless latent heat
j	mode number
R_1	residual function 1
R_2	residual function 2
a_s	anisotropy term
d_0	solute capillary length

λ	coupling constant
λ_1	primary dendrite arm spacing
c_0	initial concentration
θ_0	initial rotational angle
U	dimensionless supersaturation
D	diffusion coefficient
D_s	diffusion coefficient in solid
D_l	diffusion coefficient in liquid
G	temperature gradient
R	cooling rate
m_l	liquidus slope
V_p	pulling speed
$\tilde{y}$	dimensionless location in y axis
$\widetilde{V_p}$	dimensionless pulling speed
$\tilde{t}$	dimensionless time
$\widetilde{l_t}$	dimensionless thermal length
μ_{aux}	auxiliary variable
ω	test function

ABSTRACT

Cracking in continuous casting has always been one of the main problems of steel mills. Many cracks that occur during solidification are difficult to observe from outside the industrial mold. In order to better understand the formation of this defect, compared with the large-scale simulation used in the entire industrial process, microsimulation is also essential because The micro-segregation between dendrites at the solidification front is the one of the causes of crack formation. Besides, the cracks are mainly formed between the grain boundaries. A comprehensive study of using phase field method to simulate microstructure evolution has been conducted. A variety of two-dimensional models based on phase-filed method has been developed in order to simulate dendrites growth in continuous casting process. The basic concepts of phase-field method are presented. Among those models, Kobayashi model was first introduced to describe the morphology of pure material solidification, in this article, which are pure water and pure iron. In order to get closer to the actual situation of continuous casting, a multi-component model was introduced to solve the problem. To go a step further, by introducing a series of temperature parameters and modifications to a series of terms, the Fe-C binary alloy directional solidification model was used to simulate the process of dendrite growth in continuous casting. Furthermore, the detailed derivation of the binary alloy solidification model and how to apply the model in open source software will also be introduced in this article. The effects of physical parameters such as anisotropic strength, temperature gradient and cooling rate on the growth and evolution of the dendrite interface were quantitatively analyzed. Finally, potential improvement of this model, optimization to primary cooling section in continuous casting process and various applications of the simulation were discussed.

1. INTRODUCTION

1.1 Background and Motivation

Continuous casting (CC) technology has become the most widely used method for casting steel. Today, Continuous casting accounts for nearly 90% of the world's steel production [1]. In general, people divide continuous casting into two major parts: primary cooling and secondary cooling. Primary cooling consists of the mold region where steel solidification initiates along the external perimeter of the molten steel, inclosing it within a thin-walled shell. Secondary cooling is the region past the mold in which the steel shell further solidifies. In recent years, engineers devote themselves to reducing the formation of defects during the process in order to improve the quality and increase the production of steel. The formation of defects in the continuous casting is a very complicated process, which is the result of the interaction of heat transfer, mass transfer and stress.

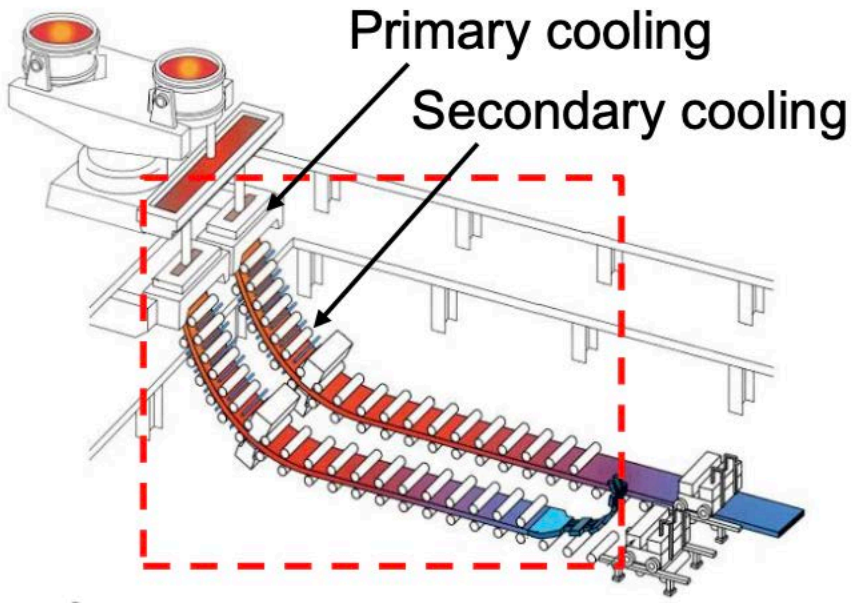

Figure 1. Continuous Casting Process [2].

For example, among the various defects of continuous casting slabs, cracks are one of the important defects. According to the statistics, about 50% of various defects in the slab originate from slab cracks. The small internal cracks in the slab may be welded in the subsequent rolling process, but due to factors such as the quality of the slab and the compression ratio limitation,

some internal cracks will remain in the subsequent products, which will affect the product quality and cause hidden safety risks. Internal cracks are hard to observe from the appearance of slab and it might be costly to take samples from semi-finished products for the sake of keeping steel quality. Besides adjusting several industrial parameters like cooling rate, pulling speed, macroscale simulation along with microscale simulation becomes a significant and efficient way to help predict defects in steel production. Macroscale and microscale simulations both have their advantages and limitations. By comprehensively considering the results of the two simulation methods, we are more likely to make accurate predictions.

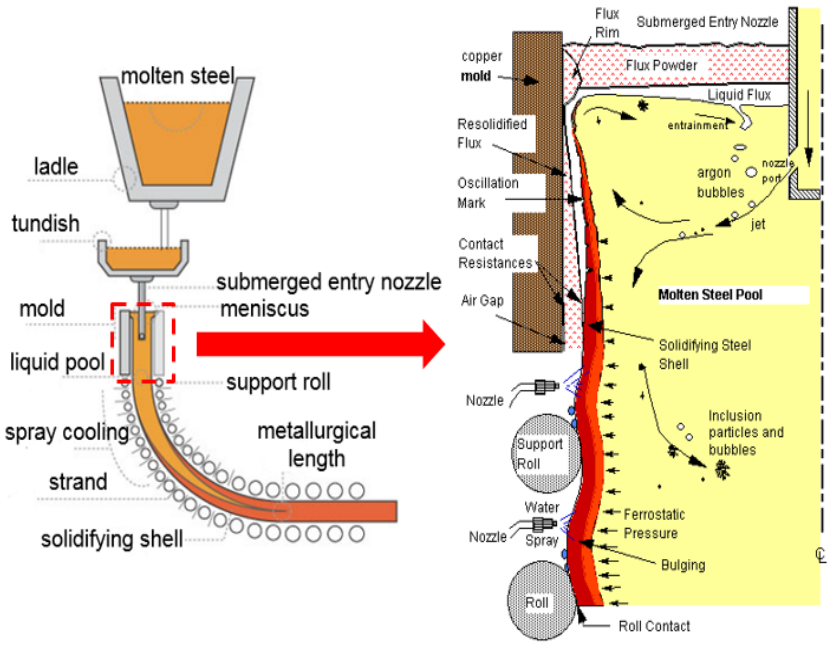

Figure 2. Physical phenomena inside both primary and secondary cooling zones [3].

Microscale simulation and crystal growth are almost inseparable. Through the microscope, people will find that crystal growth also has a charm that cannot be ignored. The growth of crystals is extremely contrast and random, and it can form very complicated shapes even in a very simple and uniform environment. By using phase-field method, it will be possible to simulate complex shape. Dendritic structures are commonly seen in solidification of metals and plays an important role in the mechanism of internal crack formation. The aim of this project is to develop a model which can describe the dendritic pattern formations under different cooling conditions and obtain the

crystal parameters in order to help conduct macroscale simulation, or even predict the formation of cracks.

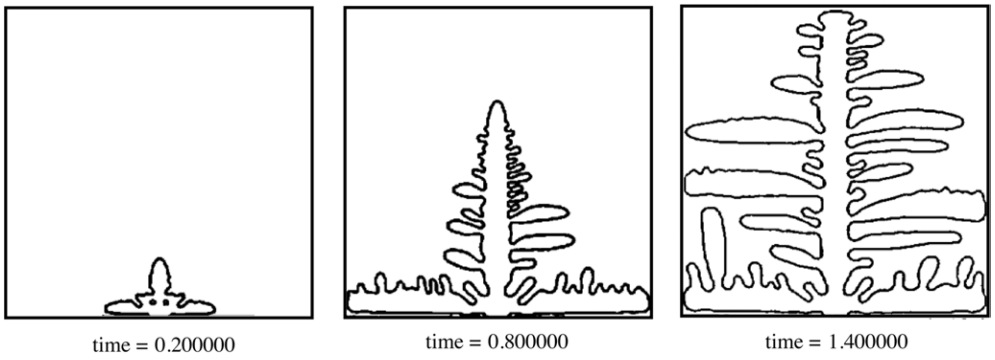

Figure 3. 2D simulation of dendritic growth of a pure substance in a highly undercooled melt [4].

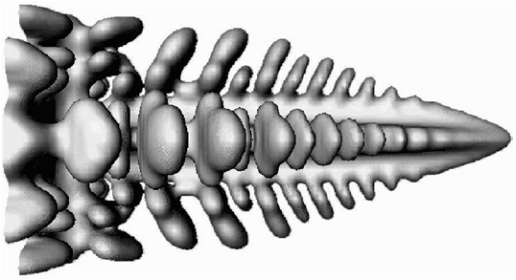

Figure 4. 3D simulation of dendritic growth of a pure substance in a highly undercooled melt [5].

A few commercial software may have the above-mentioned functions but are extremely expensive. Open source software can use phase-field method to simulate dendrite growth to a certain extent, but for directional solidification, which is close to actual steel continuous casting situation, is not feasible or to be developed. And this is the value of using existing open source software to finally achieve simulated directional solidification. This article will introduce step by step from the application of the most basic phase field method to a more complex directional solidification model. If new features are added to the open source program, the model can still improve the morphological accuracy through simple modification. The simulation results were compared with the empirical formula results.

1.2 Literature Review

Numerical simulation of microstructure is of great significance to the development and application of metal materials, and it is also one of the main development directions of computer application in the field of materials science. The main methods of numerical simulation of microstructure are: deterministic method, stochastic method and phase field method. Among them, the phase field method is a powerful tool for describing the evolution of complex phase interfaces in non-equilibrium states. It is not necessary to track the complex solid-liquid interface to simulate the complex morphology of dendrite growth during metal solidification. With the development of various related technologies, the phase field method is currently the international frontier research field of solidification simulation. The phase field method is based on the theory of Ginzburg and Landau [9]. First of all, differential equations are used to realize the combined effects of diffusion, potential energy and thermodynamic driving force, and then the above equations are solved by computer programming to obtain the instantaneous state of the research system in time and space. The phase field method is a derivative of an interdisciplinary subject, which comprehensively utilizes materials science, mathematics and physics, and computer programming. The phase field method has become a wide choice for simulating the microstructure evolution during solidification. the seminal paper of Alain Karma [6] about dendritic alloy solidification marks a breakthrough towards quantitative simulation [7].

The concept of “phase-field” is from Ryo Kobayashi’s paper [4]. Ryo Kobayashi proposed a solution to solve the solidification problem of pure materials in supercooled melts by replacing the sharp moving interface with a diffuse interface.

At the very beginning, Van der Waal [8] has modeled a liquid-gas system by continuous density function at the liquid-gas interface. Ginzburg and Landau [9] established a model first using order parameters and their gradients. Meanwhile Cahn and Hilliard [10] derived a thermodynamic formula that combined thermodynamic properties in the system with diffusion interfaces. However, it is quite a short time, only 20 years ago, the concept of diffusion interface was introduced into microstructure modeling which becomes the one of the foundations of phase field method.
The researchers are not satisfied with the theoretical model of the phase field method to simulate dendrite morphology, they tried to couple various physical parameters and combined them with other disciplines or newly developed method. Wang et al [11] derived the equations appropriate

for the adiabatic systems with temperature changes which makes the theoretical model more realistic. After that, Alain Karma [5] and Blas Echebarria [12] expanded application of the method to simulate microstructural pattern of dilute binary alloy for low speed directional solidification. Furthermore, Tomohiro Takaki [13] presented large scale competitive dendritic growth during directional solidification of Al–Si alloy. H. Yin, S.D. Felicelli [14] simulated the dendrite growth during solidification in the LENS process. And recently, Zhu et al [15] applied adaptive mesh and successfully add noise term to the directional solidification model. Phase field method becomes increasingly practical and absorbs other new methods to optimize the accuracy of the results.

2. METHODOLOGY

2.1 Phase Field Method

There are many kinds of materials used in industry, and most materials are heterogeneous at the microscopic level. In material science, their microstructure is composed of grains or phases. Different materials have different structure, orientation and chemical composition of their grains or phases. Their macroscopic physical and mechanical properties are highly dependent on crystal grains or the shape, size and distribution of crystal grains. Therefore, it is extremely important to understand the mechanism of microstructure formation and evolution. However, since the microstructure evolution is extremely complex and diverse, a large amount of theoretical and experimental research is required. Furthermore, the microstructure is essentially a thermodynamically unstable structure. The microstructure has no fixed shape, and its chemical potential is high. Once the thermodynamic parameters change, the higher driving force will make the organization continue to evolve or transform spontaneously. According to people's general thinking, it is determined that the evolution of microstructure is to track the continuous evolution of the frontier interface. Therefore, a large number of data points are needed in numerical simulation. Considering the characteristics of the microstructure listed above, this treatment method is undoubtedly undesirable and inefficient. And this indirectly led to the generation of the phase field method. The phase field method has become a powerful tool that can be used to simulate the evolution of microstructures in a variety of materials, such as solidification, phase transformation and grain growth.

2.1.1 Sharp Interface and Diffuse Interface Model

It is already mentioned that there are a lot of phase field models applied to different fields of scientific research. They are all based on a same foundation: diffuse interface model：the interface’s properties between phases are continuous changing within a very narrow region (Figure 5a). Correspondingly, the sharp interface model only allows the interfaces between phases are infinitely sharp (Figure 5b) which means the properties’ value are discontinuous.

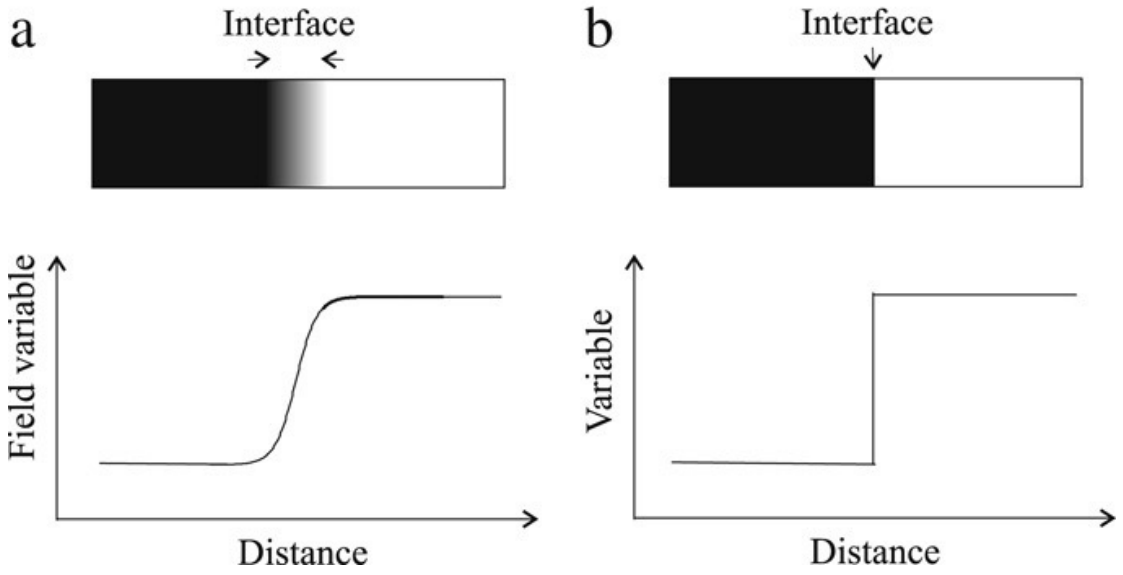

Figure 5. (a) Diffuse interface. (b) Sharp interface [16].

The following uses mathematical formulas to explain more specifically. Considering the diffusion-controlled growth. There are two phases A and B. According to Fick's second law, the equations for solute concentration in each phase are:

$$\frac{\partial c^{\alpha}}{\partial t} = D^{\alpha}\nabla^2 c^{\alpha},\ for\ phase\ \alpha \tag{1}$$

$$\frac{\partial c^{\beta}}{\partial t} = D^{\beta}\nabla^2 c^{\beta},\ for\ phase\ \beta \tag{2}$$

The flux balance equation that presents solute conservation at the interface is:

$$c^{\alpha 0} - c^{\beta 0} = D^{\beta}\frac{\partial c^{\beta}}{\partial r} - D^{\alpha}\frac{\partial c^{\alpha}}{\partial r} \tag{3}$$

The chemical potential at the interface should be in equilibrium state:

$$\mu^{\alpha}(c^{\alpha 0}) = \mu^{\beta}(c^{\beta 0}) \tag{4}$$

c^{α} and c^{β} are the molar concentrations of the solute in the α-phase and the β-phase respectively, $c^{\alpha 0}$ and $c^{\beta 0}$ are the molar concentrations at the interface. D^{α} and D^{β} are the diffusion coefficients. μ^{α} and μ^{β} are the chemical potentials of the solute in the α-phase and the β-phase respectively. r is the spatial coordinate perpendicular to the interface. From above equations, the sharp interface

model requires tracking the location and chemical components of the moving interface between the α-phase and the β-phase. Sharp interface model can efficiently solve the problems related to one-dimensional system and simple grain morphologies like spherical grains, since it is very easy to get the spatial coordinate. When the problem is expanded to two-dimensional complex alloy dendrite morphology, the model will be difficult to implement or even unfeasible from a mathematical point of view.

In the diffuse model, the physical meaning of these parameters will not change. In general, instead of tracking the moving interface to describe the microstructure, the microstructure is described by a set of continuous functions of space and time, which are called phase-field variables. The microstructure evolution is defined over the whole system. Resulting it is possible to predict the microstructure evolution even it is complicated grain morphology.

2.1.2 Order Parameter

In the previous section, phase-field variables were mentioned. Such variables can be related to many physicochemical parameters in governing equations to solve the expected practical problems. More importantly, it can also be used with a non-conservative quantity called order parameter. The actual meaning of this variable depends on the problem to be solved. In solidification problem, it can represent the state of substance. Take water for instance, if water is in liquid state, we can set order parameter $\varphi = 0$. If water turns into ice, we can set the order parameter $\varphi = 1$. By solving the governing equations, the value distribution of order parameter φ can be presented on a plane (Figure 6). In Figure 6, the blue part represents liquid water ($\varphi = 0$), and the red part represents ice ($\varphi = 1$). The function of phi is continuous, the value of φ varies continuously from 0 and 1at the interface between two phases. In Figure 6, a very narrow white curve is used. In this case, φ represents the fraction of solid phase. It should be noted that the threshold of φ is not necessarily 0 and 1. Depending on the selected model, the available values are -1, 0, 1, 2, etc. The common points of these values are integers, which helps to divide the boundaries for display.

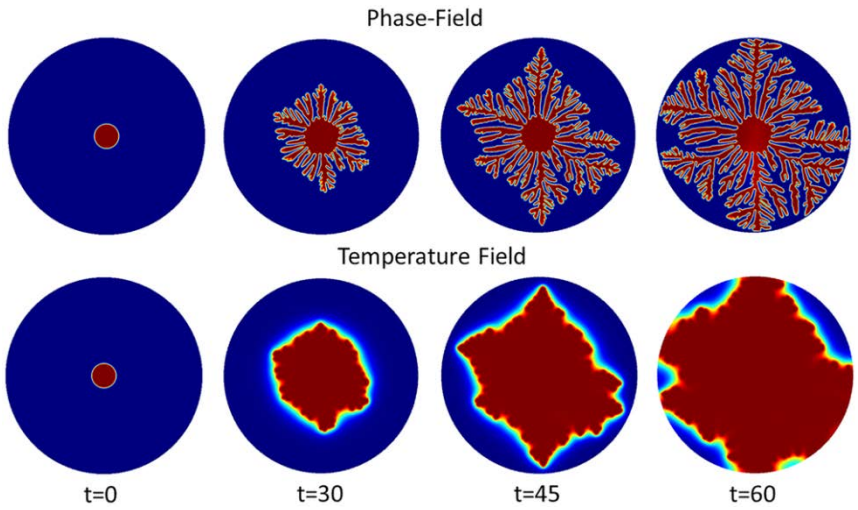

Figure 6. Simulation of formation for snowflake by using phase field method [16].

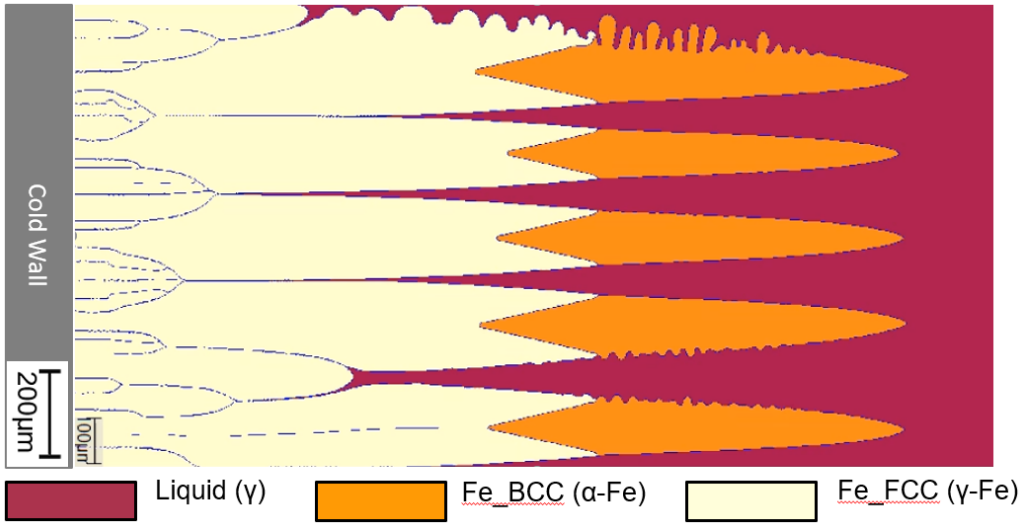

Figure 7. Directional solidification simulation for Fe-C-Mn alloy by using commercial software Micress®.

$$\gamma : \varphi = 0, \alpha - Fe: \varphi = 1, \gamma - Fe: \varphi = 2.$$

Besides solidification, phase field method is widely used in solid phase transformation. Here will briefly introduce the principle of how to achieve this function. When a substance changes from one phase to another, the crystal system tends to change accordingly, meaning that the

corresponding crystal parameters also change, such as side length of cubic unit cell (Figure 8). Set the initial side length of the cubic unit cell to be a_c and three order parameters indicate whether the length, width and height of the cube cell have changed. The phase transformation can obtained by tracking the value of η_1, η_2, η_3.

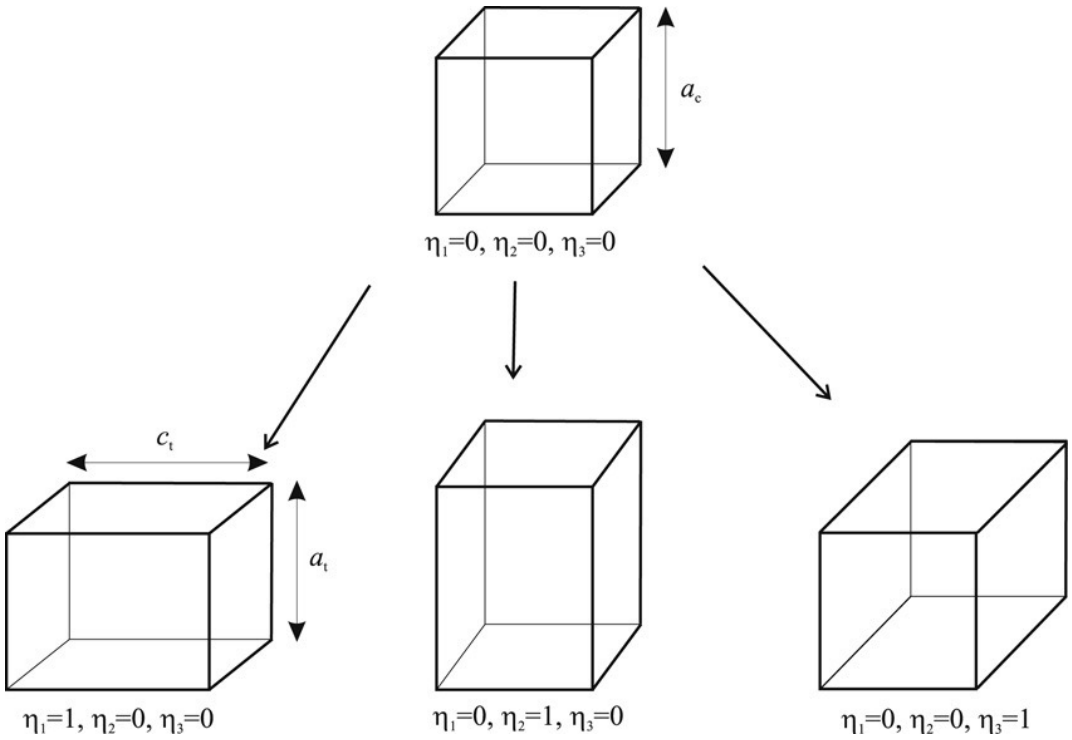

Figure 8. Three order Parameters are used to distinguish different crystal structures[15].

2.2 Pure Material Solidification

In continuous casting, dendrites grow from the cold mold wall. The thermodynamical environment is complicated inside the mold. Hence, a simplified model with only one composition and fixed temperature boundary conditions was first introduced to simulate the dendrite growth. There are two types of dendrite forms: equilibrium and growth. In equilibrium form, if it is isotropic, crystal surface tends to be sphere in order to minimize the surface energy. If there is anisotropy exists, the shape tends to become polyhedron. Apparently, dendrites are not equilibrium forms because of the complicated shapes which are formed under the surface tension and thermodynamical driving force: supercooling or supersaturation. Supercooling is the difference between the theoretical crystallization temperature and the actual given crystallization temperature and supersaturation refers to the state of supersaturation of a solution. The reason is that the temperature decreases or the solute increases or the solvent decreases. However, dendrites growth simulation cannot be accomplished by only these two parameters. Anisotropy which greatly affect the growth of dendrites should also be included in the solidification model. In this chapter, the pure material dendrite growth model will be explained, say Kobayashi model.

2.2.1 Kobayashi Model

There are two variables in this model; one is phase field φ, and the other is temperature field T. They are both the function of location and time. The phase field variable φ is an order parameter represents the status of substance. $\varphi = 0$ means being liquid and $\varphi = 1$ means being solid. The steep layer of φ, represents the interface, is consists of a series of continuous values between 0 and 1. Figure 9 shows how phase field variable φ is used to describe the shape of dendrites.

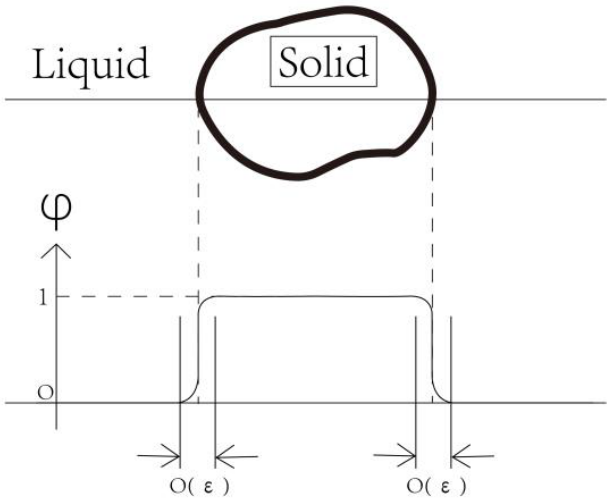

Figure 9. The solid/liquid interface is expressed by the phase field variable .

The governing equations of φ in this model is:

$$\tau\frac{\partial\varphi}{\partial t}=-\frac{\partial}{\partial x}\left(WW'\frac{\partial\varphi}{\partial y}\right)+\frac{\partial}{\partial y}\left(WW'\frac{\partial\varphi}{\partial x}\right)+\nabla\cdot(W^2\nabla\varphi)+\varphi(1-\varphi)(\varphi-\frac{1}{2}+m) \tag{5}$$

Equation (5) express the evolution of φ. Where τ is a small positive constant called interface relaxation time. The thermodynamical driving force is given by parameter m.

$$m=\frac{\alpha}{\pi}\arctan[\gamma(T_e-T)] \tag{6}$$

α and γ are positive constant coefficients. T_e represents the equilibrium temperature. Hence, the driving force in solidification is proportional to the degree of supercooling.

In order to take anisotropy into account, interface layer width W is in following form:

$$W=\bar{W}[1+\varepsilon\cos(j\theta)] \tag{7}$$

Where $\bar{W}$ is the mean value of W. ε and j are the anisotropy strength and anisotropy mode number relatively. θ represents the angle between the interface normal direction and the positive direction of x axis.

$$\theta=\arctan\left(\frac{\partial\varphi}{\partial y}\Big/\frac{\partial\varphi}{\partial x}\right) \tag{8}$$

From conservation law of enthalpy, the evolution equation for temperature field T is:

$$\frac{\partial T}{\partial t} = \nabla^2 T + K\frac{\partial \varphi}{\partial t} \tag{9}$$

K in here represents the dimensionless latent heat which is proportional to the latent heat.

2.2.2 Adjusted equations for programming

The simulations for dendrite growth are accomplished by using C^{++}based open source software. Due to the different advantages and disadvantages of various open source software, the simulations shown in this article will use two different software. Their advantages and disadvantages will also be described in the corresponding chapters. For pure material solidification, the simulations are accomplished by using MOOSE Framework. MOOSE Framework integrates some commonly used phase field models, called kernels, such as diffusion kernels and. If the desired simulation is developed based on existing kernels, it will be very fast and convenient. But it will be very difficult to achieve further development if the equations or initial conditions used beyond the range of existing kernels.

In MOOSE Framework, there is no need to derive the weak form for governing equations. The things need to do are getting the residuals ready and divide each term in residuals into the existing kernels.

The residuals and kernels are present below

$$R_1 = \underbrace{\frac{\partial \varphi}{\partial t}}_{Time\ Derivative} \underbrace{- \frac{1}{\tau}\left[-\frac{\partial}{\partial x}\left(WW'\frac{\partial \varphi}{\partial y}\right)\right] + \frac{1}{\tau}\frac{\partial}{\partial y}\left(WW'\frac{\partial \varphi}{\partial x}\right)}_{ACinterfaceKobayashi1} - \underbrace{\frac{1}{\tau}\nabla\cdot(W^2\nabla\varphi)}_{ACinterfaceKobayashi2} - \underbrace{\frac{1}{\tau}\varphi(1-\varphi)\left(\varphi - \frac{1}{2} + m\right)}_{ACParsed}] \tag{10}$$

R_1 is the residual derived from equation (5), four kernels called Time Derivative, ACinterfaceKobayashi1, ACinterfaceKobayashi2, ACParsed are used in further development.

$$R_2 = \underbrace{\frac{\partial T}{\partial t}}_{Time\ Derivative} - \underbrace{\nabla^2 T}_{Diffusion} - \underbrace{K\frac{\partial \varphi}{\partial t}}_{CoefCoupledTimeDerivative} \tag{11}$$

$R2$ is the residual derived from equation (8), three kernels called Time Derivative, Diffusion, CoefCoupledTimeDerivative are used in further development.

2.3 Directional Alloy Solidification

2.3.1 Directional Alloy Solidification Model

To make the simulation more accurate, the alloy composition must be incorporated into the simulation. Besides, more detailed thermodynamic conditions must also be involved in governing equations. Compared with pure material solidification, directional solidification for alloy is much more difficult to obtain the same reasonable results. The first reason for the difficulty is that the diffusion rate in solid and liquid is significantly different: in liquid phase, the solute diffusion will be much faster which will cause nonequilibrium at the interface. Another is the solute trapping effect which lead to solute atoms cannot escape solidification front to achieve equilibrium state at the interface.

First, we introduce equations to describe the simplified alloy solidification model, called isothermal solidification model:

$$\tau a_s^2 \frac{\partial \varphi}{\partial t} = \nabla[W(\theta)^2 \nabla\varphi] + \frac{\partial}{\partial x}\left(|\nabla\varphi^2| W(\theta) \frac{\partial W(\theta)}{\partial\left(\frac{\partial \varphi}{\partial x}\right)}\right) + \frac{\partial}{\partial y}\left(|\nabla\varphi^2| W(\theta) \frac{\partial W(\theta)}{\partial\left(\frac{\partial \varphi}{\partial y}\right)}\right) + \varphi - \varphi^3 - \lambda(1-\varphi^2)^2 U \tag{12}$$

Equation (11) is the evolution equation for phase field variable φ. Here, τ is the interface relaxation time.

$$\tau = \frac{a_1 a_2 \xi^3 d_0^2}{D_l} \tag{13}$$

In equation (12), a_1 and a_2 are constant coefficients. a_1 = 0.8839, a_2 = 0.6267, ξ is the interface layer width parameter, d_0 is the solute capillary length. λ is the coupling constant.

$$d_0 = \frac{\Gamma}{m_l c_0 (k-1)} \tag{14}$$

Γ is the Gibbs-Thomson coefficient. m_l represents the liquidus slope. c_0 is the initial concentration and k is Solubility partition coefficient. $W(\theta)$ is the gradient energy coefficient. a_s is the anisotropy term for alloy solidification. ξ is the interface layer width parameter.

$$W(\theta) = W \cdot a_s \tag{15}$$

$$W = \xi \cdot d_0 \tag{16}$$

$$a_s = 1 + \varepsilon cos4(\theta - \theta_0) \tag{17}$$

As mentioned, ε is the anisotropic strength. θ is the angle between the normal direction and the positive direction of the interface. θ_0 is the initial rotational angle with respect to the simulation frame.

Besides considering the evolution of phase field variable φ. The evolution of dimensionless supersaturation should also be involved in the model.

$$[(1+k)-(1-k)\varphi]\frac{\partial U}{\partial t} = \nabla[Dq(\varphi)\nabla U] + [1+(1-k)U]\frac{\partial \varphi}{\partial t} \tag{18}$$

Equation (16) is the evolution equation for the dimensionless supersaturation variable U. And U has a close relationship with concentration:

$$U = \frac{c-c_0}{c_0(1-k)} \tag{19}$$

$q(\varphi)$ in here is to eliminate the effect of different diffusion rates in solid and liquid.

$$q(\varphi) = (1-\varphi) + k(1+\varphi)D_s/D_l$$

D_s and D_l in here represent the solute diffusion coefficient in solid and liquid respectively.

In order to add the directional solidification effect and eliminate the solute trapping effect. Equation (12) and (16) should be rewritten as:

$$\tau\left[1+\frac{Gy-Rt}{m_l c_0}\right]a_s^2\frac{\partial \varphi}{\partial t} = \nabla[a_s^2\nabla\varphi] + \frac{\partial}{\partial x}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial \varphi}{\partial x}\right)}\right) + \frac{\partial}{\partial y}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial \varphi}{\partial y}\right)}\right) + \varphi - \varphi^3 - \lambda(1-\varphi^2)^2(U - \frac{Gy-Rt}{mlc_0}) \tag{20}$$

$$[(1+k)-(1-k)\varphi]\frac{\partial U}{\partial t} = \nabla[Dq(\varphi)\nabla U - j_{at}] + [1+(1-k)U]\frac{\partial \varphi}{\partial t} \tag{21}$$

Equation (18) and (19) are the modified equations from isothermal solidification model to simulate the directional solidification. Where G is the temperature gradient. R is the cooling rate.

The temperature field for directional solidification along the y-axis can be described as:

$$T(y) = T_0 + G(y - V_p t) \tag{22}$$

V_p is the pulling speed which equals to R/G. T_0 is the initial temperature.

To eliminate the solute trapping effect, the term needs to be proportional to the moving speed at the interface which is $\frac{\partial \varphi}{\partial t}$. Besides it is also related the interface layer width and local concentration and served as a function to transport the solute atoms from solid to liquid. Hence, j_{at} is called anti-trapping term and has following expression:

$$j_{at} = -\frac{1}{\sqrt{2}} W[1 + (1-k)U] \frac{\partial \varphi}{\partial t} \frac{\nabla \varphi}{|\nabla \varphi|} \tag{23}$$

The phase field variable φ in directional solidification model is varies from -1 to 1. $\varphi = 1$ indicating the solid phase, $\varphi = -1$ represents the liquid phase.

2.3.2 Adjusted equations for programming

The simulation of dendrites directional solidification was accomplished by using PRISMS-PF. Compared to MOOSE, PRISMS-PF allow users to customize governing equations and initial conditions more freely. This is a significant advantage for handling complicated equations. PRISMS-PF requires users to provide weak form for programming.

Dimensionless Procedure

In this case, the first step is to change the governing equations to dimensionless type to simplify coding and avoid the inaccuracy caused by very small values which means the simulation is scaled by the interface layer width W and interface relaxation time τ.

Hence, the governing equations become:

$$\left[1 - (1-k)\frac{\tilde{y} - \widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right] a_s^2 \frac{\partial \varphi}{\partial \tilde{t}} = \nabla[a_s^2 \nabla \varphi] + \frac{\partial}{\partial \tilde{x}}\left(|\nabla \varphi^2| a_s \frac{\partial a_s}{\partial\left(\frac{\partial \varphi}{\partial \tilde{x}}\right)}\right) + \frac{\partial}{\partial \tilde{y}}\left(|\nabla \varphi^2| a_s \frac{\partial a_s}{\partial\left(\frac{\partial \varphi}{\partial \tilde{y}}\right)}\right) + \varphi - \varphi^3 - \lambda(1-\varphi^2)^2\left(U + \frac{\tilde{y} - \widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right) \tag{24}$$

$$[(1+k) - (1-k)\varphi]\frac{\partial U}{\partial \tilde{t}} = \nabla\left[\widetilde{D} q(\varphi) \nabla U - j_{at}\right] + [1 + (1-k)U]\frac{\partial \varphi}{\partial \tilde{t}} \tag{25}$$

An example of detailed derivation of dimensionless equation is attached to the Appendix A.

Next step is to use an auxiliary variable μ_{aux} to further simplified the equations.

$$\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}=\mu_{aux} \tag{26}$$

$$[(1+k)-(1-k)\varphi]\frac{\partial U}{\partial\tilde{t}}=\nabla\left[\widetilde{D}q(\varphi)\nabla U-j_{at}\right]+[1+(1-k)U]\frac{\mu_{aux}}{\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right]a_s^2} \tag{27}$$

$$\mu_{aux}=\nabla(a_s^2\nabla\varphi)+\frac{\partial}{\partial\tilde{x}}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial\tilde{x}}\right)}\right)+\frac{\partial}{\partial\tilde{y}}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial\tilde{y}}\right)}\right)+\varphi-\varphi^3-\lambda(1-\varphi^2)^2(U+\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}) \tag{28}$$

Time Discretization

Considering Euler explicit time stepping, we have the time discretized equations:

$$\varphi_{n+1}=\varphi_n+\frac{\mu_{aux,n}\Delta t}{\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right]a_s^2} \tag{29}$$

$$U_{n+1}=U_n+\Delta t\left\{\frac{\nabla[\widetilde{D}q(\varphi)\nabla U_n-j_{at}]}{[(1+k)-(1-k)\varphi_n]}+\frac{[1+(1-k)U]\mu_{aux,n}}{[(1+k)-(1-k)\varphi_n]\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right]a_s^2}\right\} \tag{30}$$

$$\mu_{aux,n+1}=\nabla(a_s^2\nabla\varphi)+\frac{\partial}{\partial\tilde{x}}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial\tilde{x}}\right)}\right)+\frac{\partial}{\partial\tilde{y}}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial\tilde{y}}\right)}\right)+\varphi-\varphi^3-\lambda(1-\varphi^2)^2(U+\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}) \tag{31}$$

By using chain rules$\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial\tilde{x}}\right)}=\frac{\partial a_s}{\partial(\theta)}\cdot\frac{\partial\theta}{\partial\left(\frac{\partial\varphi}{\partial\tilde{x}}\right)}$, equation (28) can be written more compactly.

$$\mu_{aux,n+1}=\nabla\left[(a_s^2\frac{\partial\varphi_n}{\partial\tilde{x}}+a_sa_s'\frac{\partial\varphi_n}{\partial\tilde{y}})+(a_s^2\frac{\partial\varphi_n}{\partial\tilde{y}}-a_sa_s'\frac{\partial\varphi_n}{\partial x})\right]+\varphi_n-{\varphi_n}^3-\lambda(1-{\varphi_n}^2)^2(U_n+\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}) \tag{32}$$

Equation (26), (27) and (29) are going to be used for weak formulation and then convert to code.

Weak Formulation

The advantage of PRISMS-PF is that it simplifies the programming process of the finite element method, users do not need to have too much programming knowledge and skills. The weak form of the equations is the most important part that users need to input to PRISMS-PF.

By using Green theorem and divergence theorem. The weak form of equation (26), (27) and (29) are:

$$\int \omega \varphi_{n+1} dV = \int \omega \left(\varphi_n + \frac{\mu_{aux,n} \Delta t}{\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right] a_s^2} \right) dV \tag{33}$$

$$\int \omega U_{n+1} dV = \int \omega \left\{ U_n + \frac{[1+(1-k)U_n]\mu_{aux,n} \cdot \Delta t}{[(1+k)-(1-k)\varphi_n]\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}\right] a_s^2} \right\} - \nabla \omega \left\{ \frac{\nabla[\tilde{D} q(\varphi) \nabla U_n - j_{at}]\Delta t}{[(1+k)-(1-k)\varphi_n]} \right\} dV \tag{34}$$

$$\int \omega \mu_{aux,n+1} dV = \int \omega \left[\varphi_n - {\varphi_n}^3 - \lambda(1-{\varphi_n}^2)^2 (U_n + \frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_T}}) \right] - \nabla \omega \left[(a_s^2 \frac{\partial \varphi_n}{\partial \tilde{x}} + a_s a_s' \frac{\partial \varphi_n}{\partial \tilde{y}}) + (a_s^2 \frac{\partial \varphi_n}{\partial \tilde{y}} - a_s a_s' \frac{\partial \varphi_n}{\partial x}) \right] dV \tag{35}$$

Here, ω represents the test function. Equation (30), (31) and (32) are the final form to be converted to code.

3. NUMERICAL SIMULATION SETUPS

In this part, the simulated objects, parameters, initial conditions and boundary conditions will be included. Some detailed information to obtain the value of parameter is also included.

3.1 Simulations of Pure Material Solidification

3.1.1 Materials for Pure Material Solidification

In Kobayashi model, pure water and pure iron will be simulated to grow dendrites. The physical properties and other simulation parameters that used in simulations are shown in Table 1.

Table 1. Physical Parameters of Pure Material Solidification

Physical parameters	**Parameter value of water**	**Parameter value of iron**
Interface relaxation time, τ	0.0003s	0.0003s
Constant coefficient, α	0.9	0.9
Constant coefficient, γ	10.0	10.0
Equilibrium temperature,T_e	1℃	1539℃
Mean value of interface width, $\bar{W}$	0.01μm	0.01μm
Anisotropy strength, ε	0.02	0.02
Mode number, j	6	4
Dimensionless latent heat, K	-1.8	-1.8

3.1.2 Mesh and Time Step for Pure Material Solidification

The computational domain for pure material solidification is a 2D square area. The domain size is 9.0 μm × 9.0 μm which is same as Kobayashi's setting. The initial number of elements in x and y directions are 14. And the element type is linear element. Due to MOOSE framework uses adaptive mesh, there are much finer meshes in dendrite growth region. The time step for simulations is 0.0005s. According to Kobayashi [3] and Blas Echebarria [11], the value of time step is small enough to obtain accurate and clear results.

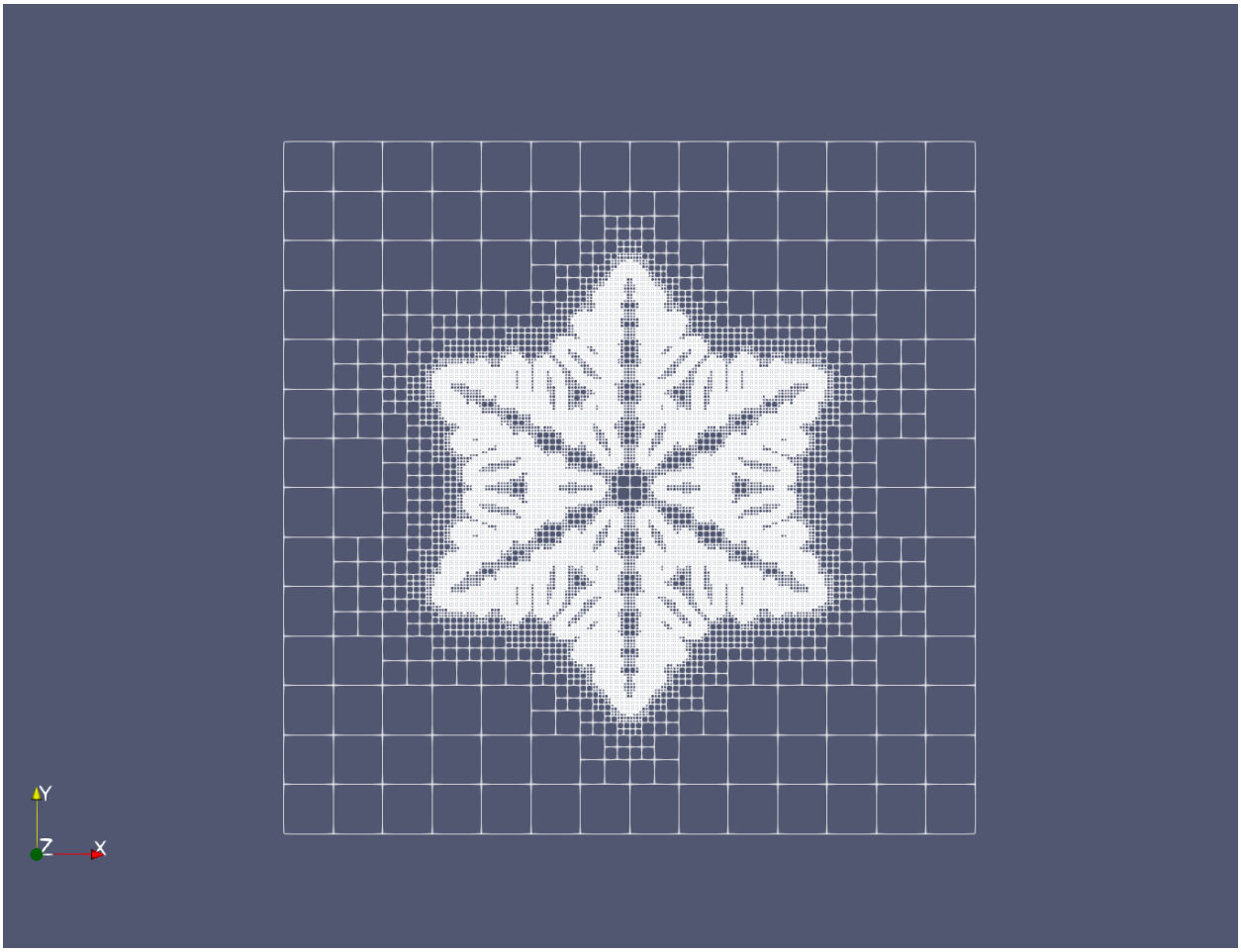

Figure 10. Mesh for pure water solidification, finer mesh can be clearly seen at the tip and side branches region of dendrites.

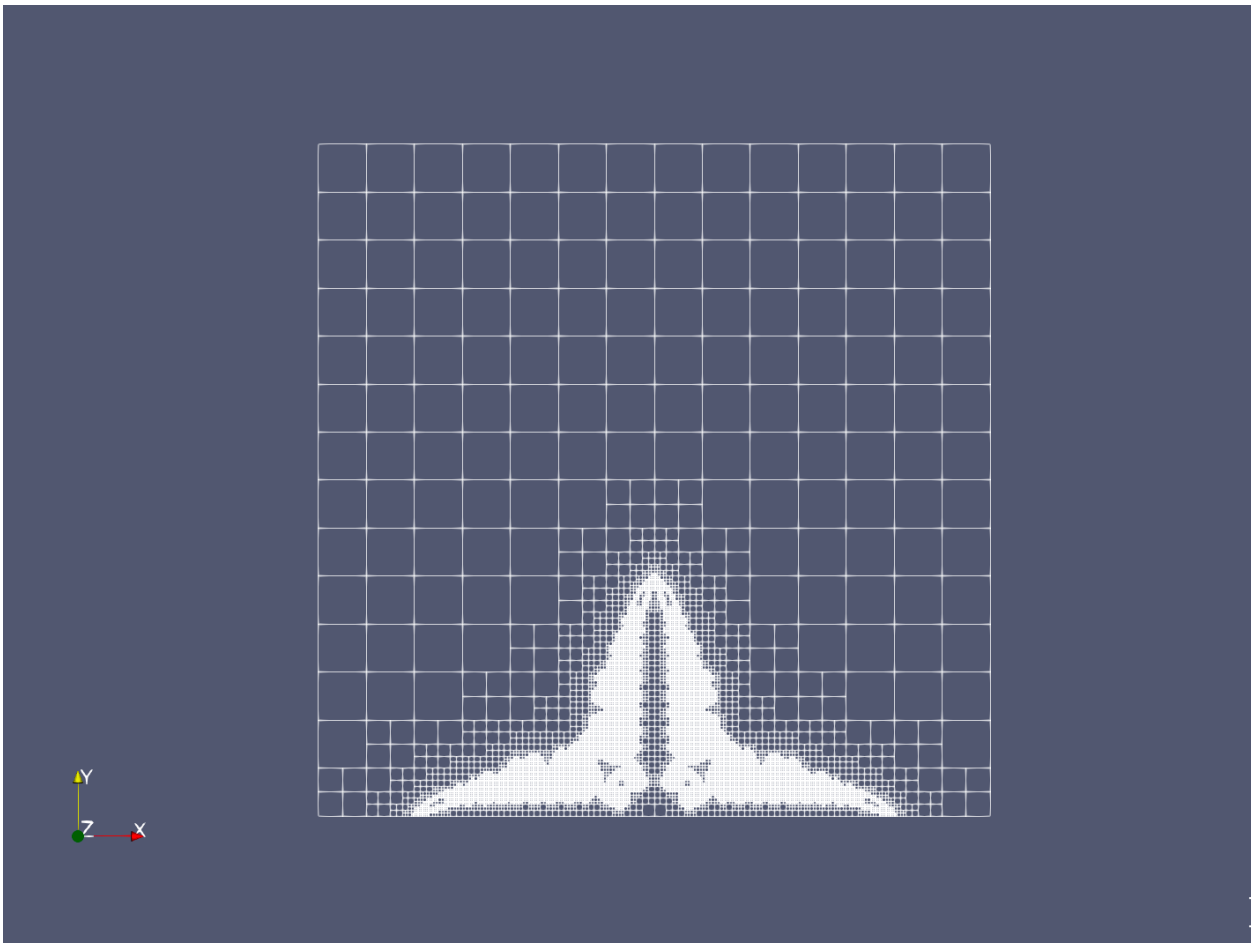

Figure 11. Mesh for pure iron solidification with only one dendrite, finer mesh can be clearly seen at the tip and side branches region of dendrites.

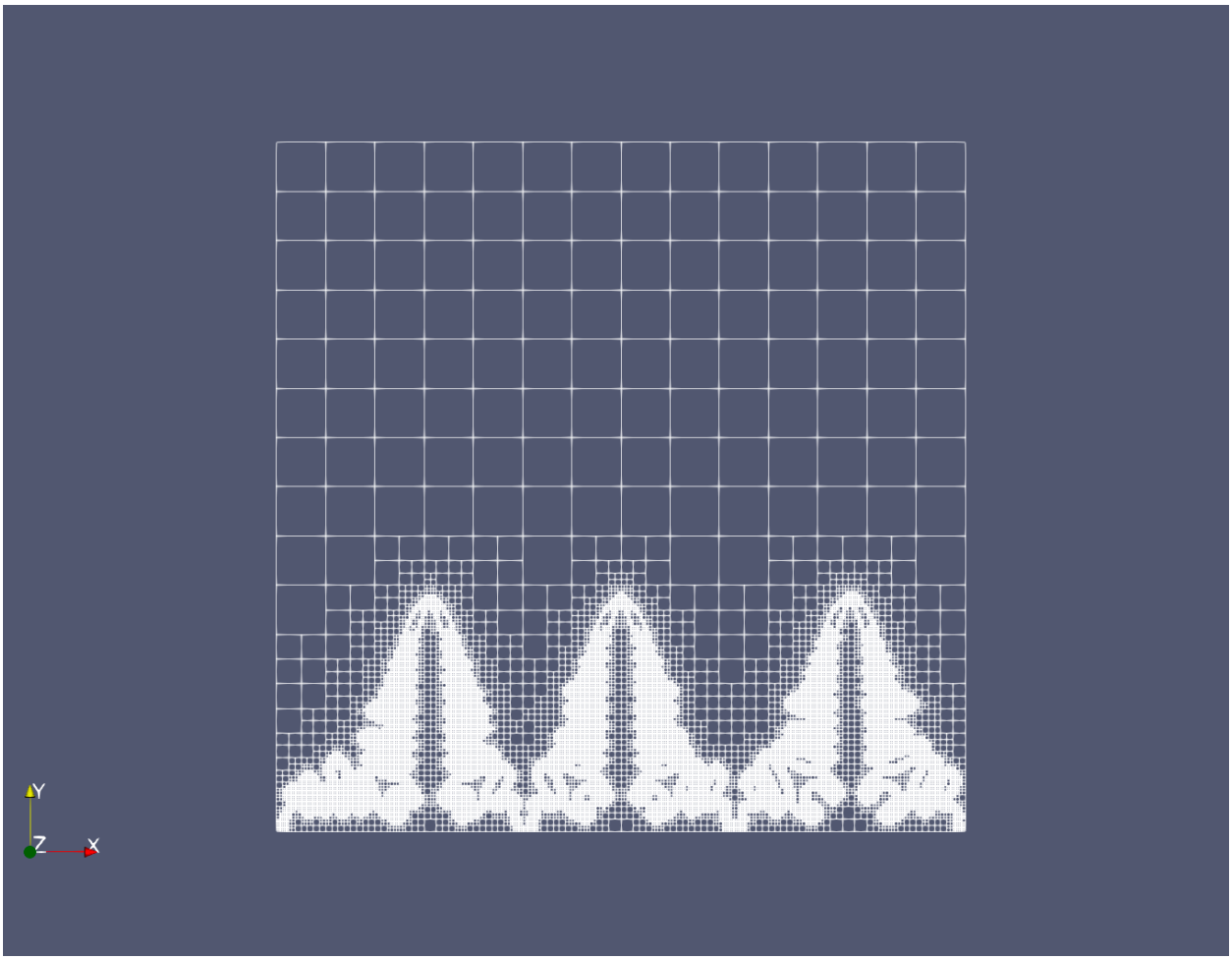

Figure 12. Mesh for pure iron solidification with multiple dendrites, finer mesh can be clearly seen at the tip and side branches region of dendrites.

3.1.3 Initial Conditions and Boundary Conditions for Pure Material Solidification

In MOOSE Framework, it is very intuitive and convenient to defining initial conditions. Users can set single nucleus or multiple nuclei to grow dendrites. At the beginning, the nucleus itself is a solid, and there is a liquid to be solidified outside the nucleus. Hence, we can set initial conditions for phase field variable φ as:

$$v(r)\begin{cases} v_{in}, & for\ |r| \le R_0 - \dfrac{W}{2} \\ v_{out} + \dfrac{1}{2}(v_{in} - v_{out})[1.0 + \cos\left(\pi \dfrac{\overrightarrow{|r|} - R + \frac{W}{2}}{W}\right), & for\ R_0 - \dfrac{W}{2} \le |r| \le R_0 + \dfrac{W}{2} \\ v_{out}, & for\ |r| \ge R_0 + \dfrac{W}{2} \end{cases}$$

Here, r represents the displacement of the current location to the center of the nucleus. v_{in} is the value of the phase field variable inside the nucleus. v_{out} is the value outside the nucleus. R_0 in here is the initial radius of nucleus. W is the interface width. This cosine profile initial conditions allows phase field variable varies continuously from 0 to 1 at the interface.

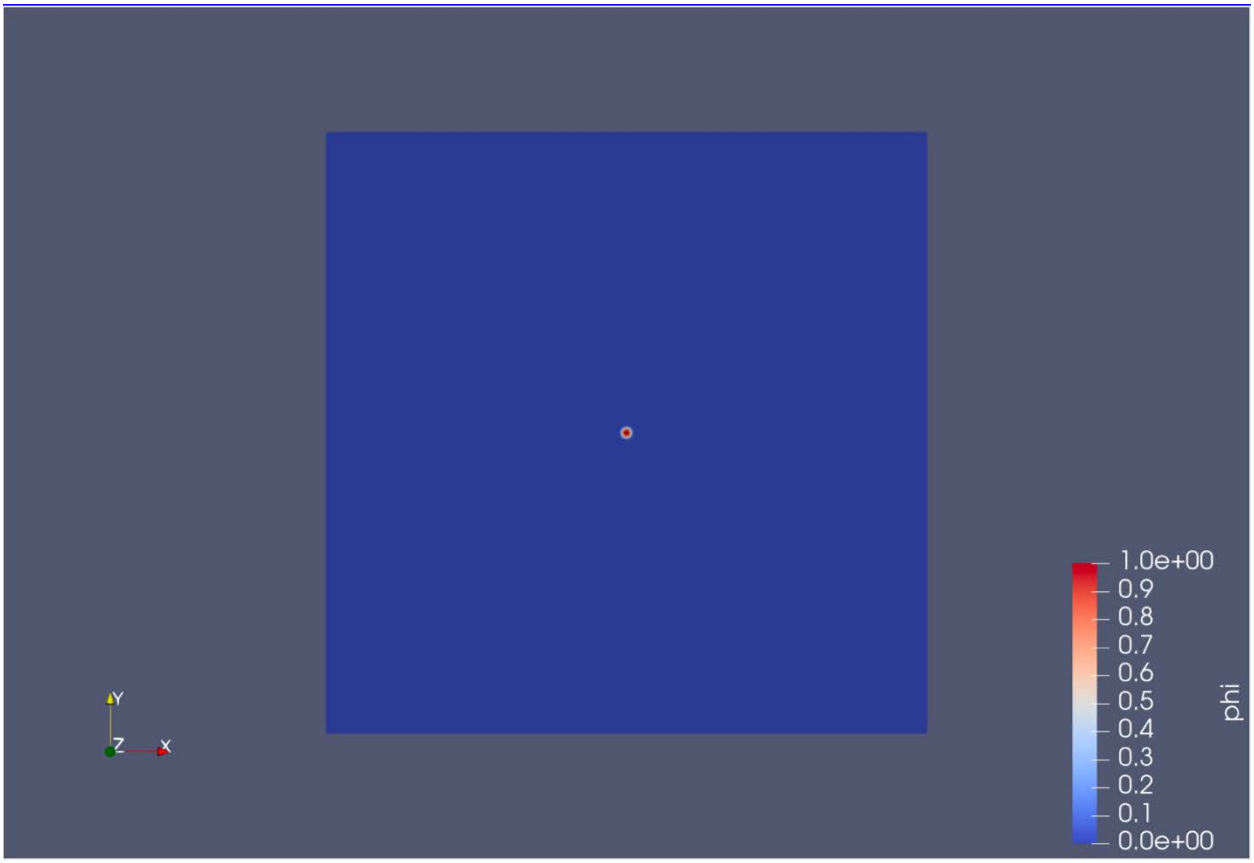

Figure 13. Initial conditions of phase field variable in pure water solidification. The red dot at the center is the nucleus of which the initial value is 1.

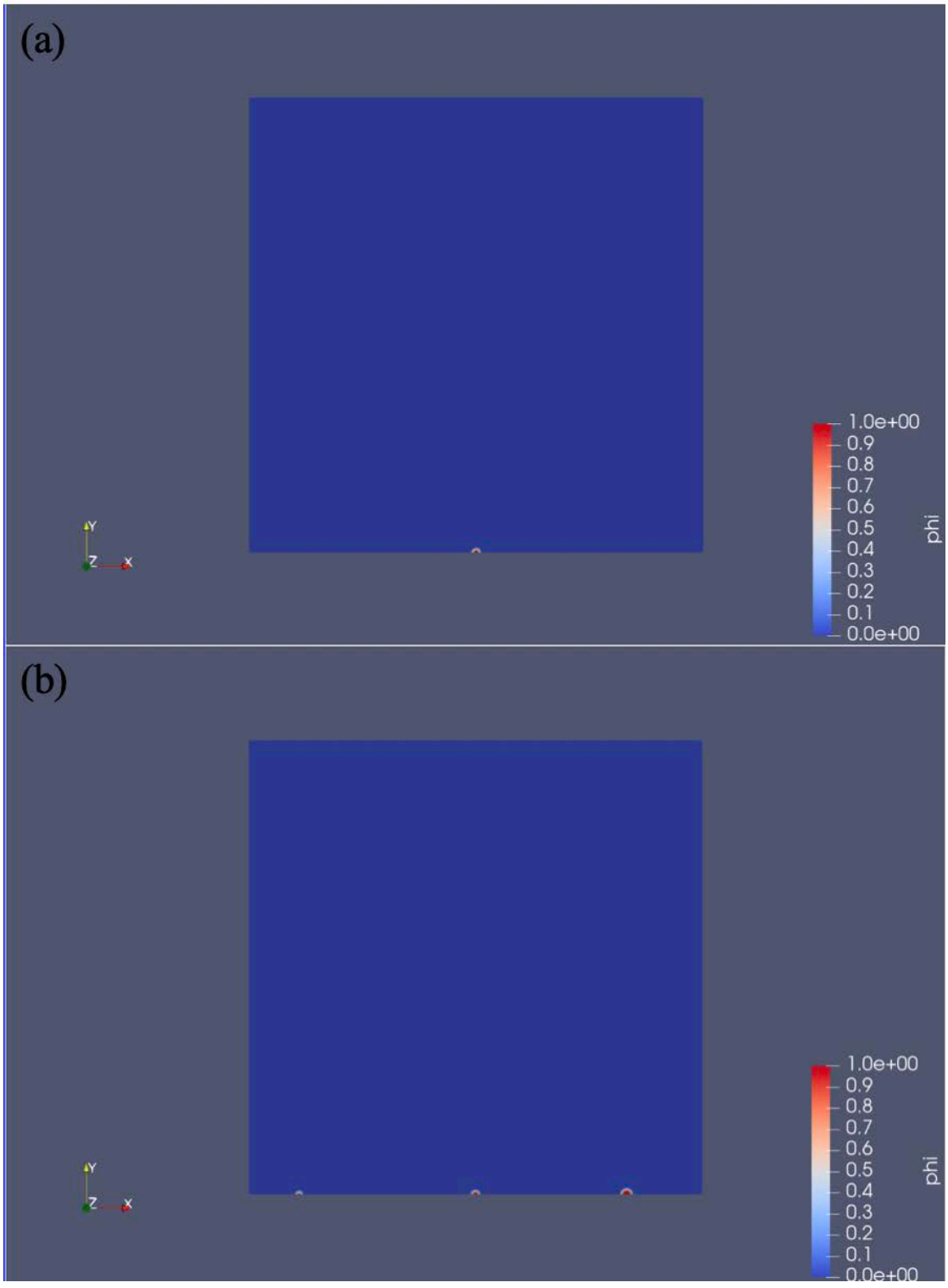

Figure 14. Initial conditions of phase field variable φ in pure iron solidification. (a)The red dot at the bottom center is the nucleus of which the initial value is 1. (b) The three red dots at the bottom is the nucleus of which the initial value is 1.

For pure water solidification, Neumann boundary conditions are used for temperature field and phase field in the computational domain:

$$\begin{cases} \dfrac{\partial \varphi}{\partial n} = 0 \\ \dfrac{\partial T}{\partial n} = 0 \end{cases}$$

For pure iron solidification, Neumann boundary condition is used for phase field and Dirichlet boundary condition is used for temperature field at the bottom in the computational domain:

$$\begin{cases} \dfrac{\partial \varphi}{\partial n} = 0 \\ T_{bottom} = 1539°\mathrm{C} \end{cases}$$

Figure 15. Boundary conditions of pure iron solidifications. Bottom temperature is 1537°C. The upper boundary value of temperature can be seen in the color bar on the right.

3.2 Simulations of Alloy Directional Solidification

3.2.1 Materials for Directional Alloy Solidification

In directional solidification model, low carbon steel (0.13%C) will be simulated to grow dendrites. The carbon contents of steel and major physical properties of steel were obtained from Won Y.M, and Thomas B.G's paper [18]. The physical properties and other simulation parameters that used in simulations are shown in Table 2.

Table 2. Physical Parameters of Directional Alloy Solidification

Physical parameters	**Parameter value**
Solute diffusion coefficient in solid, D_s	$8.92 \times 10^{-12} m^2/s$
Solute diffusion coefficient in liquid, D_l	$5.66 \times 10^{-9} m^2/s$
Initial liquid temperature, T_0	1808.15K
Gibbs-Thomson coefficient, Γ	$5.39 \times 10^{-7} m^2 \cdot K$
Liquidus slope, m_l	-78.0 K/wt.%
The initial concentration, c_0	0.13 wt.%
Temperature gradient, G.	3700 K/m
Cooling rate, R	0.045 K/s
Pulling speed, V_p	$1.22 \times 10^{-5} m/s$
Solubility partition coefficient, k	0.19
Interface width parameter, ξ	40
Anisotropy strength, ε	0.02

3.2.2 Mesh and Time Step for Directional Alloy Solidification

The computational domain for directional alloy solidification is a 2D 500×500 square area. The initial number of elements in x and y directions are 18. And the element type is cubic element. PRISMS also uses adaptive mesh which can generate finer meshes in dendrite growth region. The time step for simulations is 0.01. The length and time in this simulation are scaled by W and τ. The computational domain can accommodate up to three dendrites for competitive growth. Therefore, the simulation which has random nucleus location will be repeated many times to improve the credibility of the results.

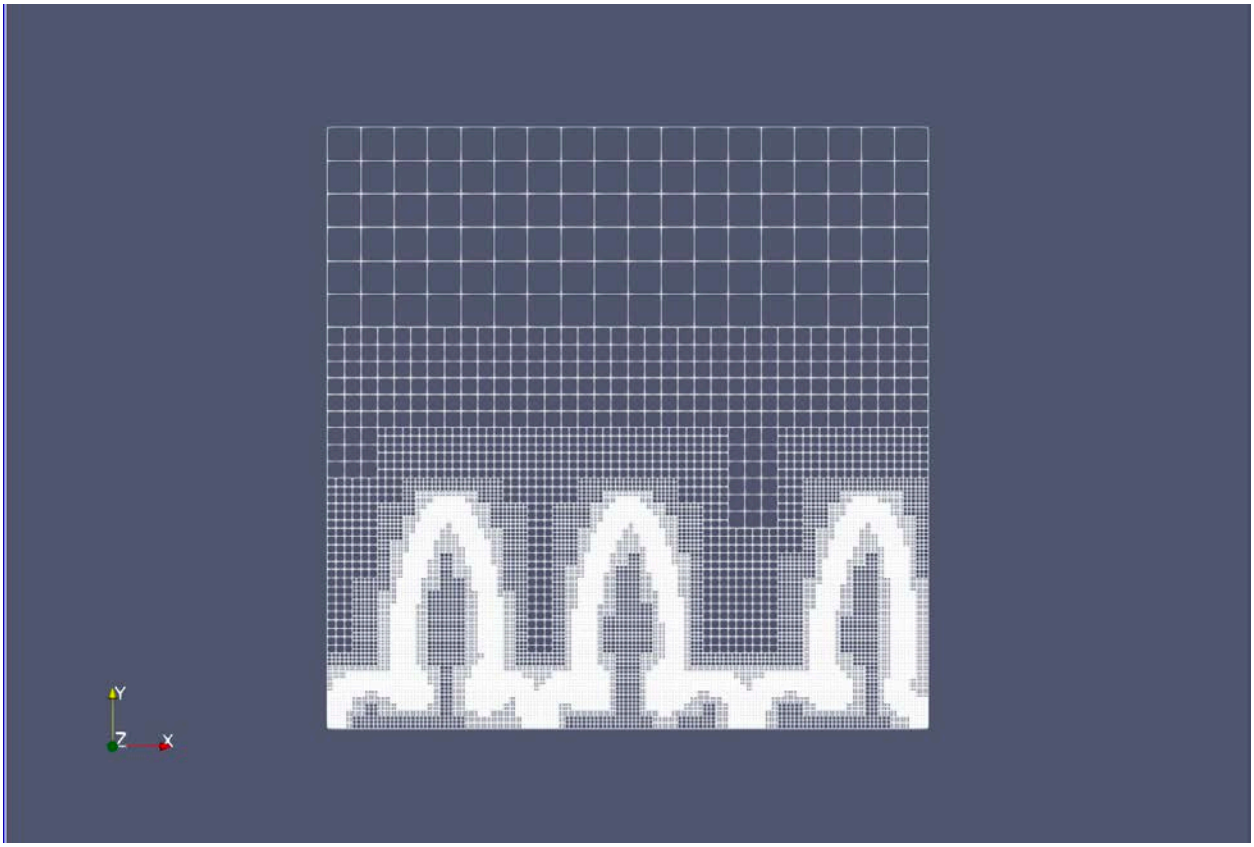

Figure 16. An example of mesh for directional alloy solidification.

3.2.3 Initial Conditions and Boundary Conditions for Directional Alloy Solidification

The directional solidification simulation has two variables need to be iterated. One is dimensionless supersaturation U, the other is phase field variable φ. For dimensionless supersaturation, according to Karma [5]'s paper, the initial value is -0.55. For phase field variable φ, it varies from -1 to 1 in the directional solidification. Hence, the tanh profile was used to describe the initial conditions of φ:

$$v(r) = \left[1 - \tanh\frac{(|r| - R_0)}{\sqrt{2}}\right] - 1$$

In directional solidification, temperature gradient and cooling rate were taken into consideration. Hence, there is no need to apply Dirichlet boundary conditions. Neumann boundary conditions were applied to whole computational domain:

$$\begin{cases} \frac{\partial \varphi}{\partial n} = 0 \\ \frac{\partial U}{\partial n} = 0 \end{cases}$$

4. RESULTS AND DISCUSSIONS

In this section, the results of a series of solidifications and the results of directional solidification will be shown.

The simulation of pure material solidification is one of the initial attempts of the phase field method. It focuses more on how to successfully simulate the complex morphology of dendrites, so the results of its physical parameters will be less accurate. Considering this situation, the simulation of pure material solidification will proceed parametric study and will be validated with literature from Kobayshi Ryo [3].

The simulation of directional solidification. The directional solidification simulation considers more physical and thermodynamic effects. Therefore, it can obtain more accurate crystal parameters. Parametric studies will also be conducted. The result will be validated with empirical formula and data from literatures.

4.1 Results of Pure Material Solidification

4.1.1 Simulation Results

The results of pure material solidification that used the simulation parameters listed in Table.1 were shown below.

Figure 17. Result of phase field variable φ for pure water solidification.

The snowflake pattern in the center is the crystal dendrites of water. The crystal dendrites fully conform to the previously setting of mode number which equals to 6 and the primary dendrites grow along six directions spaced 60 degrees apart. Because the MOOSE framework has the setting of adding noise term, the secondary dendrites which are the side branches of the 6 primary dendrites can also be seen clearly from Figure 18.

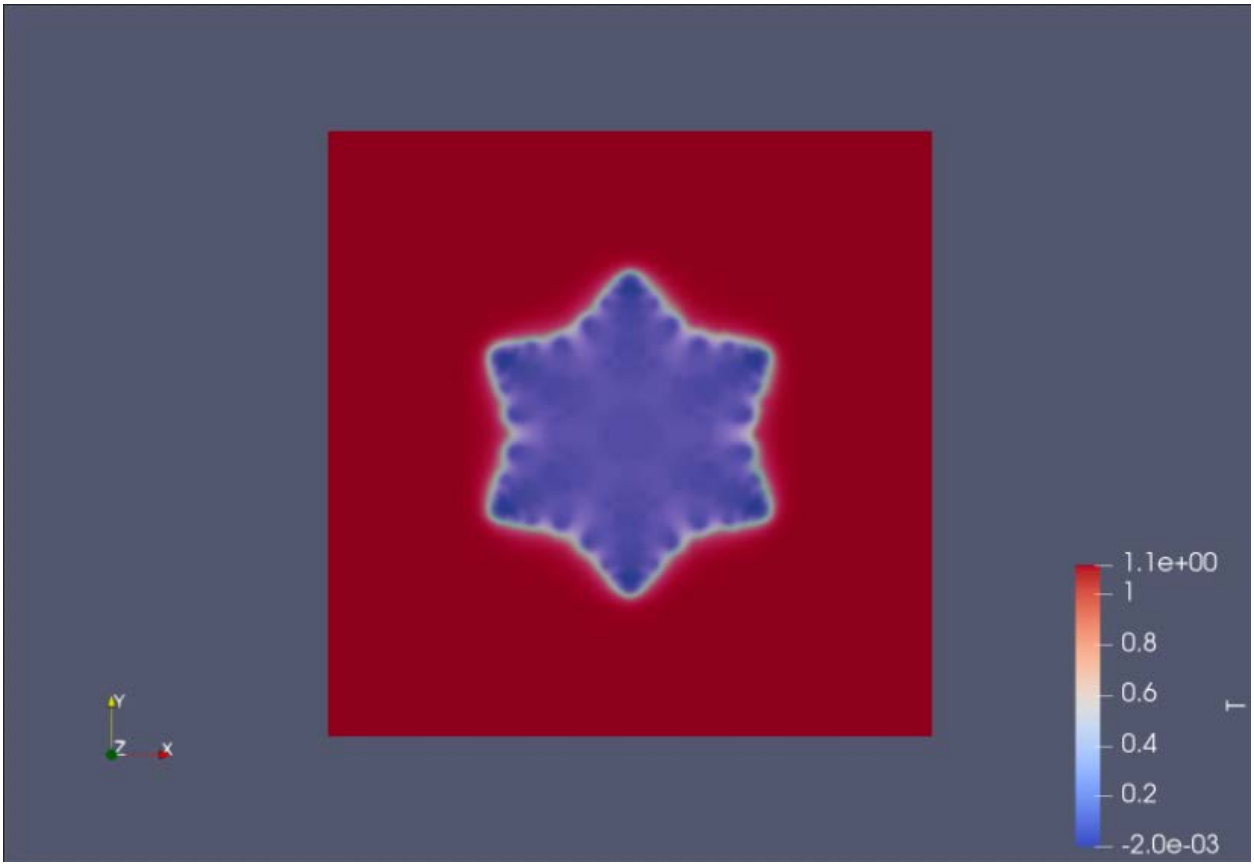

Figure 18. Result of temperature field T for pure water solidification.

The result of temperature field shows the temperature inside the crystal is about 0 °C. The temperature range at the interface is 0.4 °C to 0.6 °C which can be further solidified. The temperature of the melt environment is equal to 1.1 due to the heat released by the solidification of the crystal.

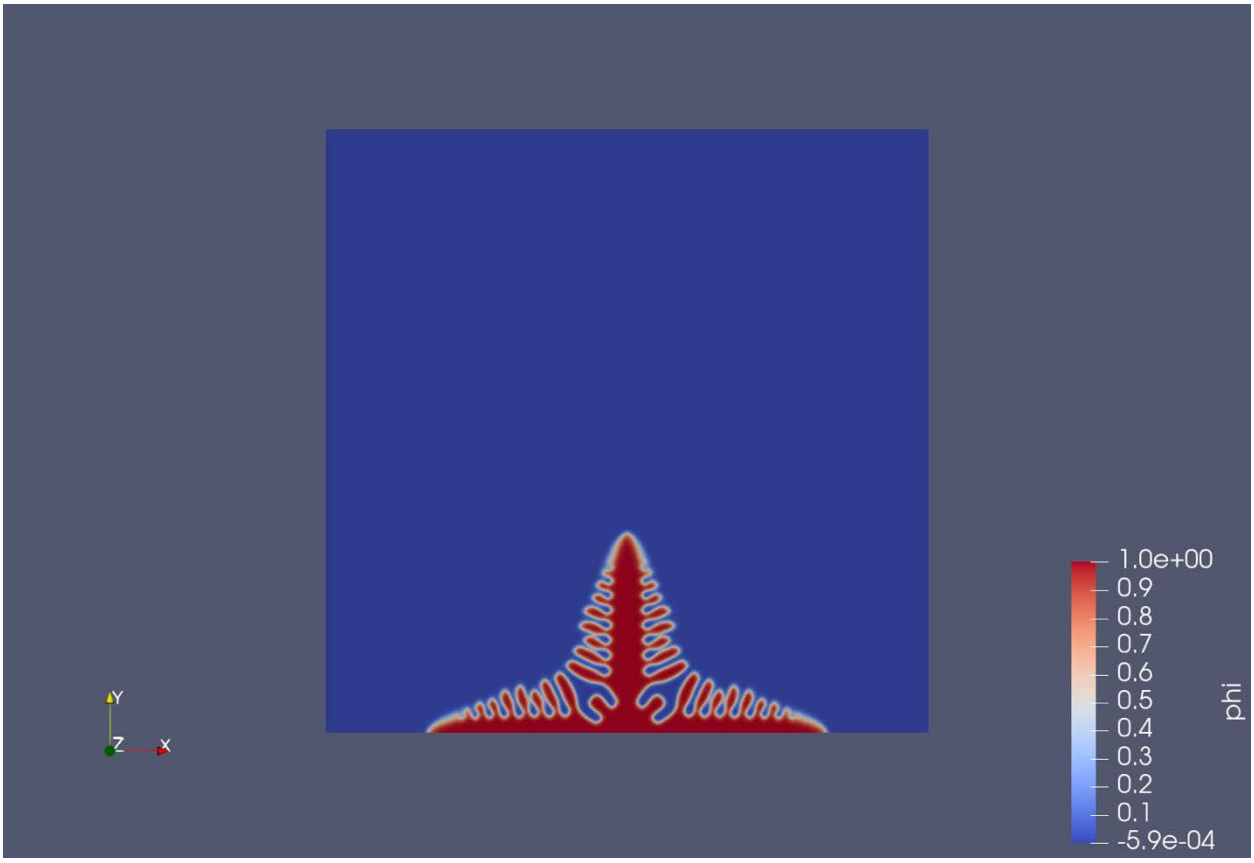

Figure 19. Result of phase field variable φ for pure iron solidification with single dendrite.

For pure iron solidification, nucleus grow from the bottom with initial temperature at 1537 °C which represents the cold mold wall in continuous casting process. Primary and secondary dendrites also formed successfully in this case.

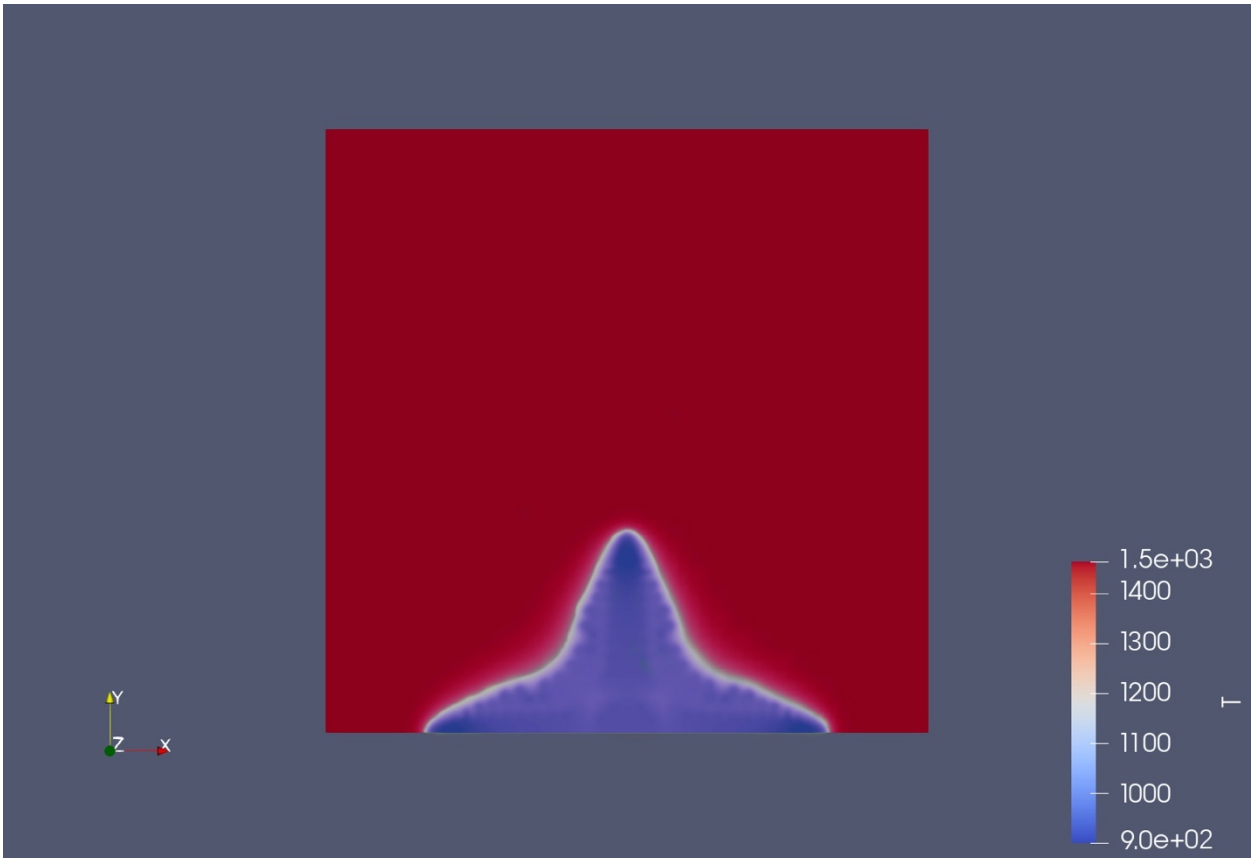

Figure 20. Result of temperature field T for pure water solidification with single dendrite.

The temperature of dendrites is about 900 °C to 1000 °C. And the temperature range at the interface is 1100 °C to 1200°C which can be further solidified. Due to the limitation of visualization tool, the highest temperature in the melt environment can only be accurate to the hundred which means the highest temperature is between 1500 °C and 1600 °C.

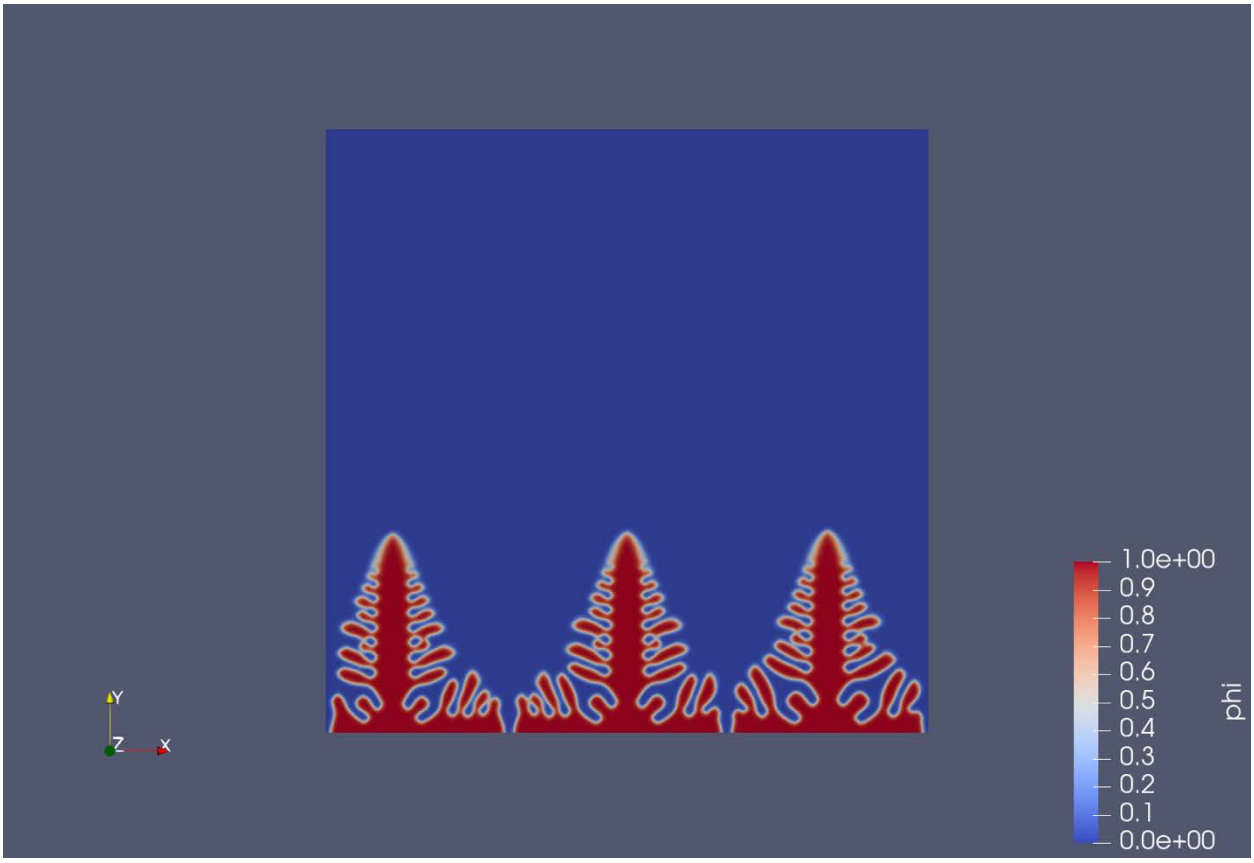

Figure 21. Result of phase field variable φ for pure iron solidification with multiple dendrites.

By adding more nuclei to grow from the bottom cold wall. Dendrites will grow competitively resulting dendrites tend to form longer secondary dendrites.

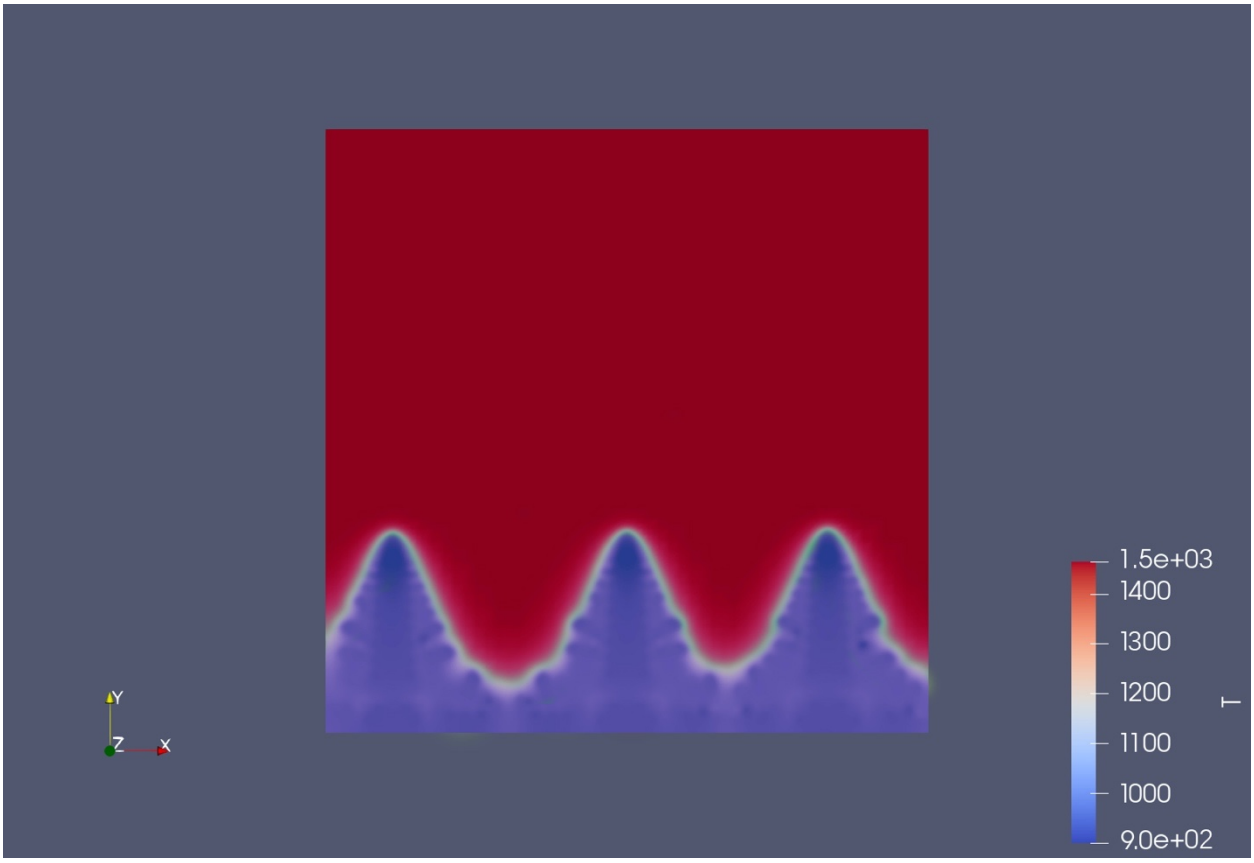

Figure 22. Result of temperature field T for pure water solidification with multiple dendrites.

Temperature field for multiple dendrites in pure iron solidification is similar to the case of single dendrite. The Kobayashi model is a successful attempt to simulate complex dendrite morphology using the phase field method. However, due to its limitations, it does not include more detailed material properties and heat transfer models corresponding to the coupling. It pays more attention to the evolution of complex morphology during dendrite growth, so its temperature field results are not accurate. In this simulation, the pure substances are almost fixed temperature for different state. It does not take into account heat conduction and heat convection, and this is unreasonable in actual industrial production. In the process of production, there is a very important parameter called primary dendrite arm spacing (PDAS) and secondary dendrite arm spacing (SDAS) in unidirectional solidification. During the solidification process, primary dendrites grow out of the condensation nucleus, secondary dendrites grow from the primary dendrites and form the side branches. The PDAS is the distance between two adjacent primary dendrites and the SDAS is the distance between two adjacent secondary dendrites. Generally speaking, small dendrite spacings

will make the mechanical properties of the material better. And the degree of influence is greater than the effect of grain size on mechanical properties. This is also one of the reasons why micro-simulation is now valued. The Kobayashi model does not consider directional solidification, so it is meaningless to discuss the dendrite arm spacing in this section. The results of this important parameter will be discussed in later sections.

4.1.2 Parametric Study

In this section, parametric study will focus on dimensionless latent K and anisotropy strength ε. Different anisotropy strength and dimensionless latent heat will be used to conduct parametric study to explore its impact on the simulation and will be validated with literature.

Anisotropy Strength ε

Five different anisotropy strength were selected to conduct parametric study of pure iron solidification at fixed dimensionless latent heat K= 2.0 according the Kobayashi's paper [3]. The following pictures are the result of different anisotropic strength. Anisotropic strength is a simulation parameter and does not have too many physical connections with other parameters. It only affects the growth pattern of dendritic branches. For $\varepsilon = 0.000$, dendrites tend to form viscous fingering shape. For $\varepsilon = 0.005$, isotropic and dendritic features can be seen from crystal shapes, viscous finger-like structure is still existing on the side branches. For $\varepsilon = 0.010 \sim 0.050$, the results in this interval are very close to the actual metal crystal morphology. By comparing the dendrite morphology of the actual metal crystal under the scanning electron microscope, the most suitable anisotropic strength can be obtained for simulation. For a specific material, the optimal anisotropy strength used for simulation is a fixed value. From Figure 24, the side branches dendrites tend to grow perpendicularly. This corresponds to the situation where ε equals to 0.020 ~0.050. Hence, the values of the anisotropic strength ($\varepsilon = 0.020$) selected in this paper are reasonable

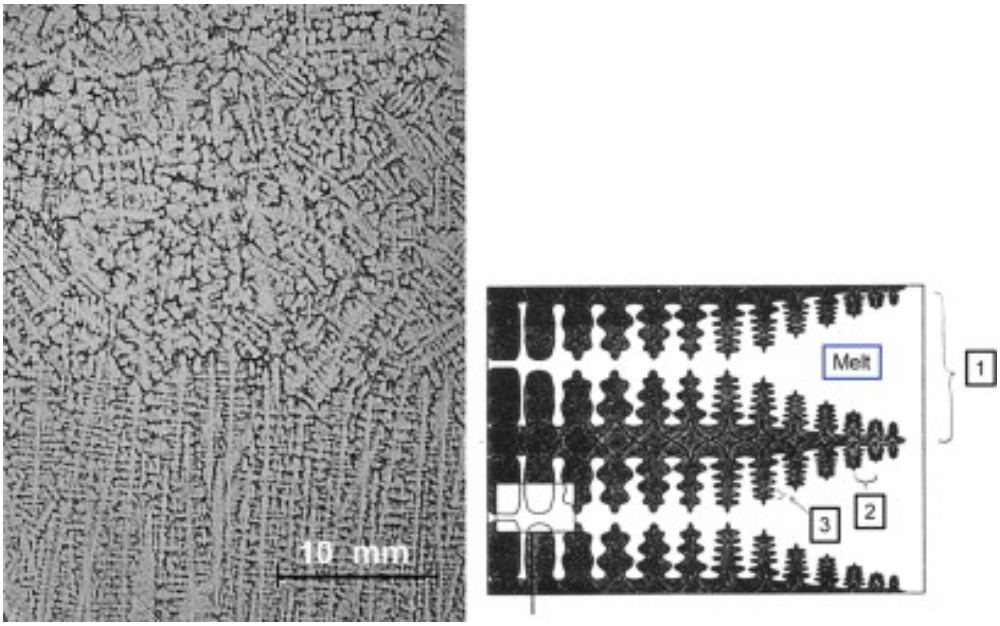

Figure 23. Columnar dendrites and equiaxed dendrites in the sample taken from steel billet (left) and a schematic picture of an ideal columnar dendrite (right) [30]

1:PDAS 2:SDAS

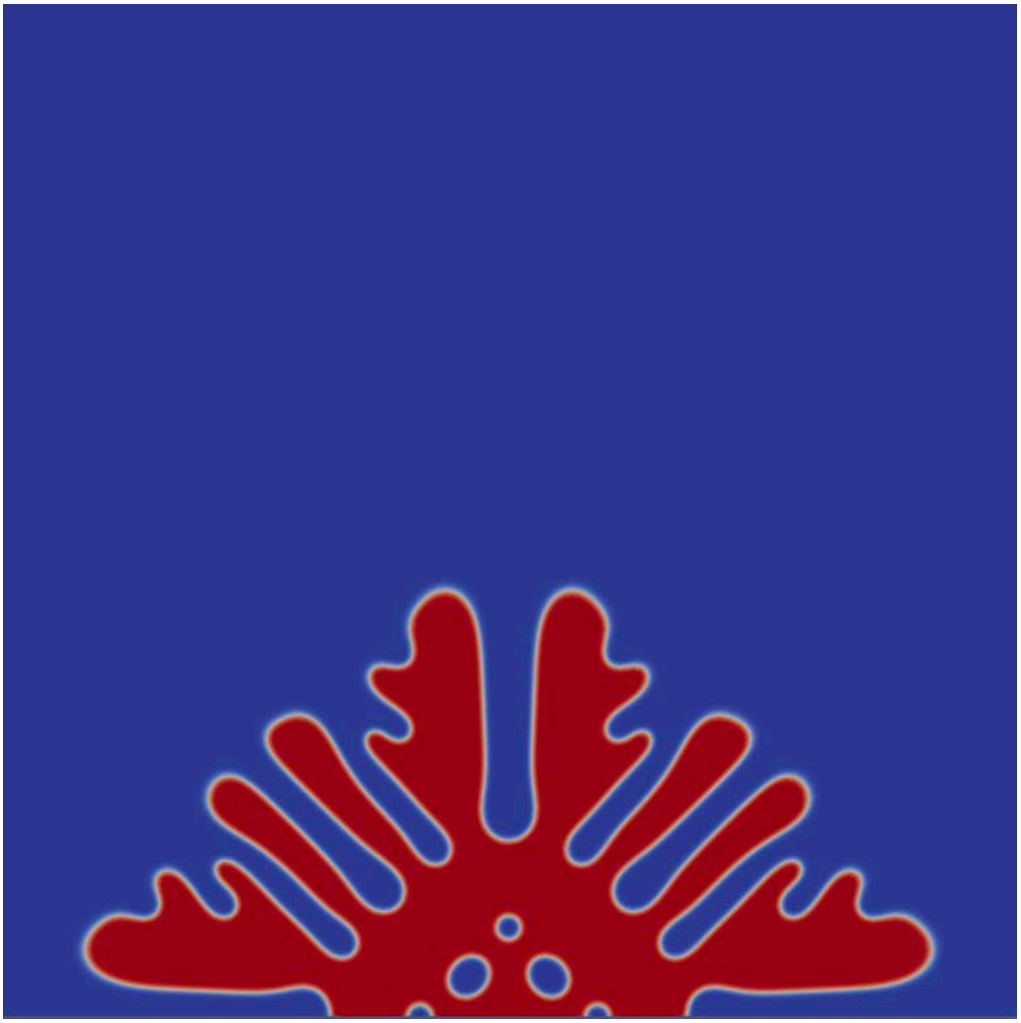

Figure 24. Result of anisotropy strength $\varepsilon = 0.000$

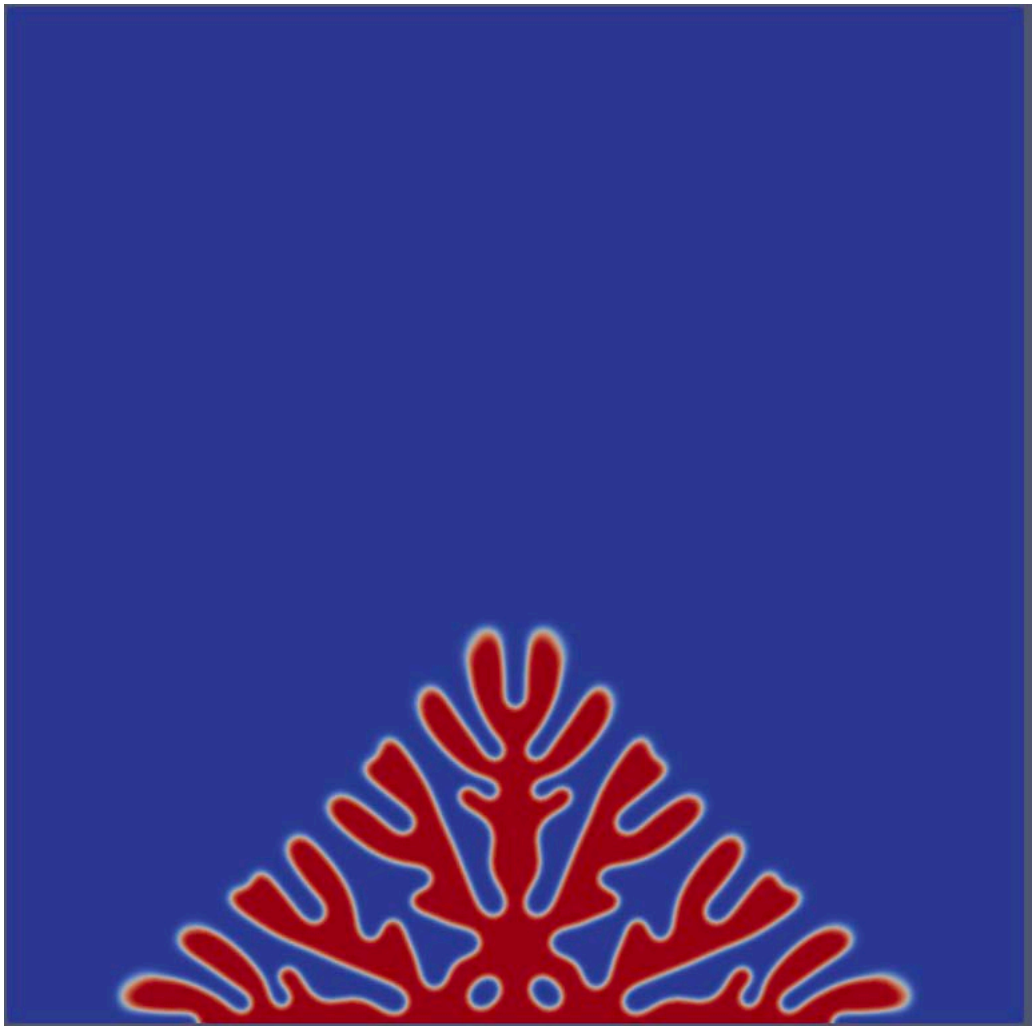

Figure 25. Result of anisotropy strength $\varepsilon = 0.005$

Figure 26. Result of anisotropy strength $\varepsilon = 0.010$

Figure 27. Result of anisotropy strength $\varepsilon = 0.020$

Figure 28. Result of anisotropy strength $\varepsilon = 0.050$

Dimensionless Latent Heat K

Similarly, five different dimensionless latent heat were selected to conduct parametric study of pure water solidification at fixed dimensionless anisotropy strength $\varepsilon = 0.040$ according the Kobayashi's paper [3]. The energy released during the solidification process is latent heat, so it determines whether the condensation nucleus can form dendrites. The following pictures are the result of different dimensionless latent heat. According to the results, the dimensionless latent heat K needs to be greater than 1.6 to form a reasonable dendrite morphology.

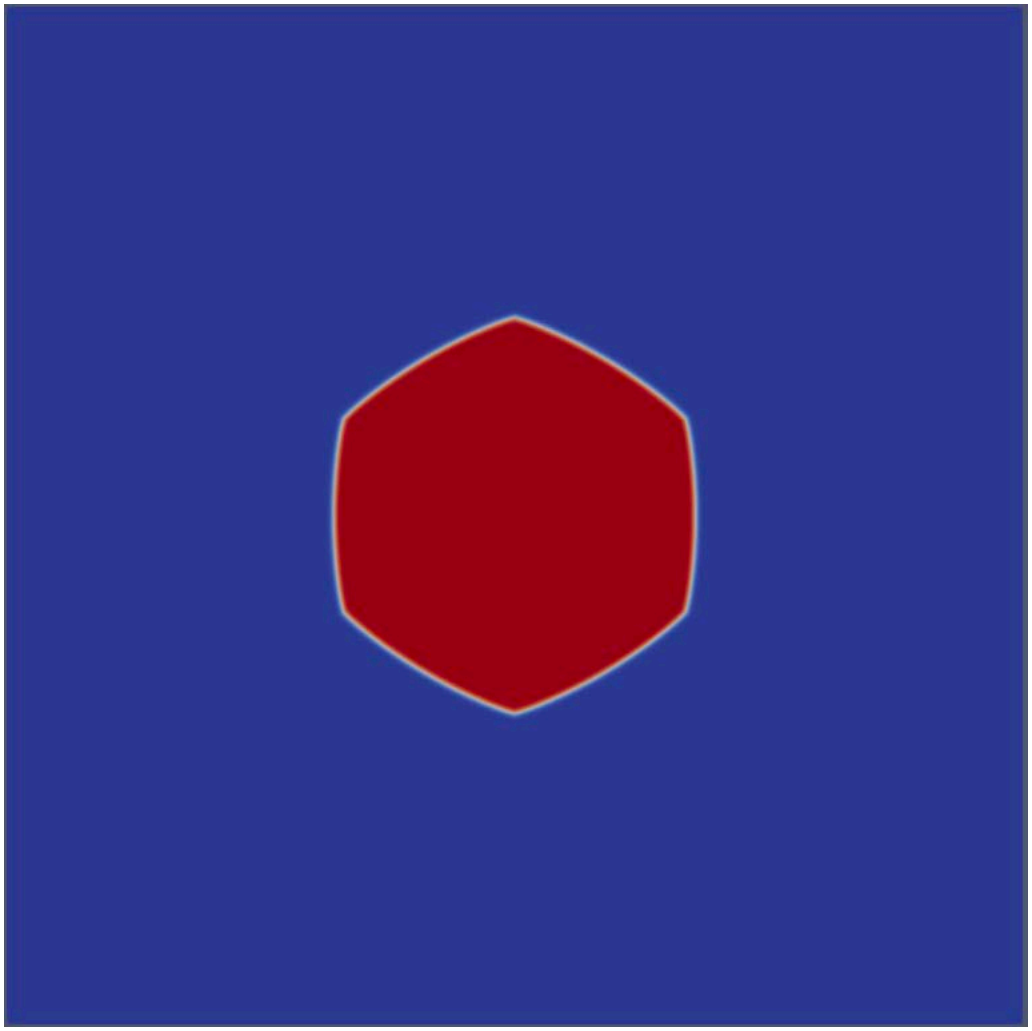

Figure 29. Result of dimensionless latent heat $K = 0.8$

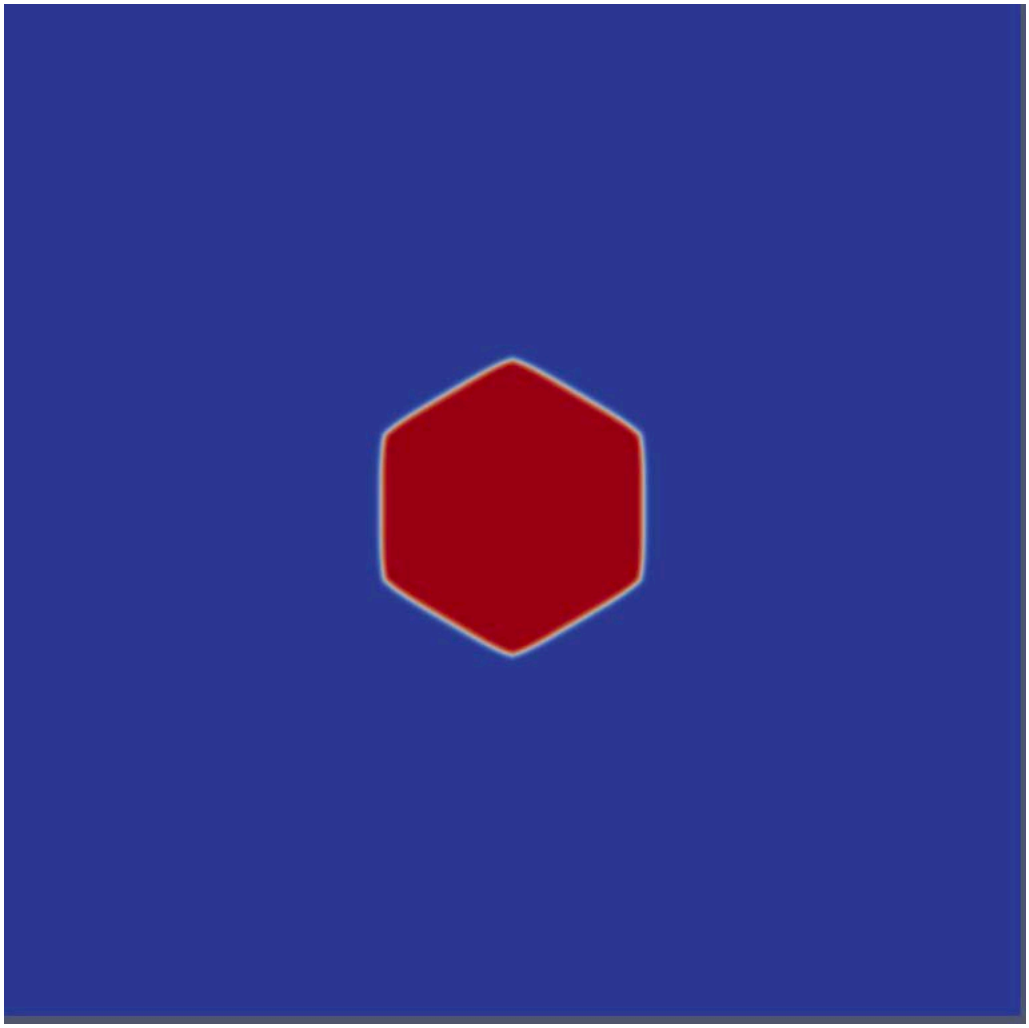

Figure 30. Result of dimensionless latent heat $K = 1.0$

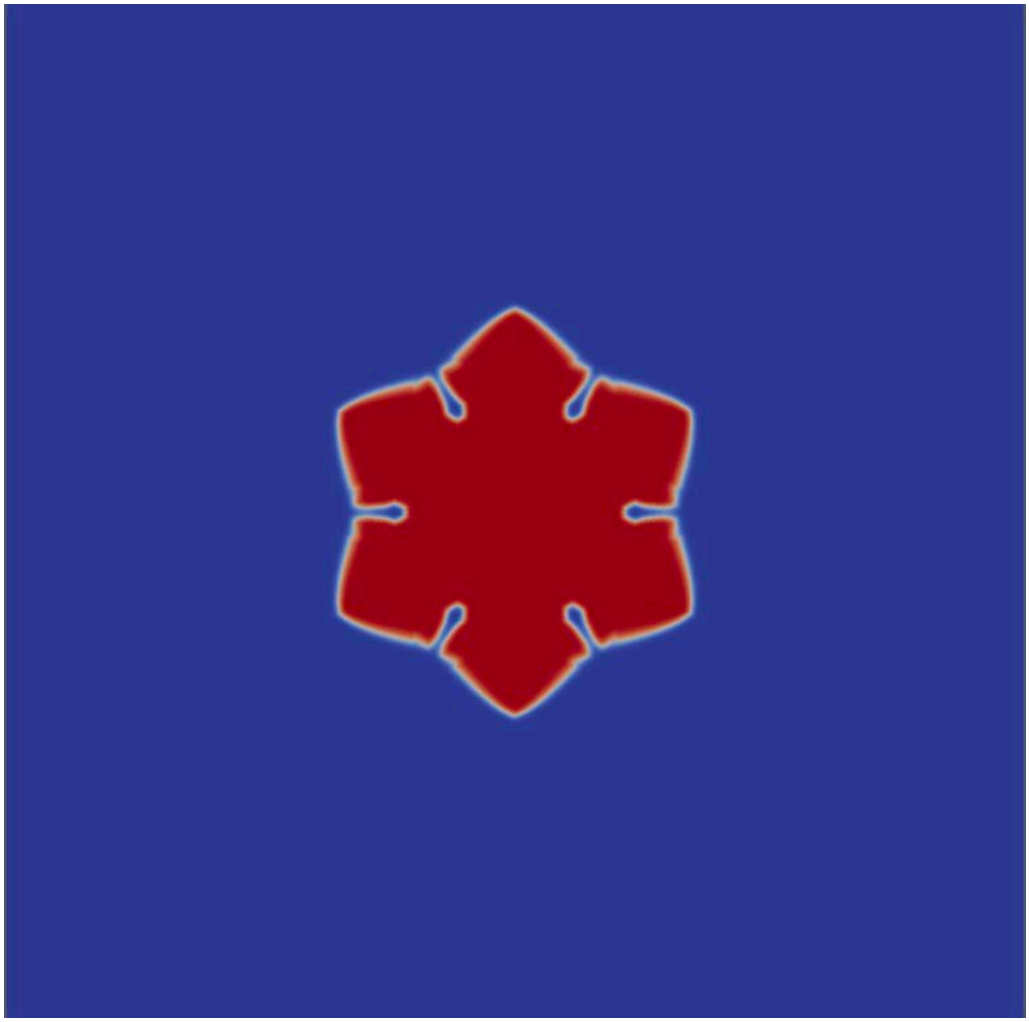

Figure 31. Result of dimensionless latent heat $K = 1.2$

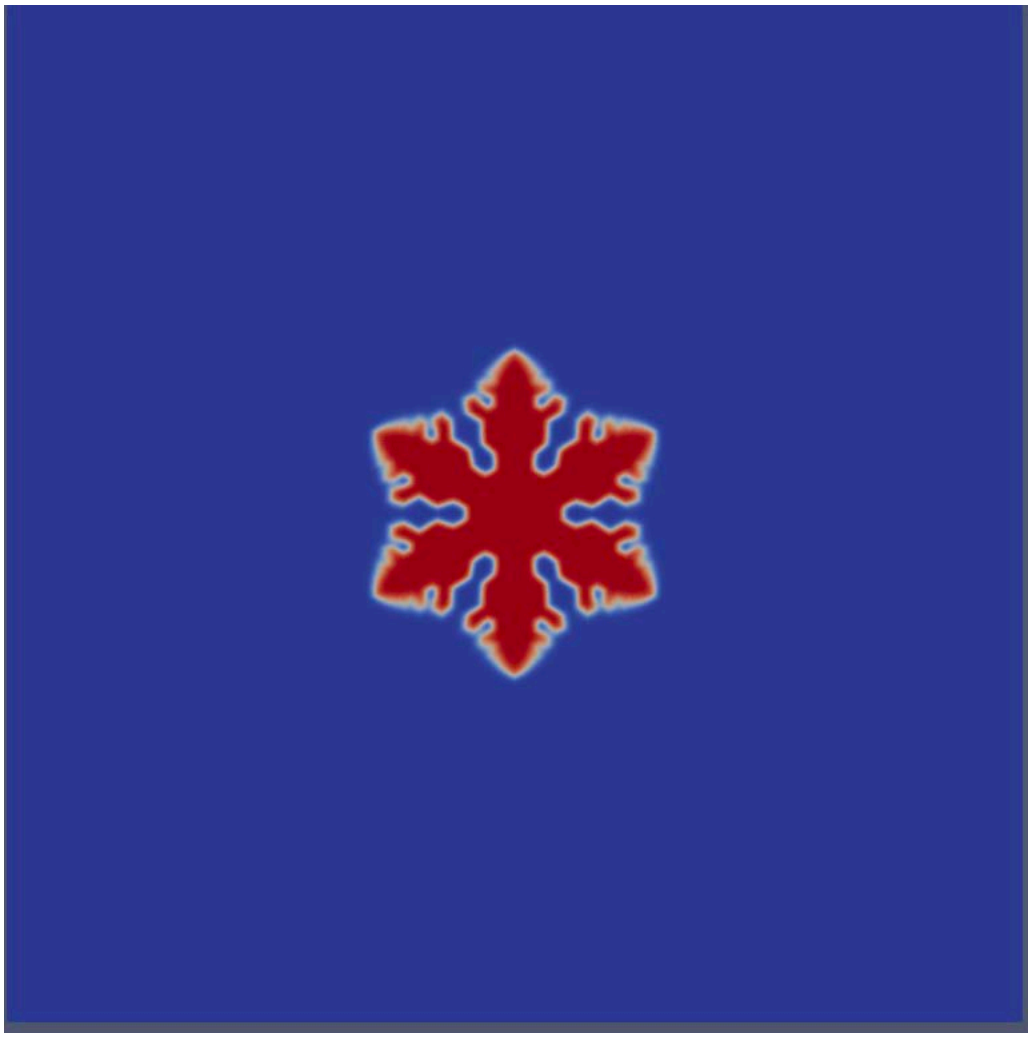

Figure 32. Result of dimensionless latent heat $K = 1.6$

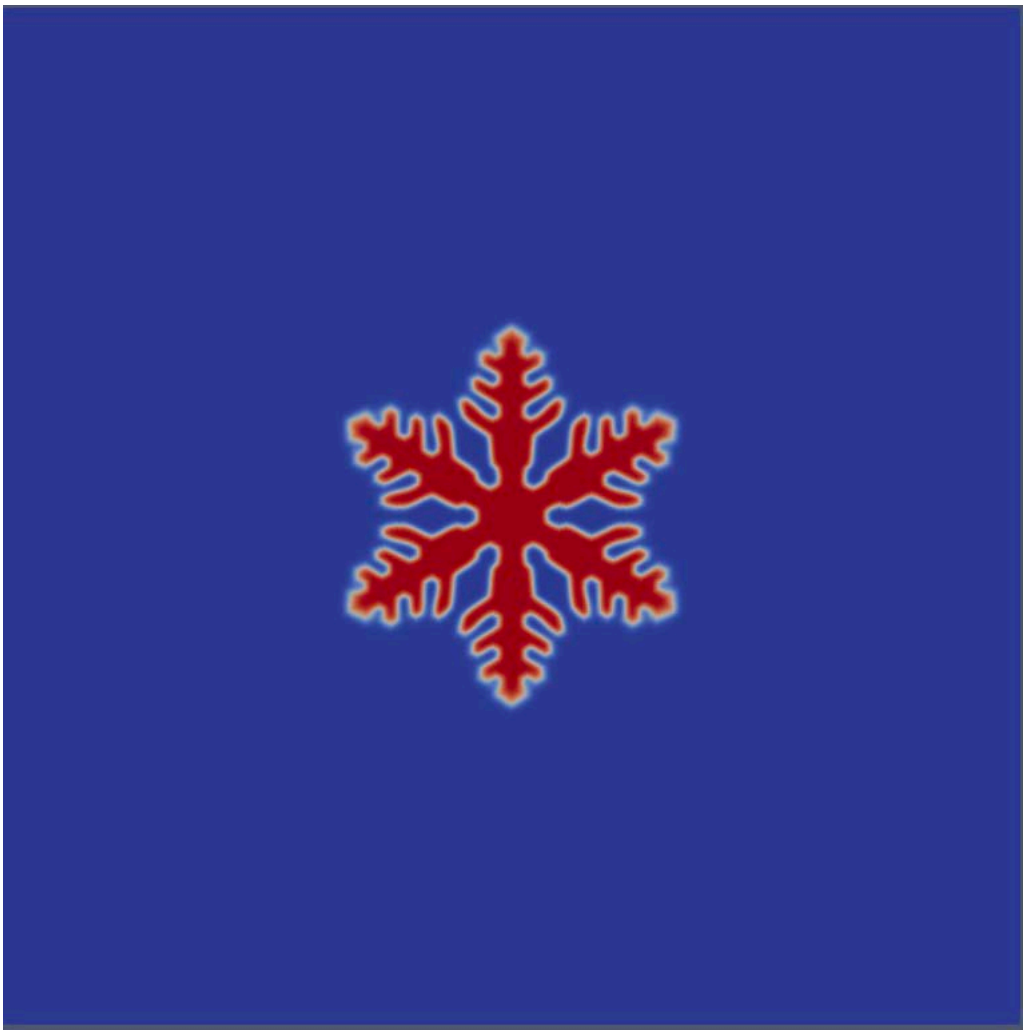

Figure 33. Result of dimensionless latent heat $K = 2.0$

4.1.3 Validation

The results of parametric study were validated with the results from Kobayashi's paper [3].

Figure 34 shows the comparison of the results at different anisotropy strength ε. The results on the right are Kobayashi's result in 1992. For the first simulation, where $\varepsilon = 0.000$, it is in perfect isotropic growth condition. The tip will split as the crystal grows. For $\varepsilon = 0.005$, the crystal structure has the features of both isotropic and dendritic structure. When $\varepsilon = 0.010$, the result shows the typical dendritic structure. The formation of side branches can be seen at the primary dendrite in the middle. For $\varepsilon = 0.020$, the simulation result on the left is not so consistent with the results on the right. This may be because when the anisotropic intensity is greater than 0.02, the dendrite morphology becomes complicated. For complex shapes, MOOSE needs more computing time to obtain the next simulation results. In the process of operation, some reasonable results in Kobayashi code may be ignored by MOOSE due to the large errors, resulting small shorter PDAS and SDAS in the left simulation results. But it has the features of dendrites tend to grow perpendicularly. For $\varepsilon = 0.050$, compared to $\varepsilon = 0.020$, competition between dendrites is clearly shown from the left result. And the side branches are longer than previous one. It is very clear that the side branch structure has highly dependency on anisotropy strength.

Figure 35 shows the comparison of the results at different dimensionless latent heat K. For the results using different values of K, the results are very consistent with the results of Kobayashi. For $K = 0.8$, the crystal structure tends to be convex. But as K increases, such tendency will disappear. For $K = 1.0$, the crystal has a tendency to grow in six directions. For $K = 1.2$, coarse snowflake pattern can be seen, and there are signs of dendrites on both sides of the tip. For $K = 1.6$, the crystal shows the side branch structure. As dimensionless latent heat K increases, the side branches will become thinner. At $K = 2.0$, we can get a typical snowflake pattern for pure water solidification.

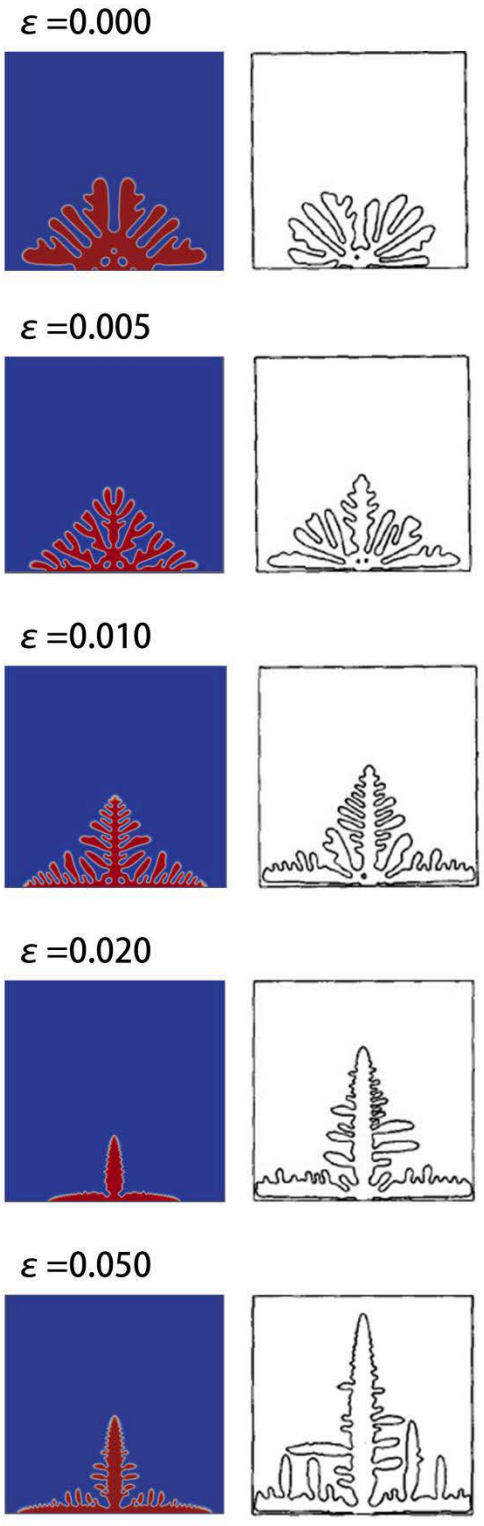

Figure 34. Comparison between results at different anisotropy strength ε .

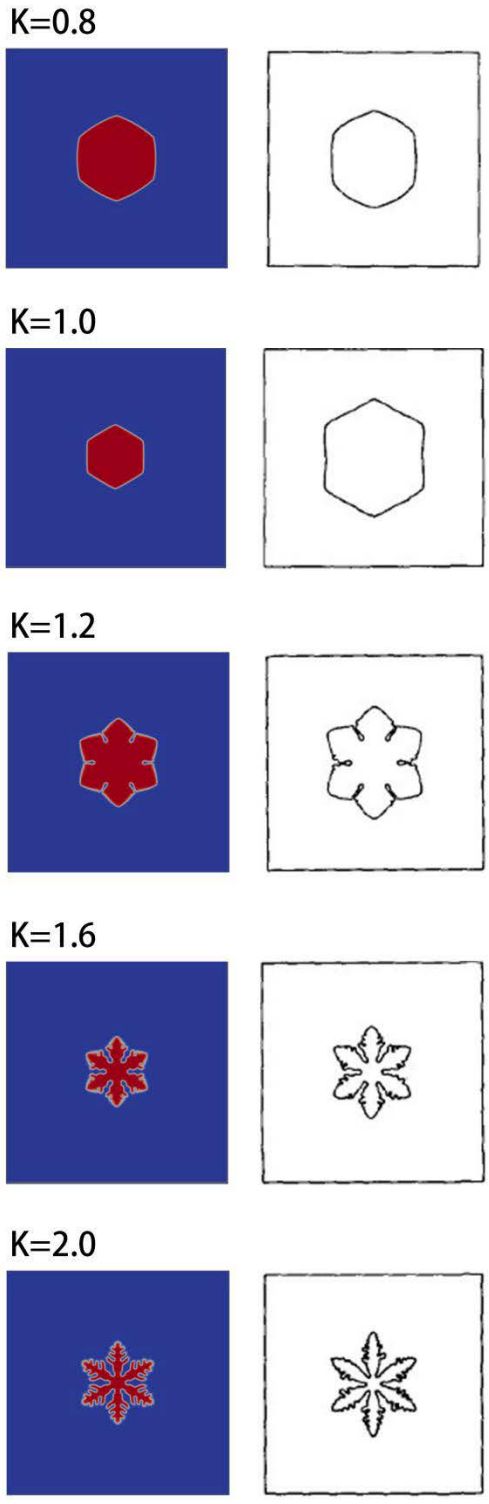

Figure 35. Comparison between results at different dimensionless latent heat K.

4.2 Results of Directional Alloy Solidification

4.2.1 Simulation results

The 500×500 computational domain can accommodate up to three dendrites for competitive growth. Therefore, three typical repetitive simulations were conducted in order to improve accuracy and credibility of results. The results are shown below. The PDAS are measured by using visualization tool Paraview.

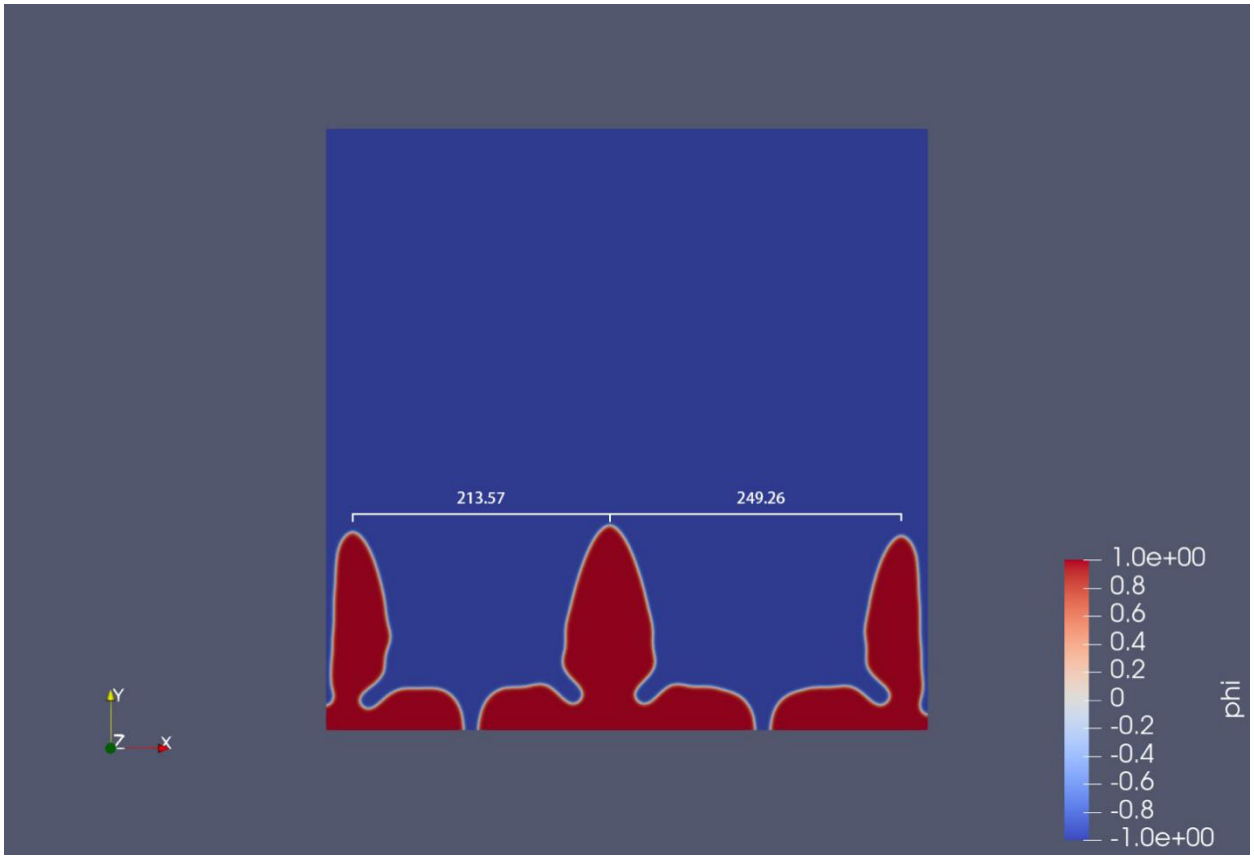

Figure 36. simulation #1 of directional alloy solidification.

The primary dendrite arm spacings (PDAS) in this simulation are 213.57 and 249.26.

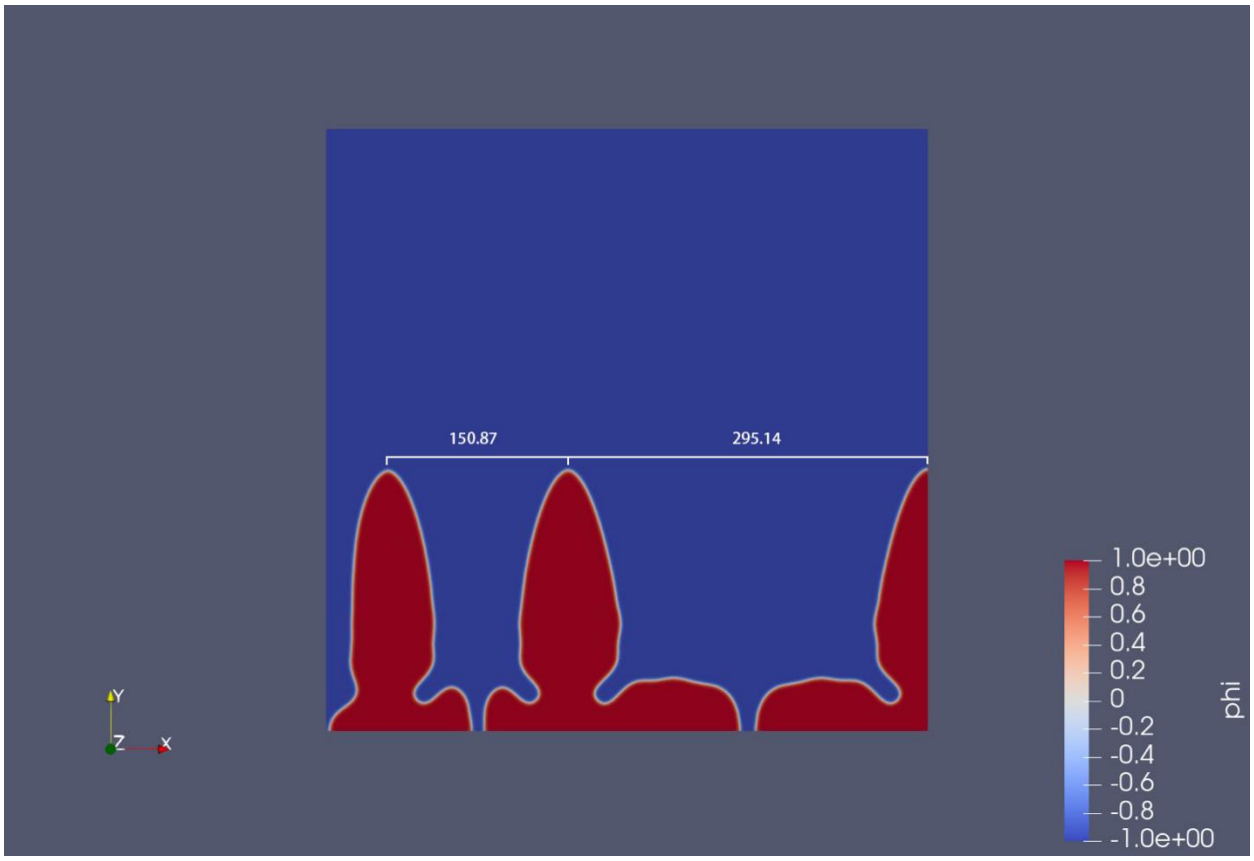

Figure 37. simulation #2 of directional alloy solidification.

The primary dendrite arm spacings (PDAS) in this simulation are 150.87 and 295.14.

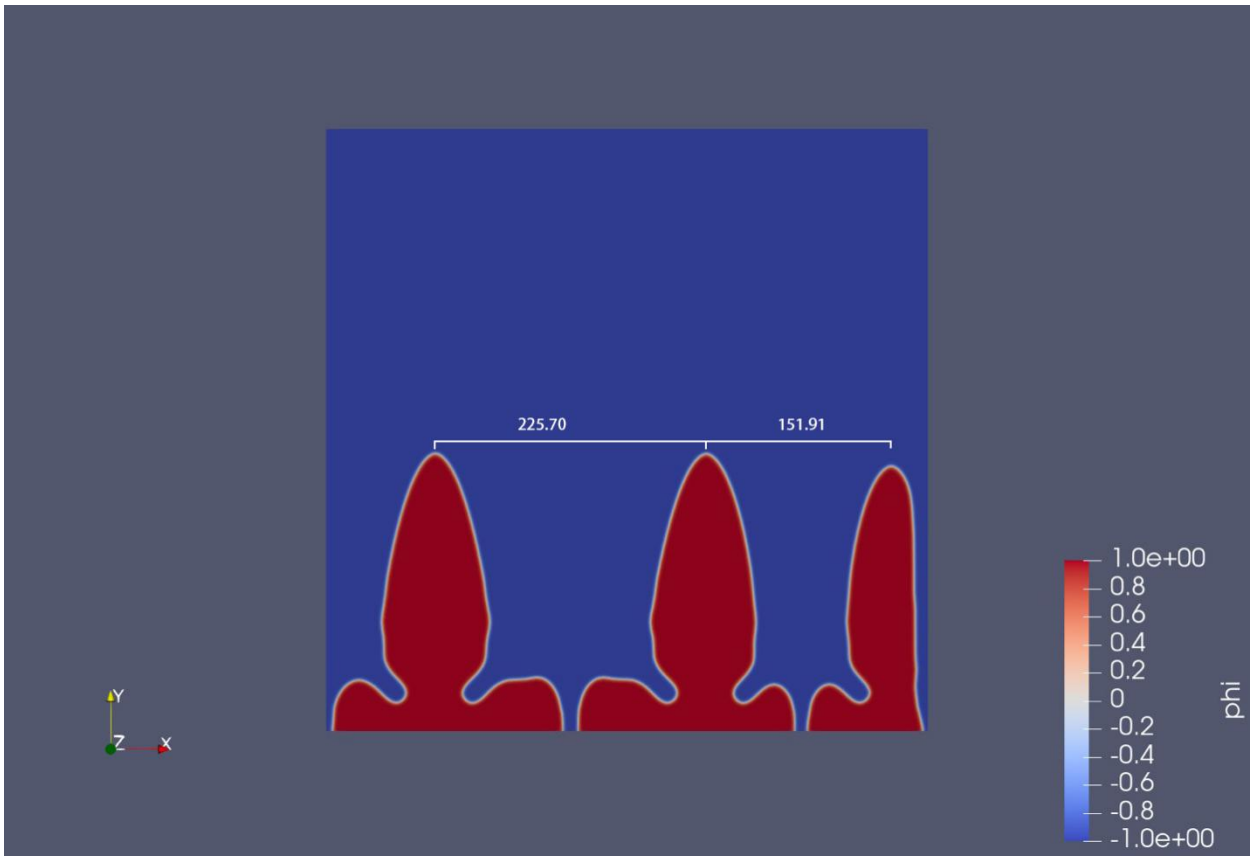

Figure 38. simulation #3 of directional alloy solidification.

The primary dendrite arm spacings (PDAS) in this simulation are 225.70 and 151.91.

4.2.2 Parametric Study

In this section, parametric study will focus on temperature gradient G and cooling rate R which closely related to steel continuous casting process.

Temperature Gradient

All the parameters except temperature gradient G are fixed and G is increasing gradually from 3700K/m.

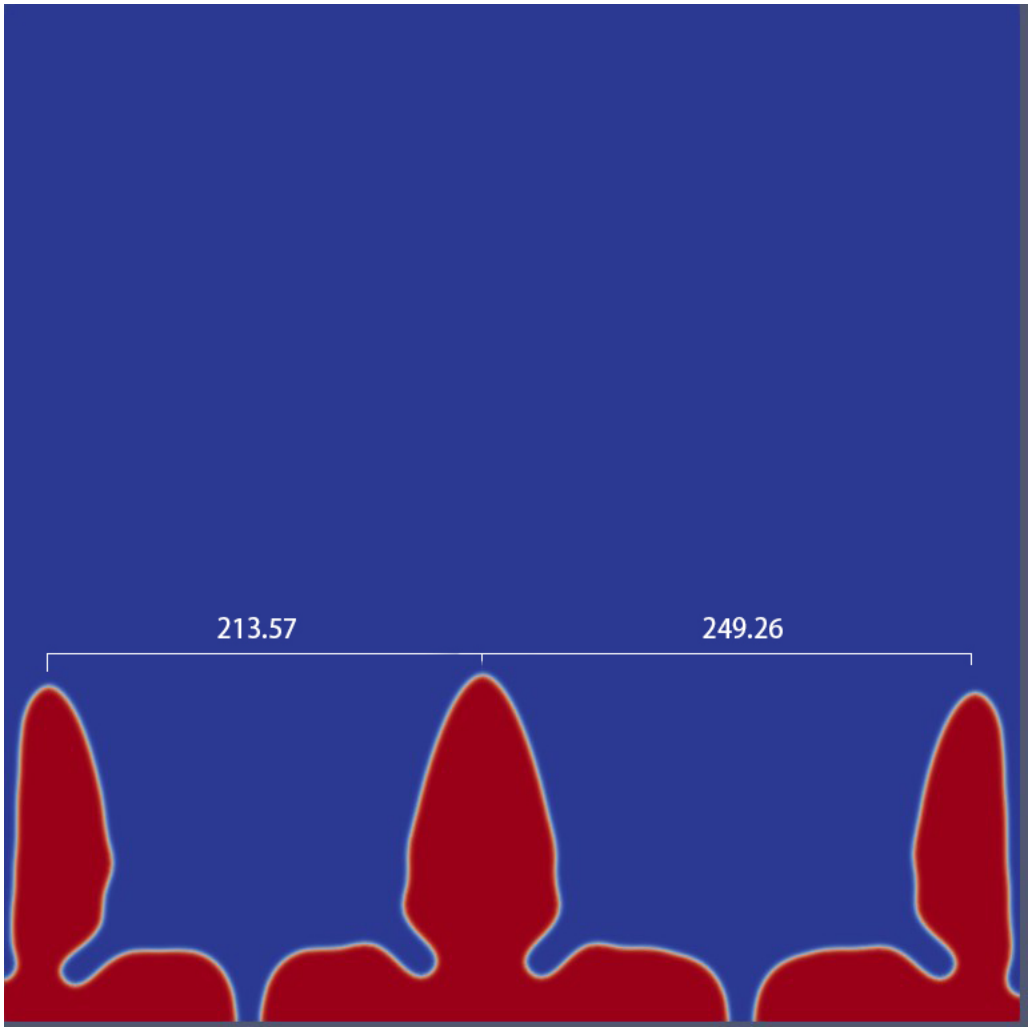

Figure 39. Result of temperature gradient $G = 3700K/m$

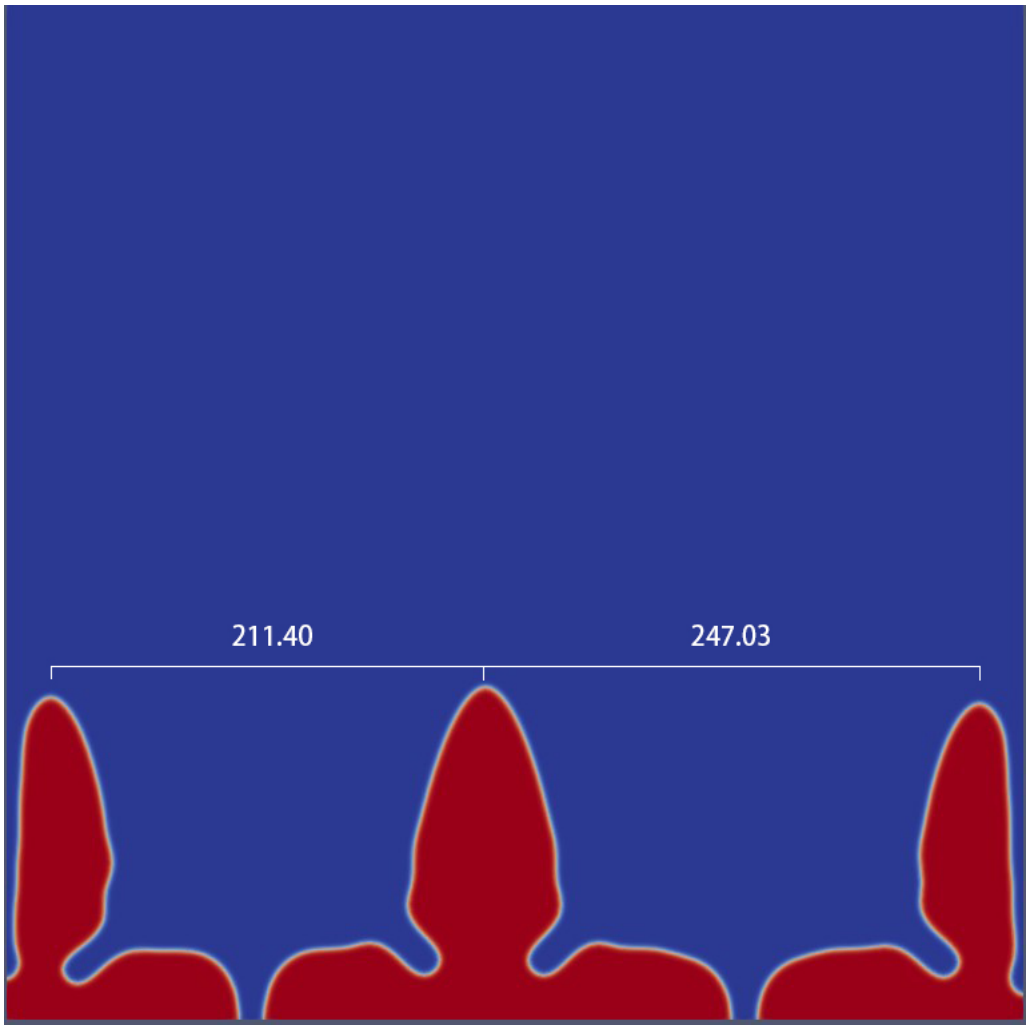

Figure 40. Result of temperature gradient $G = 3800K/m$

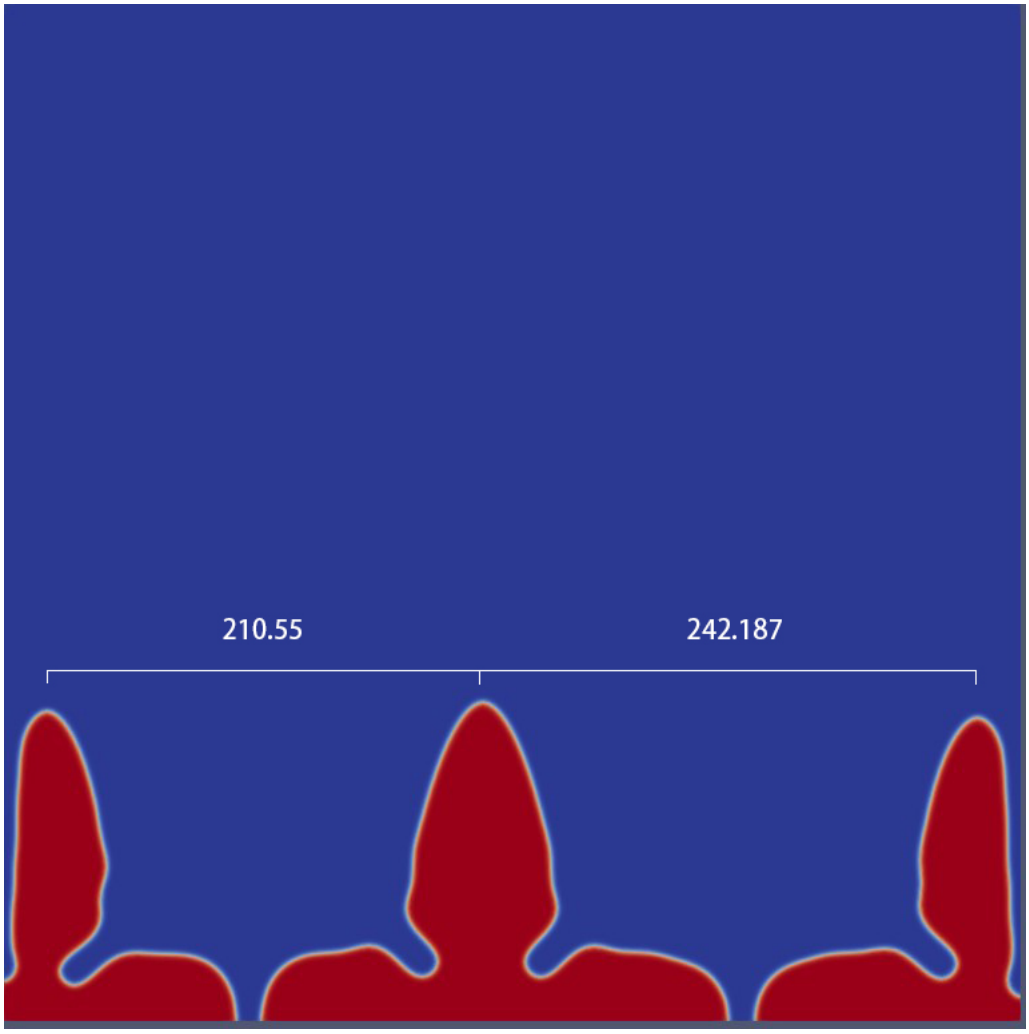

Figure 41. Result of temperature gradient $G = 3900K/m$

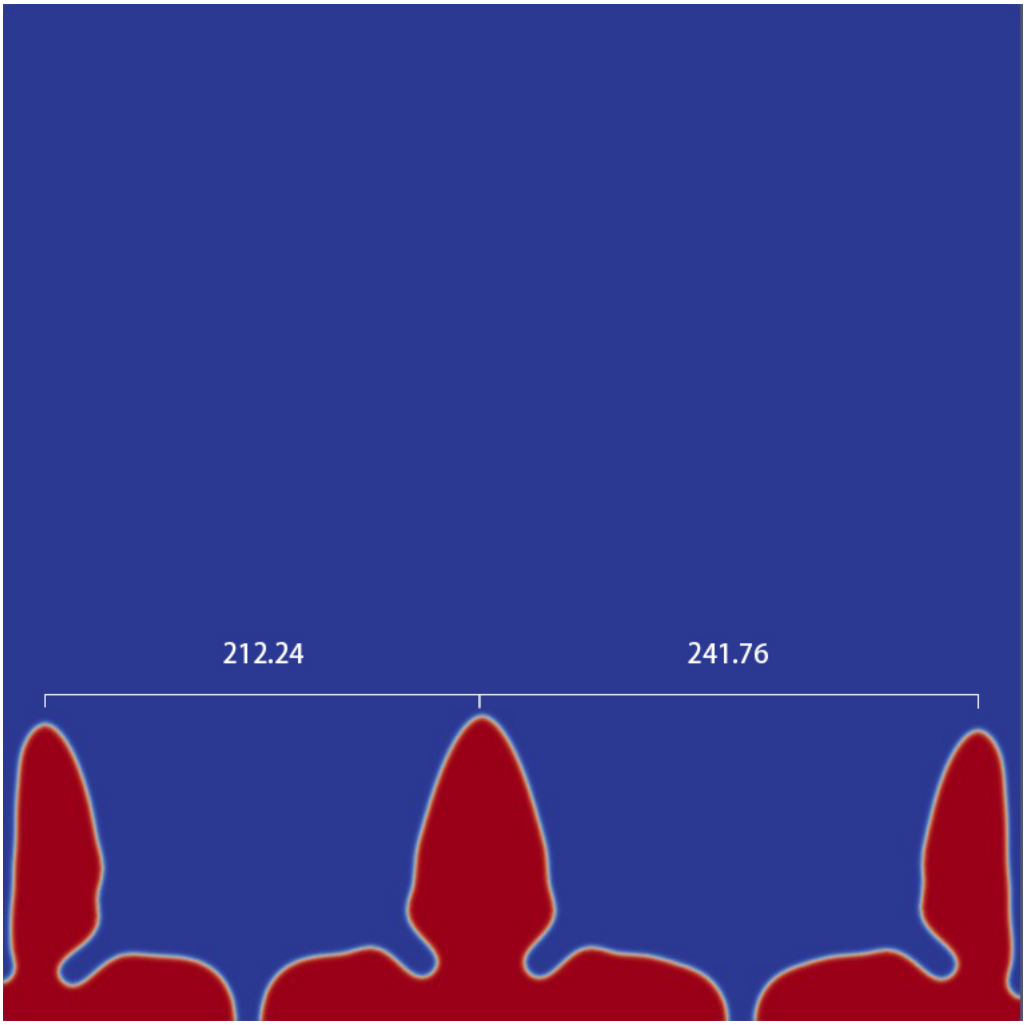

Figure 42. Result of temperature gradient $G = 4000K/m$

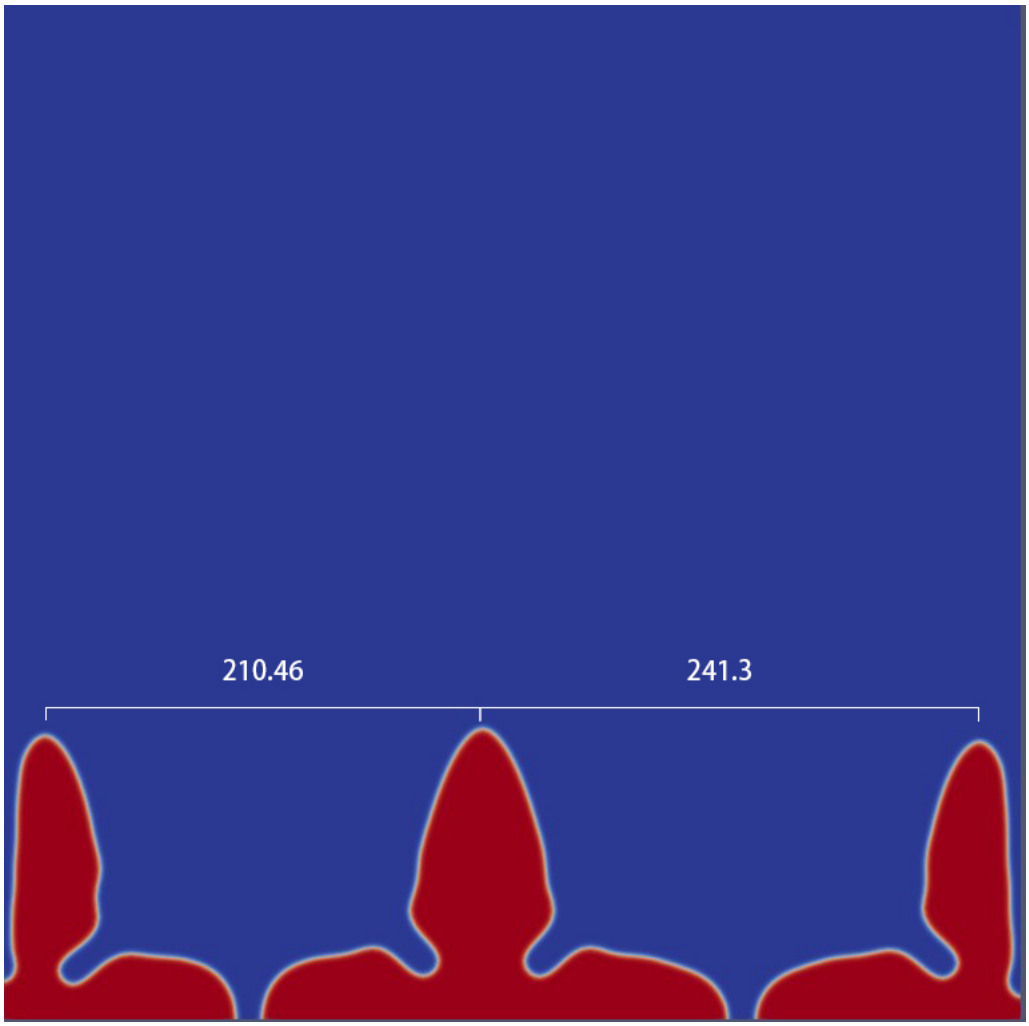

Figure 43. Result of temperature gradient $G = 4100K/m$

Cooling Rate

All the parameters except cooling rate R are fixed and R is increasing gradually from 0.045K/m.

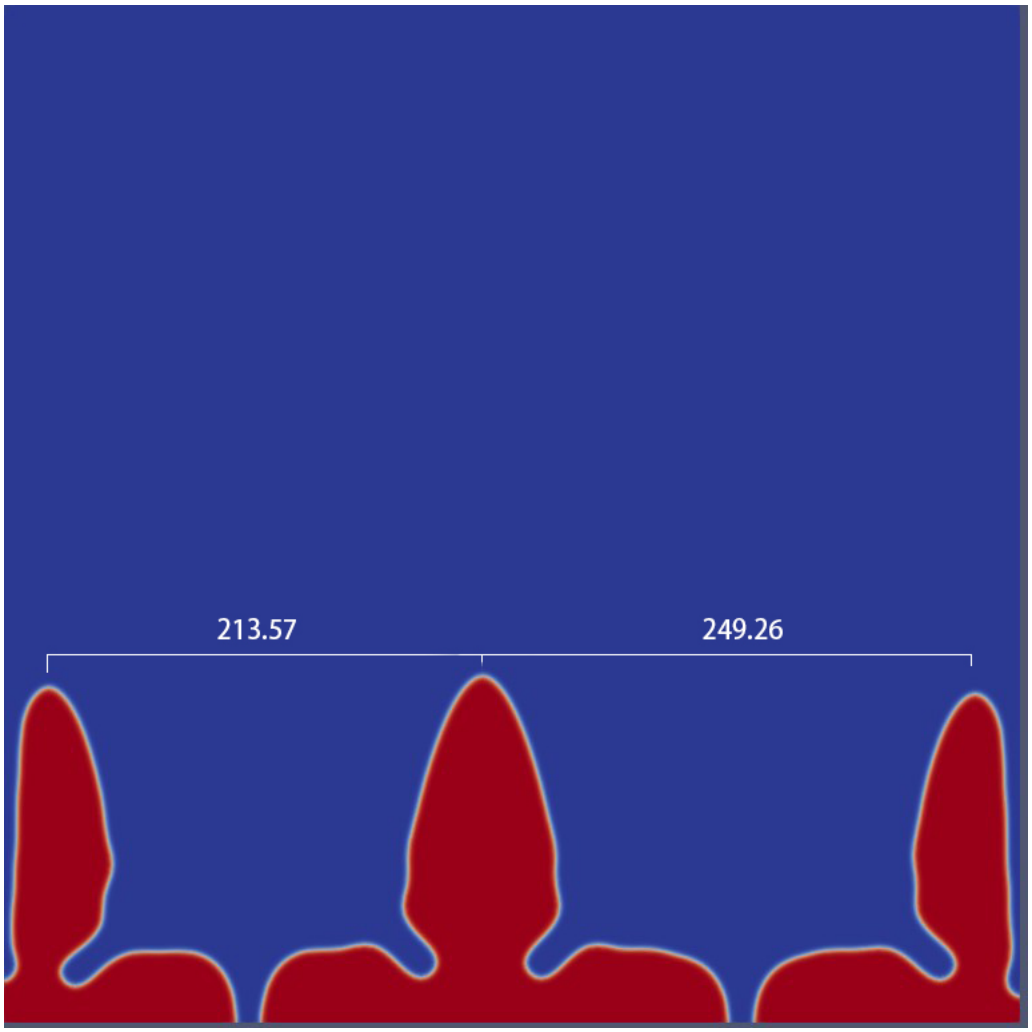

Figure 44. Result of cooling rate $R = 0.045K/s$

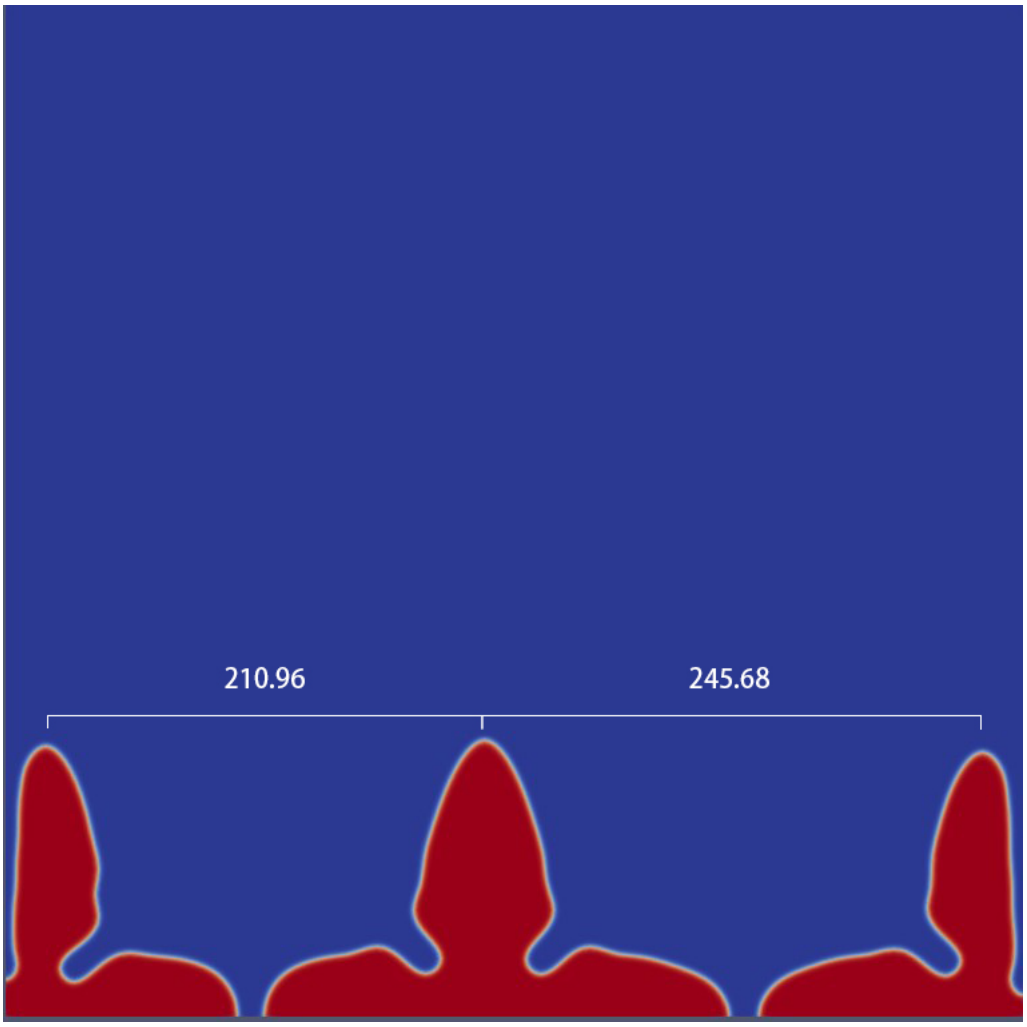

Figure 45. Result of cooling rate $R = 0.050 K/s$

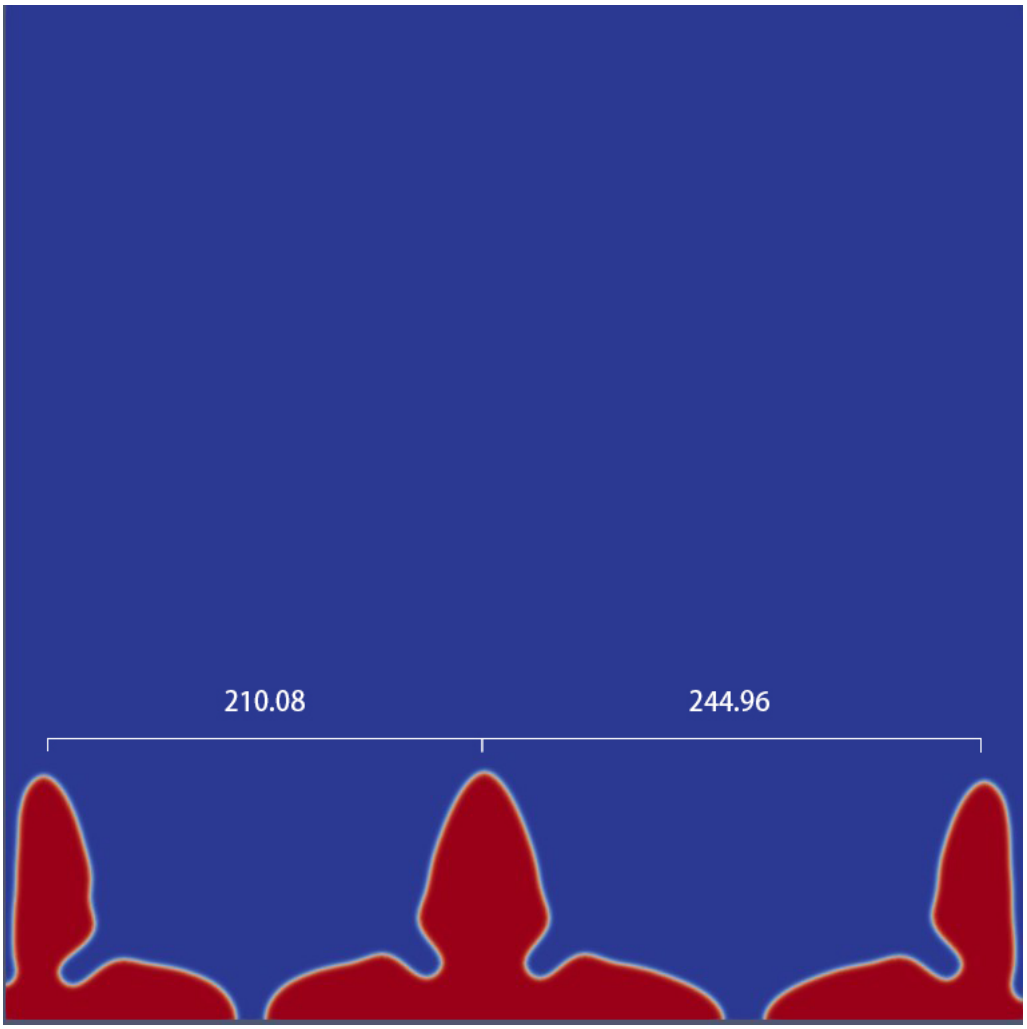

Figure 46. Result of cooling rate $R = 0.055K/s$

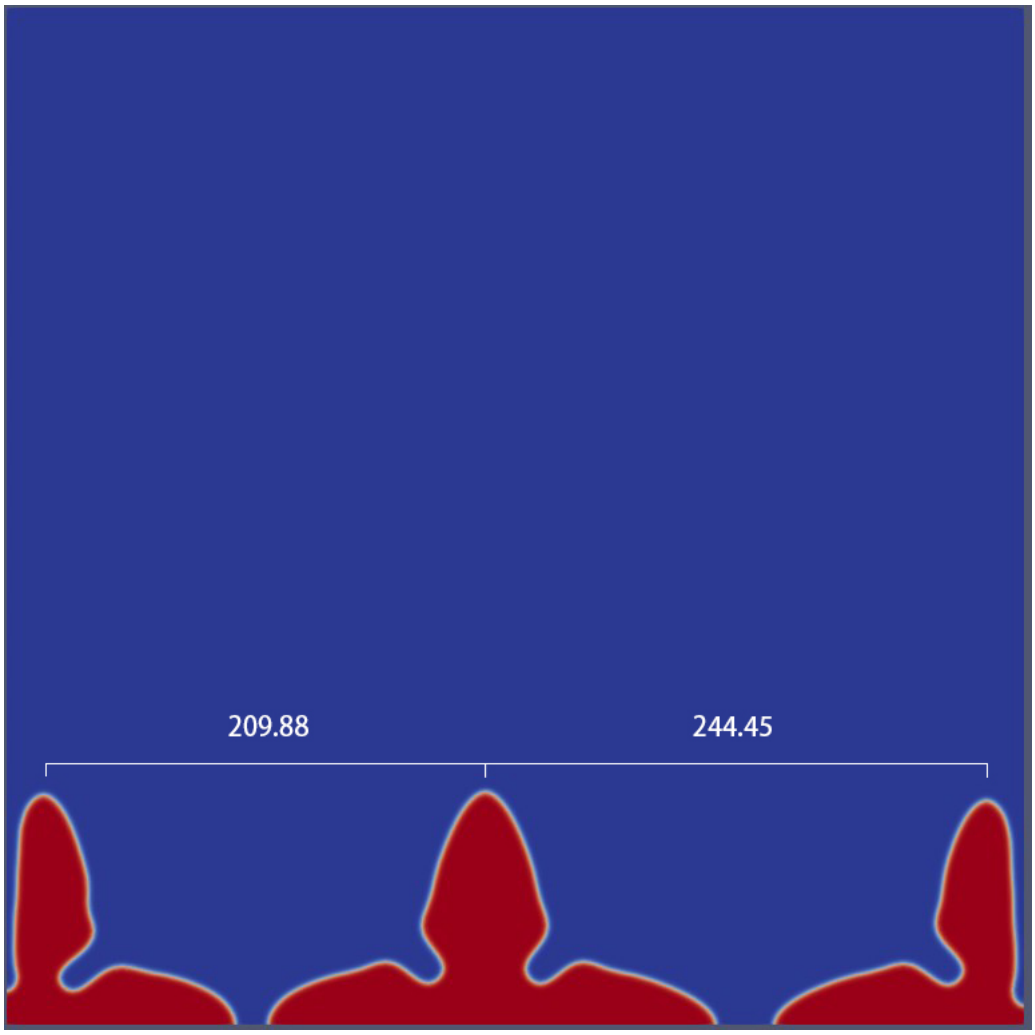

Figure 47. Result of cooling rate $R = 0.060 K/s$

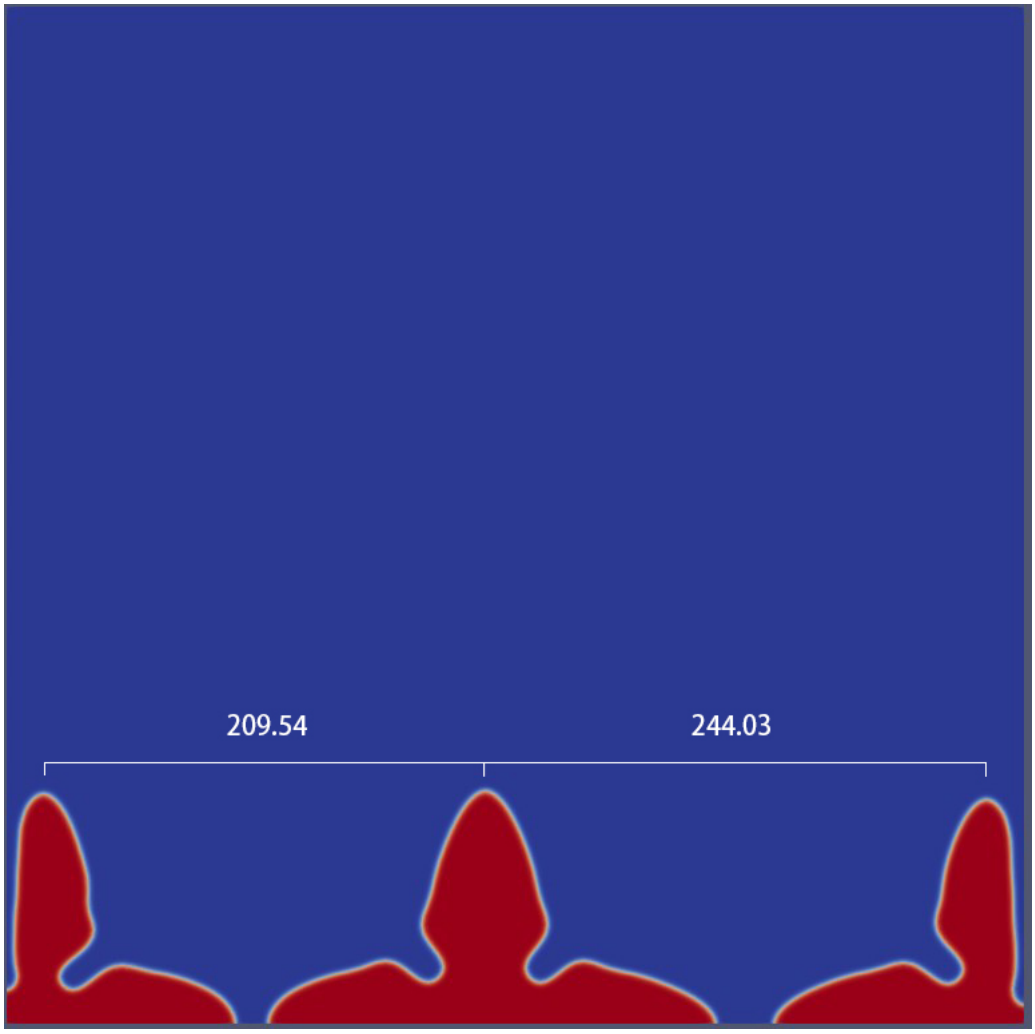

Figure 48. Result of cooling rate $R = 0.065K/s$

In the continuous casting process, process parameters will affect the size and morphology of primary dendrite arm spacing (PDAS), secondary dendrite arm spacing (SDAS) and primary dendrite arm aspect ratio. The continuous casting process parameters include supercooling, secondary cooling specific water volume, pulling speed, electromagnetic stirring, etc. And at the same time, Carbon content and other factors will also affect the primary dendrite spacing, secondary dendrite spacing and primary dendrite arm aspect ratio. It has been proved that as the cooling rate and temperature gradient increase, the PDAS will gradually decrease. As the cooling rate and temperature gradient increase, the primary dendrite arm spacing will gradually decrease. The simulation results basically conform to this law.

4.2.3 Validation

From simulation results in section 4.2.1. Mean value of PDAS can be obtained:

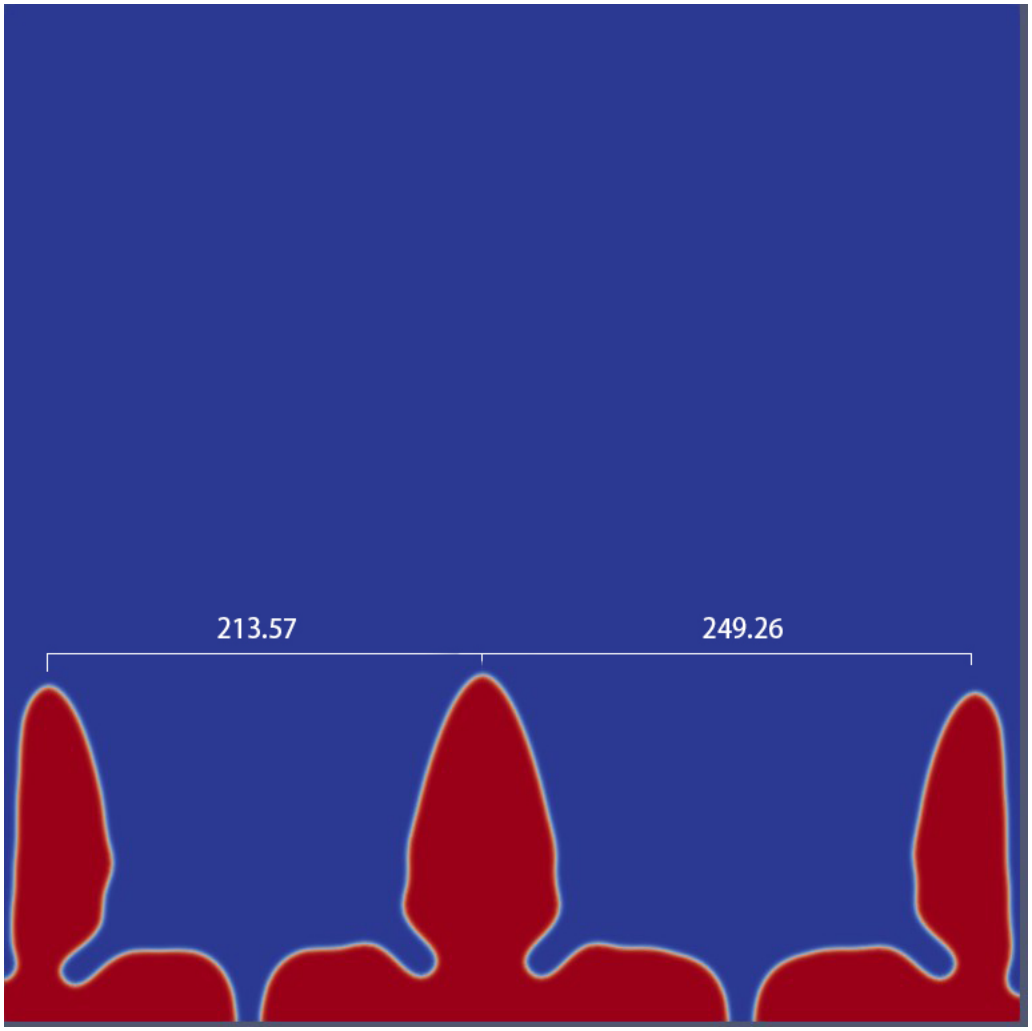

Figure 49. PDAS in simulation #1

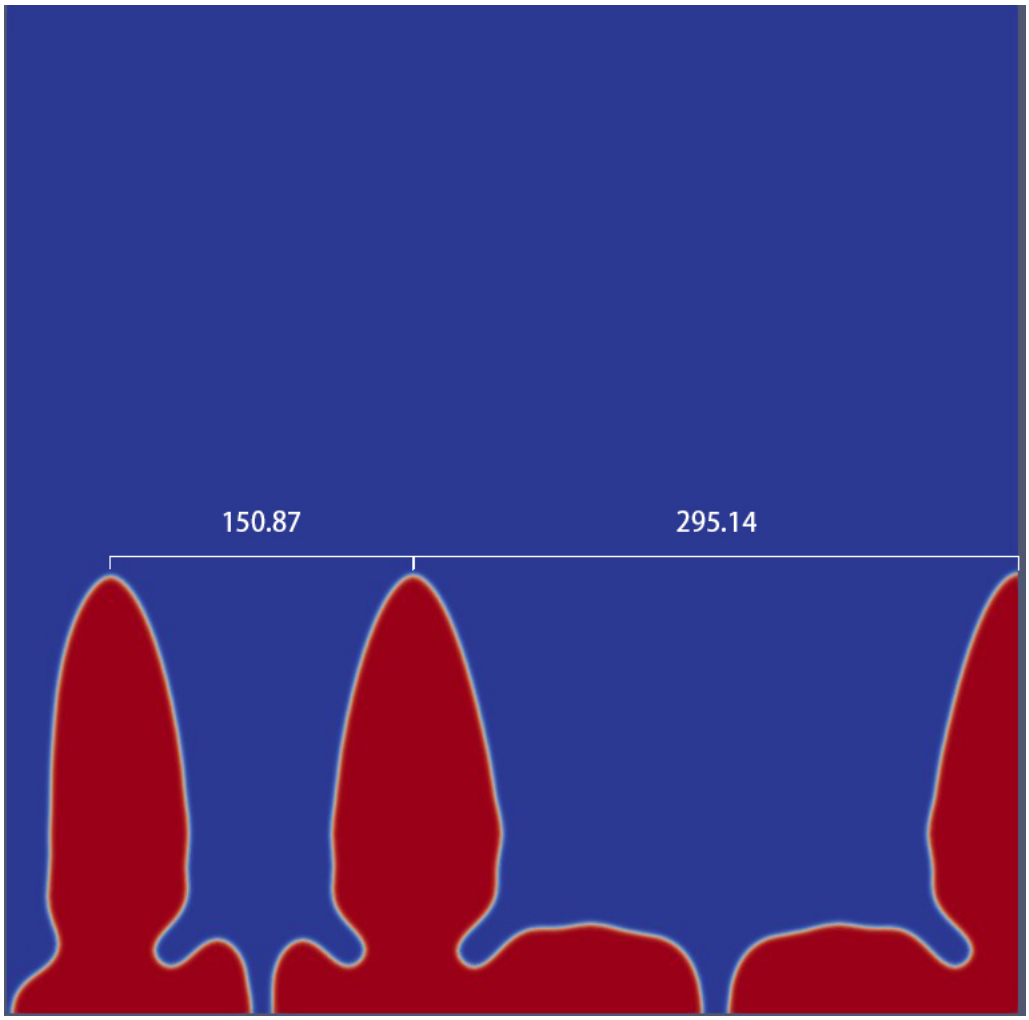

Figure 50. PDAS in simulation #2

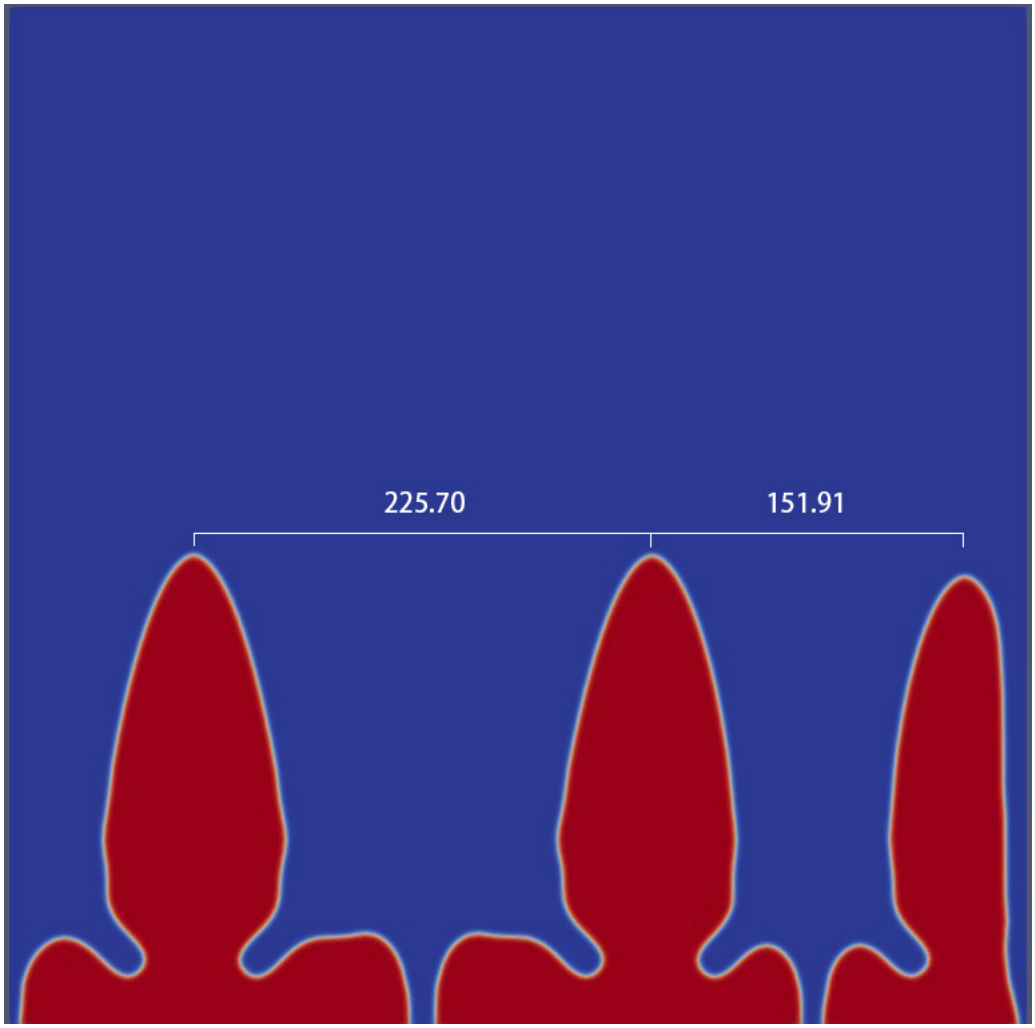

Figure 51. PDAS in simulation #3

$$\overline{\lambda_1} = \frac{213.57 + 249.26 + 150.87 + 295.14 + 225.70 + 151.91}{6} = 214.41$$

The length in the directional alloy simulation was scaled by interface layer width W. Hence, the value of PDAS in SI unit is:

$$\overline{\lambda_1} \times W = \overline{\lambda_1} \times \xi d_0 = \overline{\lambda_1} \times \frac{\xi\Gamma}{m_l c_0 (k-1)} = 5.63 \times 10^{-4} m = 563 \mu m$$

According to Won Y.M and Thomas B.G's paper [17], their measured PDAS for unidirectionally solidified crystal is 360 μm. Besides, their empirical formula [18] for low carbon steel to determine the PDAS is:

$$\lambda_1 = 278.748 \times R^{-2.06277638} \times c_0^{-0.316225+2.0325c_0} \tag{36}$$

Here, R is the cooling rate and c_0 is the initial carbon concentration. By using the parameter values from Table 2. PDAS can be calculated as:

$$\lambda_1 = 278.748 \times 0.045^{-2.06277638} \times 0.13^{-0.316225+2.0325\times0.13} = 587.62\mu m$$

The deviation between the simulated value and the calculated value is:

$$\frac{563 - 587.62}{587.62} \times 100\% = -4.19\%$$

For the model that only considers part of the continuous casting process parameters, the degree of deviation is acceptable.

4.3 Discussions

The models provided in this article is a prototype of the phase field method developed using finite element software. It has the basic functions of showing the evolution of microscopic morphology, roughly showing the temperature field distribution and obtaining dendrite structure parameters. These functions can help to carry out or verify the macro simulation. Through the combination of these basic functions and experiments, researchers can accumulate experimental data to form a database of microstructures to further accurately predict or directly provide related crystal parameters.

Due to the complexity of the equations used in the phase field method, the method used in this article to classify the equations in the code to achieve modularity. This also means that this code has the potential for subsequent development. For example, for the directional solidification of the alloy, the morphology of the secondary dendrites cannot be well displayed due to the absence of noise term. When open source software provides this function, the function can be easily added to the code to get a better morphology. And accurate microstructure morphology can be applied to particle entrapment in industry.

In addition to being used in continuous casting of steel, many emerging technologies are now also used in industry. The model has been shown to simulate the evolution of microstructure in these emerging technologies by coupling a series of governing equations. For example, laser manufacturing technology and additive manufacturing technology. Simulation of microstructures helps researchers and engineers predict or even directly design the function of materials.

5. CONCLUSION

Simple analytical models of pure material solidification and directional alloy solidification based on phase field method have been developed, which involves the effect of anti-trapping and difference of diffusion rate in solid and liquid.

The results of pure material solidification have highly dependence on anisotropy strength and dimensionless latent heat. The anisotropic strength determines the crystal growth mode and the features of dendrites. Dimensionless latent heat determines whether the crystal can branch to grow. High anisotropy strength and dimensionless latent will make the crystal tend to grow more primary dendrites, secondary dendrites and even tertiary dendrites, resulting complicated shapes of crystal. From the results of parametric study, it can be seen that the values of $\varepsilon = 0.02$ and $K = 1.8$ set by the simulations are also relatively reasonable.

The results of directional alloy solidification are reasonable with deviation equals to -4.19%. With the change of temperature gradient and cooling rate, the change of PDAS conforms to the rule obtained in the experiments.

The simple model presented here can be effectively integrated into macro simulations, for example to help simulate the mushy zone in primary cooling section.

APPENDIX A. EXAMPLE OF DIMENSIONLESS PROCEDURE

Original Equation (SI Units):

$$\tau_0\left[1+\frac{Gy-Rt}{m_l c_0}\right]a_s^2\frac{\partial\varphi}{\partial t}=\nabla[a_s^2\nabla\varphi]+\frac{\partial}{\partial x}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial x}\right)}\right)+\frac{\partial}{\partial y}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial y}\right)}\right)+\varphi-\varphi^3-$$
$$\lambda(1-\varphi^2)^2(U-\frac{Gy-Rt}{mlc_0})$$

Dimensionless Equation:

$$\left[1-(1-k)\frac{y-V_p t}{l_T}\right]a_s^2\frac{\partial\varphi}{\partial t}=\nabla[a_s^2\nabla\varphi]+\frac{\partial}{\partial x}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial x}\right)}\right)+\frac{\partial}{\partial y}\left(|\nabla\varphi^2|a_s\frac{\partial a_s}{\partial\left(\frac{\partial\varphi}{\partial y}\right)}\right)+\varphi-\varphi^3-$$
$$\lambda(1-\varphi^2)^2(U+\frac{y-V_p t}{l_T})$$

The dimensionlee equation is derived from original equation in SI units. Here the LHS of original equation is taken as an example to carry out the dimensionless procedure.

First, we define the dimesionless parameter:

$$\begin{cases}\tilde{t}=\dfrac{t}{\tau}\\ \widetilde{V_p}=\dfrac{V_p\tau}{W}=\dfrac{R\tau}{GW}\\ \tilde{y}=\dfrac{y}{W}\\ \widetilde{l_T}=\dfrac{l_T}{W}=\dfrac{|m|c_0(1-k)}{GW}\end{cases}$$

$$\tau\left[1+\frac{Gy-Rt}{m_l c_0}\right]a_s^2\frac{\partial\varphi}{\partial t}=\tau\left[1+\frac{Gy-Rt}{m_l c_0}\right]a_s^2\frac{\partial\varphi}{\partial(\tau_0\tilde{t})}=\left[1+\frac{Gy-Rt}{m_l c_0}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}=\left[1+\frac{y-\frac{R}{G}t}{\frac{m_l c_0}{G}}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}$$

$$=\left[1-(1-k)\frac{y-V_p t}{\frac{m_l c_{0(1-k)}}{G}}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}(m<0)=\left[1-\left(1-k\right)\frac{y-V_p t}{l_T}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}$$

$$=\left[1-(1-k)\frac{\tilde{y}*W-\widetilde{V_p}*\frac{W}{\tau}*\tau_0\tilde{t}}{\widetilde{l_t}*W}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}$$

$$=\left[1-(1-k)\frac{\tilde{y}-\widetilde{V_p}\tilde{t}}{\widetilde{l_t}}\right]a_s^2\frac{\partial\varphi}{\partial\tilde{t}}$$

APPENDIX B. CODE FOR PURE MATERIAL SOLIDIFICATION

1. Simulation execution file for pure water solidification：

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 14
  ny = 14
  xmax = 9
  ymax = 9
  uniform_refine = 3
[]

[Variables]
  [./phi]
  [../]
  [./T]
  [../]
[]

[ICs]
  [./phiIC]
```

```
    type = SmoothCircleIC
    variable = phi
    int_width = 0.1
    x1 =  4.5
    y1 = 0
    radii = 0.07
    outvalue = 0
    invalue = 1
  [../]
[]

[Kernels]
  [./phi_dot]
    type = TimeDerivative
    variable = phi
  [../]
  [./anisoACinterface1]
    type = ACInterfaceKobayashi1
    variable = phi
    mob_name = M
  [../]
  [./anisoACinterface2]
    type = ACInterfaceKobayashi2
```

```
  variable = phi
  mob_name = M
[../]
[./AllenCahn]
  type = AllenCahn
  variable = phi
  mob_name = M
  f_name = fbulk
  args = T
[../]
[./T_dot]
  type = TimeDerivative
  variable = T
[../]
[./CoefDiffusion]
  type = Diffusion
  variable = T
[../]
[./phi_dot_T]
  type = CoefCoupledTimeDerivative
  variable = T
  v = phi
  coef = -1.8
```

```
  [../]
[]


[Materials]
  [./free_energy]
    type = DerivativeParsedMaterial
    f_name = fbulk
    args = 'phi T'
    constant_names = pi
    constant_expressions = 4*atan(1)
    function = 'm:=0.9 * atan(10 * (0 - T)) / pi; 1/4*w^4 - (1/2 - m/3) * w^3 + (1/4 - m/2) * w^2'
    derivative_order = 2
    outputs = exodus
  [../]
  [./material]
    type = InterfaceOrientationMaterial
    op = phi
    mode_number = 6
  [../]
  [./consts]
    type = GenericConstantMaterial
    prop_names  = 'M'
    prop_values = '3333.333'
```

```
  [../]
[]

[Preconditioning]
  [./SMP]
    type = SMP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  scheme = bdf2
  solve_type = PJFNK
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart'
  petsc_options_value = 'hypre    boomeramg      31'

  nl_abs_tol = 1e-10
  nl_rel_tol = 1e-08
  l_max_its = 30

  end_time = 1
```

```
  [./TimeStepper]
    type = IterationAdaptiveDT
    optimal_iterations = 6
    iteration_window = 2
    dt = 0.0005
    growth_factor = 1.1
    cutback_factor = 0.75
  [../]
  [./Adaptivity]
    initial_adaptivity = 3 # Number of times mesh is adapted to initial condition
    refine_fraction = 0.7 # Fraction of high error that will be refined
    coarsen_fraction = 0.1 # Fraction of low error that will coarsened
    max_h_level = 5 # Max number of refinements used, starting from initial mesh (before uniform refinement)
    weight_names = 'w T'
    weight_values = '1 0.5'
  [../]
[]

[Outputs]
  interval = 5
  exodus = true
[]
```

2. Simulation execution file for pure iron solidification:

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 14
  ny = 14
  xmax = 9
  ymax = 9
  uniform_refine = 3
[]

[Variables]
  [./phi]
  [../]
  [./T]
  [../]
[]

[ICs]
  [./phiIC]
    type = SpecifiedSmoothCircleIC
    variable = phi
    int_width = 0.1
```

```
    x_positions = '1 4.5 7.5'
    y_positions = '0 0 0'
    z_positions = '0 0 0'
    radii = '0.05 0.07 0.10'
    outvalue = 0
    invalue = 1
  [../]
[]

[BCs]
  [./bottom_T]
    type = DirichletBC
    variable = T
    boundary = 0
    value = 1539
  [../]
[]


[Kernels]
  [./phi_dot]
    type = TimeDerivative
    variable = phi
```

```
[../]
[./anisoACinterface1]
  type = ACInterfaceKobayashi1
  variable = phi
  mob_name = M
[../]
[./anisoACinterface2]
  type = ACInterfaceKobayashi2
  variable = phi
  mob_name = M
[../]
[./AllenCahn]
  type = AllenCahn
  variable = phi
  mob_name = M
  f_name = fbulk
  args = T
[../]
[./T_dot]
  type = TimeDerivative
  variable = T
[../]
[./CoefDiffusion]
```

```
    type = Diffusion
    variable = T
  [../]
  [./phi_dot_T]
    type = CoefCoupledTimeDerivative
    variable = T
    v = phi
    coef = -1.8
  [../]
[]

[Materials]
  [./free_energy]
    type = DerivativeParsedMaterial
    f_name = fbulk
    args = 'phi T'
    constant_names = pi
    constant_expressions = 4*atan(1)
    function = 'm:=0.9 * atan(10 * (1 - T)) / pi; 1/4*phi^4 - (1/2 - m/3) * phi^3 + (1/4 - m/2) * phi^2'
    derivative_order = 2
    outputs = exodus
  [../]
  [./material]
```

```
    type = InterfaceOrientationMaterial
    op = phi
    mode_number = 4
  [../]
  [./consts]
    type = GenericConstantMaterial
    prop_names  = 'M'
    prop_values = '3333.333'
  [../]
[]

[Preconditioning]
  [./SMP]
    type = SMP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  scheme = bdf2
  solve_type = PJFNK
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart'
```

```
  petsc_options_value = 'hypre    boomeramg      31'

  nl_abs_tol = 1e-10
  nl_rel_tol = 1e-08
  l_max_its = 30

  end_time = 1

  [./TimeStepper]
    type = IterationAdaptiveDT
    optimal_iterations = 6
    iteration_window = 2
    dt = 0.0005
    growth_factor = 1.1
    cutback_factor = 0.75
  [../]
  [./Adaptivity]
    initial_adaptivity = 3 # Number of times mesh is adapted to initial condition
    refine_fraction = 0.7 # Fraction of high error that will be refined
    coarsen_fraction = 0.1 # Fraction of low error that will coarsened
    max_h_level = 5 # Max number of refinements used, starting from initial mesh (before uniform
refinement)
    weight_names = 'phi T'
```

```
    weight_values = '1 0.5'
  [../]
[]

[Outputs]
  interval = 5
  exodus = true
[]
```

3. Kernels

```
#include "ACInterfaceKobayashi1.h"

registerMooseObject("PhaseFieldApp", ACInterfaceKobayashi1);

InputParameters
ACInterfaceKobayashi1::validParams()
{
  InputParameters params = JvarMapKernelInterface<KernelGrad>::validParams();
  params.addClassDescription("Anisotropic gradient energy Allen-Cahn Kernel Part 1");
  params.addParam<MaterialPropertyName>("mob_name", "L", "The mobility used with the kernel");
  params.addParam<MaterialPropertyName>("eps_name", "eps", "The anisotropic interface parameter");
  params.addParam<MaterialPropertyName>(
```

```
    "deps_name",
    "deps",
    "The derivative of the anisotropic interface parameter with respect to angle");
  params.addParam<MaterialPropertyName>(
    "depsdgrad_op_name",
    "depsdgrad_op",
    "The derivative of the anisotropic interface parameter eps with respect to grad_op");
  params.addParam<MaterialPropertyName>(
    "ddepsdgrad_op_name", "ddepsdgrad_op", "The derivative of deps with respect to grad_op");
  return params;
}

ACInterfaceKobayashi1::ACInterfaceKobayashi1(const InputParameters & parameters)
  : DerivativeMaterialInterface<JvarMapKernelInterface<KernelGrad>>(parameters),
    _L(getMaterialProperty<Real>("mob_name")),
    _dLdop(getMaterialPropertyDerivative<Real>("mob_name", _var.name())),
    _eps(getMaterialProperty<Real>("eps_name")),
    _deps(getMaterialProperty<Real>("deps_name")),
    _depsdgrad_op(getMaterialProperty<RealGradient>("depsdgrad_op_name")),
    _ddepsdgrad_op(getMaterialProperty<RealGradient>("ddepsdgrad_op_name"))
{
  // reserve space for derivatives
  _dLdarg.resize(_n_args);
```

```
  _depsdarg.resize(_n_args);
  _ddepsdarg.resize(_n_args);

  // Iterate over all coupled variables
  for (unsigned int i = 0; i < _n_args; ++i)
  {
    _dLdarg[i] = &getMaterialPropertyDerivative<Real>("mob_name", i);
    _depsdarg[i] = &getMaterialPropertyDerivative<Real>("eps_name", i);
    _ddepsdarg[i] = &getMaterialPropertyDerivative<Real>("deps_name", i);
  }
}

RealGradient
ACInterfaceKobayashi1::precomputeQpResidual()
{
  // Set modified gradient vector
  const RealGradient v(-_grad_u[_qp](1), _grad_u[_qp](0), 0);

  // Define anisotropic interface residual
  return _eps[_qp] * _deps[_qp] * _L[_qp] * v;
}

RealGradient
```

```
ACInterfaceKobayashi1::precomputeQpJacobian()
{
  // Set modified gradient vector
  const RealGradient v(-_grad_u[_qp](1), _grad_u[_qp](0), 0);

  // dvdgrad_op*_grad_phi
  const RealGradient dv(-_grad_phi[_j][_qp](1), _grad_phi[_j][_qp](0), 0);

  // Derivative of epsilon wrt nodal op values
  Real depsdop_i = _depsdgrad_op[_qp] * _grad_phi[_j][_qp];
  Real ddepsdop_i = _ddepsdgrad_op[_qp] * _grad_phi[_j][_qp];
  ;

  // Set the Jacobian
  RealGradient jac1 = _eps[_qp] * _deps[_qp] * dv;
  RealGradient jac2 = _deps[_qp] * depsdop_i * v;
  RealGradient jac3 = _eps[_qp] * ddepsdop_i * v;

  return _L[_qp] * (jac1 + jac2 + jac3);
}

Real
ACInterfaceKobayashi1::computeQpOffDiagJacobian(unsigned int jvar)
```

```
{
  // get the coupled variable jvar is referring to
  const unsigned int cvar = mapJvarToCvar(jvar);

  // Set modified gradient vector
  const RealGradient v(-_grad_u[_qp](1), _grad_u[_qp](0), 0);

  // Set off-diagonal jaocbian terms from mobility dependence
  Real dsum =
      _L[_qp] * (_deps[_qp] * (*_depsdarg[cvar])[_qp] * _phi[_j][_qp] * v * _grad_test[_i][_qp]);
  dsum +=
      _L[_qp] * (_eps[_qp] * (*_ddepsdarg[cvar])[_qp] * _phi[_j][_qp] * v * _grad_test[_i][_qp]);
  dsum += (*_dLdarg[cvar])[_qp] * _phi[_j][_qp] * _eps[_qp] * _deps[_qp] * v *
_grad_test[_i][_qp];

  return dsum;
}
#include "ACInterfaceKobayashi2.h"

registerMooseObject("PhaseFieldApp", ACInterfaceKobayashi2);

InputParameters
ACInterfaceKobayashi2::validParams()
```

```
{
  InputParameters params = JvarMapKernelInterface<KernelGrad>::validParams();
  params.addClassDescription("Anisotropic Gradient energy Allen-Cahn Kernel Part 2");
  params.addParam<MaterialPropertyName>("mob_name", "L", "The mobility used with the kernel");
  params.addParam<MaterialPropertyName>("eps_name", "eps", "The anisotropic parameter");
  params.addParam<MaterialPropertyName>(
      "depsdgrad_op_name",
      "depsdgrad_op",
      "The derivative of the anisotropic interface parameter eps with respect to grad_op");
  return params;
}

ACInterfaceKobayashi2::ACInterfaceKobayashi2(const InputParameters & parameters)
  : DerivativeMaterialInterface<JvarMapKernelInterface<KernelGrad>>(parameters),
    _L(getMaterialProperty<Real>("mob_name")),
    _dLdop(getMaterialPropertyDerivative<Real>("mob_name", _var.name())),
    _eps(getMaterialProperty<Real>("eps_name")),
    _depsdgrad_op(getMaterialProperty<RealGradient>("depsdgrad_op_name")),
    _dLdarg(_n_args),
    _depsdarg(_n_args)
{
  // Iterate over all coupled variables
```

```
  for (unsigned int i = 0; i < _n_args; ++i)
  {
    _dLdarg[i] = &getMaterialPropertyDerivative<Real>("mob_name", i);
    _depsdarg[i] = &getMaterialPropertyDerivative<Real>("eps_name", i);
  }
}

RealGradient
ACInterfaceKobayashi2::precomputeQpResidual()
{
  // Set interfacial part of residual
  return _eps[_qp] * _eps[_qp] * _L[_qp] * _grad_u[_qp];
}

RealGradient
ACInterfaceKobayashi2::precomputeQpJacobian()
{
  // Calculate depsdop_i
  Real depsdop_i = _depsdgrad_op[_qp] * _grad_phi[_j][_qp];

  // Set Jacobian using product rule
  return _L[_qp] *
         (_eps[_qp] * _eps[_qp] * _grad_phi[_j][_qp] + 2.0 * _eps[_qp] * depsdop_i * _grad_u[_qp]);
```

```
}

Real
ACInterfaceKobayashi2::computeQpOffDiagJacobian(unsigned int jvar)
{
  // get the coupled variable jvar is referring to
  const unsigned int cvar = mapJvarToCvar(jvar);

  // Set off-diagonal jaocbian terms from mobility and epsilon dependence
  Real dsum = _L[_qp] * 2.0 * _eps[_qp] * (*_depsdarg[cvar])[_qp] * _phi[_j][_qp] * _grad_u[_qp]
*
              _grad_test[_i][_qp];
  dsum += _eps[_qp] * _eps[_qp] * (*_dLdarg[cvar])[_qp] * _phi[_j][_qp] * _grad_u[_qp] *
          _grad_test[_i][_qp];

  return dsum;
}
#include "CoefCoupledTimeDerivative.h"

registerMooseObject("PhaseFieldApp", CoefCoupledTimeDerivative);

InputParameters
CoefCoupledTimeDerivative::validParams()
```

```
{
  InputParameters params = CoupledTimeDerivative::validParams();
  params.addClassDescription("Scaled time derivative Kernel that acts on a coupled variable");
  params.addRequiredParam<Real>("coef", "Coefficient");
  return params;
}

CoefCoupledTimeDerivative::CoefCoupledTimeDerivative(const InputParameters & parameters)
  : CoupledTimeDerivative(parameters), _coef(getParam<Real>("coef"))
{
}

Real
CoefCoupledTimeDerivative::computeQpResidual()
{
  return CoupledTimeDerivative::computeQpResidual() * _coef;
}

Real
CoefCoupledTimeDerivative::computeQpOffDiagJacobian(unsigned int jvar)
{
  return CoupledTimeDerivative::computeQpOffDiagJacobian(jvar) * _coef;
}
```

```
#include "InterfaceOrientationMaterial.h"
#include "MooseMesh.h"
#include "MathUtils.h"

registerMooseObject("PhaseFieldApp", InterfaceOrientationMaterial);

InputParameters
InterfaceOrientationMaterial::validParams()
{
  InputParameters params = Material::validParams();
  params.addParam<Real>(
      "anisotropy_strength", 0.04, "Strength of the anisotropy (typically < 0.05)");
  params.addParam<unsigned int>("mode_number", 6, "Mode number for anisotropy");
  params.addParam<Real>(
      "reference_angle", 90, "Reference angle for defining anisotropy in degrees");
  params.addParam<Real>("eps_bar", 0.01, "Average value of the interface parameter epsilon");
  params.addRequiredCoupledVar("op", "Order parameter defining the solid phase");
  return params;
}

InterfaceOrientationMaterial::InterfaceOrientationMaterial(const InputParameters & parameters)
  : Material(parameters),
    _delta(getParam<Real>("anisotropy_strength")),
```

```
    _j(getParam<unsigned int>("mode_number")),
    _theta0(getParam<Real>("reference_angle")),
    _eps_bar(getParam<Real>("eps_bar")),
    _eps(declareProperty<Real>("eps")),
    _deps(declareProperty<Real>("deps")),
    _depsdgrad_op(declareProperty<RealGradient>("depsdgrad_op")),
    _ddepsdgrad_op(declareProperty<RealGradient>("ddepsdgrad_op")),
    _op(coupledValue("op")),
    _grad_op(coupledGradient("op"))
{
  // this currently only works in 2D simulations
  if (_mesh.dimension() != 2)
    mooseError("InterfaceOrientationMaterial requires a two-dimensional mesh.");
}

void
InterfaceOrientationMaterial::computeQpProperties()
{
  const Real tol = 1e-9;
  const Real cutoff = 1.0 - tol;

  // cosine of the gradient orientation angle
  Real n = 0.0;
```

```
const Real nsq = _grad_op[_qp].norm_sq();
if (nsq > tol)
  n = _grad_op[_qp](0) / std::sqrt(nsq);
if (n > cutoff)
  n = cutoff;

if (n < -cutoff)
  n = -cutoff;

const Real angle = std::acos(n) * MathUtils::sign(_grad_op[_qp](1));

// Compute derivative of angle wrt n
const Real dangledn = -MathUtils::sign(_grad_op[_qp](1)) / std::sqrt(1.0 - n * n);

// Compute derivative of n with respect to grad_op
RealGradient dndgrad_op;
if (nsq > tol)
{
  dndgrad_op(0) = _grad_op[_qp](1) * _grad_op[_qp](1);
  dndgrad_op(1) = -_grad_op[_qp](0) * _grad_op[_qp](1);
  dndgrad_op /= (_grad_op[_qp].norm_sq() * _grad_op[_qp].norm());
}
```

```
 // Calculate interfacial parameter epsilon and its derivatives
 _eps[_qp] = _eps_bar * (_delta * std::cos(_j * (angle - _theta0 * libMesh::pi / 180.0)) + 1.0);
 _deps[_qp] = -_eps_bar * _delta * _j * std::sin(_j * (angle - _theta0 * libMesh::pi / 180.0));
 Real d2eps =
     -_eps_bar * _delta * _j * _j * std::cos(_j * (angle - _theta0 * libMesh::pi / 180.0));

 // Compute derivatives of epsilon and its derivative wrt grad_op
 _depsdgrad_op[_qp] = _deps[_qp] * dangledn * dndgrad_op;
 _ddepsdgrad_op[_qp] = d2eps * dangledn * dndgrad_op;
}
```

APPENDIX C. CODE FOR DIRECTIONAL ALLOY SOLIDIFICATION

The code for directional alloy solidification has 4 parts: equation.cc, ICs_and_BCs.cc, customPDE.h and parameters. In

1. equation.cc

```
void variableAttributeLoader::loadVariableAttributes(){
    // Variable 0
    set_variable_name                (0,"U");
    set_variable_type                (0,SCALAR);
    set_variable_equation_type       (0,EXPLICIT_TIME_DEPENDENT);

  set_dependencies_value_term_RHS(0, "U,mu,phi,grad(phi)");
  set_dependencies_gradient_term_RHS(0, "grad(U),grad(phi),phi");

  // Variable 1
    set_variable_name                (1,"phi");
    set_variable_type                (1,SCALAR);
    set_variable_equation_type       (1,EXPLICIT_TIME_DEPENDENT);

  set_dependencies_value_term_RHS(1, "phi,U,mu");
  set_dependencies_gradient_term_RHS(1, "");

    // Variable 2
```

```
        set_variable_name                       (2,"mu");
        set_variable_type                       (2,SCALAR);
        set_variable_equation_type      (2,AUXILIARY);

    set_dependencies_value_term_RHS(2, "phi,U,grad(phi)");
    set_dependencies_gradient_term_RHS(2, "grad(phi)");
}

template <int dim, int degree>
void
customPDE<dim,degree>::explicitEquationRHS(variableContainer<dim,degree,dealii::VectorizedArray<double> > & variable_list,
                                           dealii::Point<dim, dealii::VectorizedArray<double> > q_point_loc)
const {

// --- Getting the values and derivatives of the model variables ---

// The dimensionless solute supersaturation and its derivatives
scalarvalueType U = variable_list.get_scalar_value(0);
scalargradType Ux = variable_list.get_scalar_gradient(0);

// The order parameter and its derivatives
scalarvalueType phi = variable_list.get_scalar_value(1);
```

```
scalargradType phix = variable_list.get_scalar_gradient(1);

// The auxiliary parameter and its derivatives
scalarvalueType mu = variable_list.get_scalar_value(2);

// --- Setting the expressions for the terms in the governing equations ---

// The azimuthal angle(checked)
scalarvalueType theta;
for (unsigned i=0; i< phi.n_array_elements;i++){
        theta[i] = std::atan2(phix[0][i],phix[1][i]);
}


// Anisotropic term(checked)
scalarvalueType a_n;
a_n = (constV(1.0)+constV(epsilon)*std::cos(constV(4.0)*(theta-constV(theta_0))));


//coefficient before phi
scalarvalueType t_n = constV(userInputs.dtValue*this->currentIncrement);
scalarvalueType y = q_point_loc[1];
scalarvalueType coef_phi = (constV(1.0)-constV(1.0-k)*(y-Vp*t_n)/l_t);
```

```
// coeffcient before U
scalarvalueType coef_U = (constV(1.0+k)-constV(1.0-k)*phi);

// q(phi) term
scalarvalueType q_phi = ((constV(1.0)-phi)+constV(k)*(constV(1.0)+phi)*constV(Ds/Dl));

// grad_phi and grad_U dot product term
scalarvalueType prod_term = (constV(D*(1.0-k))*(phix*Ux)/coef_U/coef_U);
// coef_j
vectorvalueType coef_j =constV(1.0-k)*phix/coef_U/coef_U;

// Antitrapping term
scalargradType j_at;
//j_at[0] = constV(-1.0)/constV(sqrt(2.0))*constV(W0)*(constV(1.0)+(constV(1.0-k))*U)*(mu/a_n/a_n/coef_phi)*(std::cos(theta));
//j_at[1] = constV(-1.0)/constV(sqrt(2.0))*constV(W0)*(constV(1.0)+(constV(1.0-k))*U)*(mu/a_n/a_n/coef_phi)*(std::sin(theta));

// Define required equations
```

```
scalarvalueType eq_U = (U+constV(userInputs.dtValue)*(constV(1.0)+constV(1.0-k)*U)*mu/a_n/a_n/coef_phi/coef_U-constV(userInputs.dtValue)*q_phi*prod_term);

//

scalargradType eqx_U =  (constV(-1.0)*constV(userInputs.dtValue)*D*Ux*q_phi-j_at/coef_U);

//

scalarvalueType eq_phi = (phi+constV(userInputs.dtValue)*mu/a_n/a_n/coef_phi);



// --- Submitting the terms for the governing equations ---

// Terms for the equation to evolve the concentration

variable_list.set_scalar_value_term_RHS(0,eq_U);

variable_list.set_scalar_gradient_term_RHS(0,eqx_U);

// Terms for the equation to evolve the order parameter

variable_list.set_scalar_value_term_RHS(1,eq_phi);

}

//
=====================================================================

// nonExplicitEquationRHS (needed only if one or more equation is time independent or auxiliary)

/=====================================================================
```

```
template <int dim, int degree>
void
customPDE<dim,degree>::nonExplicitEquationRHS(variableContainer<dim,degree,dealii::VectorizedArray<double> > & variable_list,
                                              dealii::Point<dim, dealii::VectorizedArray<double> > q_point_loc)
const {
 // --- Getting the values and derivatives of the model variables ---
// The temperature and its derivatives
 scalarvalueType U = variable_list.get_scalar_value(0);
// The order parameter and its derivatives
 scalarvalueType phi = variable_list.get_scalar_value(1);
 scalargradType phix = variable_list.get_scalar_gradient(1);
// --- Setting the expressions for the terms in the governing equations ---

 // The azimuthal angle
 scalarvalueType theta;
 for (unsigned i=0; i< phi.n_array_elements;i++){
        theta[i] = std::atan2(phix[0][i],phix[1][i]);
 }
// Anisotropic term
 scalarvalueType a_n;
 a_n = (constV(1.0)+constV(epsilon)*std::cos(constV(4.0)*(theta-constV(theta_0))));
```

```
//gradient energy coefficient, its derivative and square
 scalarvalueType a_d = constV(-4.0)*constV(epsilon)*std::sin(constV(4.0)*(theta-constV(theta_0)));
// The anisotropy term that enters in to the  equation for mu
 scalargradType aniso;
 aniso[0] = a_n*a_n*phix[0]-a_n*a_d*phix[1];
 aniso[1] = a_n*a_n*phix[1]+a_n*a_d*phix[0];
 // Define the terms in the equations
scalarvalueType t = constV(userInputs.dtValue*this->currentIncrement);
scalarvalueType y = q_point_loc[1];
scalarvalueType eq_mu =
((phi-constV(lamda))*(U+(y-constV(Vp)*t)/l_t)*(constV(1.0)-phi*phi)*(constV(1.0)-phi*phi));
 scalargradType eqx_mu = (-aniso);
  // --- Submitting the terms for the governing equations ---
 variable_list.set_scalar_value_term_RHS(2,eq_mu);
 variable_list.set_scalar_gradient_term_RHS(2,eqx_mu);
}
//
========================================================================
// equationLHS (needed only if at least one equation is time independent)
//
========================================================================
```

```
template <int dim, int degree>

void
customPDE<dim,degree>::equationLHS(variableContainer<dim,degree,dealii::VectorizedArray
<double> > & variable_list,

                dealii::Point<dim, dealii::VectorizedArray<double> > q_point_loc) const {

}
```

2. ICs_and_BCs.cc

```
template <int dim, int degree>

void customPDE<dim,degree>::setInitialCondition(const dealii::Point<dim> &p, const unsigned
int index, double & scalar_IC, dealii::Vector<double> & vector_IC){


   double center[3][3] = {{0.18,0,0},{0.63,0,0},{0.95,0,0}};

   double rad[3] = {1.0,0.8,0.9};

        double dist;

        scalar_IC = 0;


        // Initial condition for the concentration field

        if (index == 0){

      scalar_IC = -0.55;

        }

        // Initial condition for the order parameter field

        else if (index == 1) {

                // Initial condition for the order parameter field
```

```
                for (unsigned int i=0; i<3; i++){
                        dist = 0.0;
                        for (unsigned int dir = 0; dir < dim; dir++){
                                dist += (p[dir]-center[i][dir]*userInputs.domain_size[dir])*(p[dir]-center[i][dir]*userInputs.domain_size[dir]);
                        }
                        dist = std::sqrt(dist);
            scalar_IC +=    0.5*(1.0-std::tanh((dist-rad[i])/1.414));
                }
        scalar_IC = 2*scalar_IC-1;
            }
}



template <int dim, int degree>
void customPDE<dim,degree>::setNonUniformDirichletBCs(const dealii::Point<dim> &p, const unsigned int index, const unsigned int direction, const double time, double & scalar_BC, dealii::Vector<double> & vector_BC)
{
```

3. CustomPDE.h

```
#include "../../include/matrixFreePDE.h"
```

```
template <int dim, int degree>
class customPDE: public MatrixFreePDE<dim,degree>
{
public:
  // Constructor
  customPDE(userInputParameters<dim> _userInputs): MatrixFreePDE<dim,degree>(_userInputs) , userInputs(_userInputs) {};

  // Function to set the initial conditions (in ICs_and_BCs.h)
  void setInitialCondition(const dealii::Point<dim> &p, const unsigned int index, double & scalar_IC, dealii::Vector<double> & vector_IC);

  // Function to set the non-uniform Dirichlet boundary conditions (in ICs_and_BCs.h)
  void setNonUniformDirichletBCs(const dealii::Point<dim> &p, const unsigned int index, const unsigned int direction, const double time, double & scalar_BC, dealii::Vector<double> & vector_BC);

private:
    #include "../../include/typeDefs.h"

    const userInputParameters<dim> userInputs;

    // Function to set the RHS of the governing equations for explicit time dependent equations (in equations.h)
```

```
void explicitEquationRHS(variableContainer<dim,degree,dealii::VectorizedArray<double> >
& variable_list,
                         dealii::Point<dim, dealii::VectorizedArray<double> >
q_point_loc) const;

// Function to set the RHS of the governing equations for all other equations (in equations.h)
void
nonExplicitEquationRHS(variableContainer<dim,degree,dealii::VectorizedArray<double> > &
variable_list,
                         dealii::Point<dim, dealii::VectorizedArray<double> >
q_point_loc) const;

    // Function to set the LHS of the governing equations (in equations.h)
    void equationLHS(variableContainer<dim,degree,dealii::VectorizedArray<double> > &
variable_list,
                         dealii::Point<dim, dealii::VectorizedArray<double> >
q_point_loc) const;

    // Function to set postprocessing expressions (in postprocess.h)
    #ifdef POSTPROCESS_FILE_EXISTS
    void                                                    postProcessedFields(const
variableContainer<dim,degree,dealii::VectorizedArray<double> > & variable_list,

    variableContainer<dim,degree,dealii::VectorizedArray<double> > & pp_variable_list,
```

```
                                        const dealii::Point<dim, dealii::VectorizedArray<double> >
q_point_loc) const;
        #endif


        // Function to set the nucleation probability (in nucleation.h)
        #ifdef NUCLEATION_FILE_EXISTS
        double getNucleationProbability(variableValueContainer variable_value, double dV)
const;
        #endif


        //
=================================================================
        // Methods specific to this subclass
        //
=================================================================



        //
=================================================================
        // Model constants specific to this subclass
        //
=================================================================


// Matrial Properties constant
```

```
    //double ml = userInputs.get_model_constant_double("ml");
    double c0 = userInputs.get_model_constant_double("c0");
    double Dl = userInputs.get_model_constant_double("Dl");
    double Ds = userInputs.get_model_constant_double("Ds");
    //double gamma = userInputs.get_model_constant_double("gamma");
    //double d0 = userInputs.get_model_constant_double("d0");



//Dimensionless parameters

        //double Tl = userInputs.get_model_constant_double("Tl");
        //double G = userInputs.get_model_constant_double("G");
        //double R = userInputs.get_model_constant_double("R");
        //double Vp = userInputs.get_model_constant_double("Vp");

        //double dt = userInputs.get_model_constant_double("dt");
        //double dx = userInputs.get_model_constant_double("dx");

// New input
        double W0 = userInputs.get_model_constant_double("W0");
    double tau0 = userInputs.get_model_constant_double("tau0");
    double epsilon = userInputs.get_model_constant_double("epsilon");
    double k = userInputs.get_model_constant_double("k");
```

```
    double lamda = userInputs.get_model_constant_double("lamda");
    double D = userInputs.get_model_constant_double("D");
    double Vp = userInputs.get_model_constant_double("Vp");
    double l_t = userInputs.get_model_constant_double("l_t");
    double theta_0 = userInputs.get_model_constant_double("theta_0");


        //
==========================================================


};
```

4. Parameters.in

```
#
============================================================================
============

# Set the number of dimensions (2 or 3 for a 2D or 3D calculation)

#
============================================================================
============

set Number of dimensions = 2


#
============================================================================
============

# Set the length of the domain in all three dimensions
```

```
# (Domain size Z ignored in 2D)
#
============================================================================
============
# Each axes spans from zero to the specified length
set Domain size X = 500
set Domain size Y = 500
set Domain size Z = 500


#
============================================================================
============
# Set the element parameters
#
============================================================================
============
# The number of elements in each direction is 2^(refineFactor) * subdivisions
# Subdivisions Z ignored in 2D
# For optimal performance, use refineFactor primarily to determine the element size
set Subdivisions X = 3
set Subdivisions Y = 3
set Subdivisions Z = 3


set Refine factor = 6
```

```
# Set the polynomial degree of the element (allowed values: 1, 2, or 3)
set Element degree = 3

#
============================================================================
============

# Set the adaptive mesh refinement parameters
#
============================================================================
============

# Set the flag determining if adaptive meshing is activated
set Mesh adaptivity = true

# Set the maximum and minimum level of refinement
# When adaptive meshing is enabled, the refine factor set in the block above is
# only used to generate the first pass of the mesh as the initial conditions are
# applied. It should be set somewhere between the max and min levels below.
set Max refinement level = 6
set Min refinement level = 0
# Set the number of time steps between remeshing operations
set Steps between remeshing operations = 250

# Set the criteria for adapting the mesh
subsection Refinement criterion: phi
```

```
    # Select whether the mesh is refined based on the variable value (VALUE),
    # its gradient (GRADIENT), or both (VALUE_AND_GRADIENT)
    set Criterion type = VALUE
    # Set the lower and upper bounds for the value-based refinement window
    set Value lower bound = -0.9999
    set Value upper bound = 0.9999
end


#
============================================================================
============
# Set the time step parameters
#
============================================================================
============
# The size of the time step
set Time step = 0.01


# The simulation ends when either the number of time steps is reached or the
# simulation time is reached.
set Number of time steps = 500000
```

```
#
#============================================================================
#============

# Set the boundary conditions

#
#============================================================================
#============

# Set the boundary condition for each variable, where each variable is given by

# its name, as defined in equations.h. The four boundary condition

# types are NATURAL, DIRICHLET, NON_UNIFORM_DIRICHLET and PERIODIC. If all

# of the boundaries have the same boundary condition, only one boundary condition

# type needs to be given. If multiple boundary condition types are needed, give a

# comma-separated list of the types. The order is the minimum of x, maximum of x,

# minimum of y, maximum of y, minimum of z, maximum of z (i.e left, right, bottom,

# top in 2D and left, right, bottom, top, front, back in 3D). The value of a

# Dirichlet BC is specified in the following way -- DIRCHILET: val -- where 'val' is

# the desired value. If the boundary condition is NON_UNIFORM_DIRICHLET, the

# boundary condition should be specified in the appropriate function in 'ICs_and_BCs.h'.

# Example 1: All periodic BCs for variable 'c'

# set Boundary condition for variable c = PERIODIC

# Example 2: Zero-derivative BCs on the left and right, Dirichlet BCs with value

# 1.5 on the top and bottom for variable 'n' in 2D

# set Boundary condition for variable n = NATURAL, NATURAL, DIRICHLET: 1.5, DIRICHLET: 1.5
```

PERIODIC, NATURAL, PERIODIC, NATURAL

set Boundary condition for variable U = NATURAL

set Boundary condition for variable phi = NATURAL

set Boundary condition for variable mu = NATURAL

#
==========

Set the model constants

#
==========

Set the user-defined model constants, which must have a counter-part given in

customPDE.h. These are most often used in the residual equations in equations.h,

but may also be used for initial conditions and nucleation calculations. The type

options currently are DOUBLE, INT, BOOL, TENSOR, and [symmetry] ELASTIC CONSTANTS

where [symmetry] is ISOTROPIC, TRANSVERSE, ORTHOTROPIC, or ANISOTROPIC.

Solute diffusion coefficient in solids

set Model constant Ds = 8.92e-12, DOUBLE

Solute diffusion coefficient in liquid

set Model constant Dl = 5.66e-9, DOUBLE

```
# Anisotropy strength

set Model constant epsilon = 0.02, DOUBLE


# Solubility partition coefficient

set Model constant k = 0.19, DOUBLE


# initial bulk concentration

set Model constant c0 = 0.13, DOUBLE


# Coupling constant

set Model constant lamda = 35.356 , DOUBLE


# dimensionless length unit

set Model constant W0 = 1.0, DOUBLE


# dimensionless time unit

set Model constant tau0 = 1.0, DOUBLE


# dimensionless diffusion

set Model constant D = 22.16, DOUBLE


# dimensionless pulling speed

set Model constant Vp = 0.1252, DOUBLE
```

dimensionless thermal length

set Model constant l_t = 847.266, DOUBLE

initial angle

set Model constant theta_0 = 2.36, DOUBLE

#
==========

Set the output parameters

#
==========

Type of spacing between outputs ("EQUAL_SPACING", "LOG_SPACING", "N_PER_DECADE",

or "LIST")

set Output condition = EQUAL_SPACING

Number of times the program outputs the fields (total number for "EQUAL_SPACING"

and "LOG_SPACING", number per decade for "N_PER_DECADE", ignored for "LIST")

set Number of outputs = 100

The number of time steps between updates being printed to the screen

set Skip print steps = 1000

#
==========
==========

Set the checkpoint/restart parameters

#
==========
==========

Whether to start this simulation from the checkpoint of a previous simulation

set Load from a checkpoint = false

Type of spacing between checkpoints ("EQUAL_SPACING", "LOG_SPACING", "N_PER_DECADE",

or "LIST")

set Checkpoint condition = EQUAL_SPACING

Number of times the creates checkpoints (total number for "EQUAL_SPACING"

and "LOG_SPACING", number per decade for "N_PER_DECADE", ignored for "LIST")

set Number of checkpoints = 2

REFERENCES

[1] J. P. Birat et al., The Making, Shaping and Treating of Steel: Casting Volume, 11th ed. Warrendale: Association for Iron & Steel Technology, 2010.

[2] JFE 21st Century Foundation, retrieved from http://www.jfe-21st-cf.or.jp/chapter_2/2j_2.html

[3] B. G. Thomas, "Intro to Continuous Casting - CCC - U of I," Continuous Casting Consortium. [Online]. Available: http://ccc.illinois.edu/introduction/basicphenom.html. [Accessed: 16-Feb-2018].

[4] Kobayashi R 1993 Modelling and numerical simulations of dendritic crystal growth Physica D 63 410-23

[5] Kobayashi R 1994 A numerical approach to three-dimensional dendritic solidification Exp. Math. 359–81

[6] Karma A 2001 Phase-field formulation for quantitative modeling of alloy solidification Phys. Rev. Lett. 87 115701

[7] Ingo Steinbach, Ingo Steinbach, "Phase-field models in materials science" Published 30 July 2009.Online at stacks.iop.org/MSMSE/17/073001.

[8] J. Rowlinson, Translation of J.D. van der Waals' "The thermodynamic theory of capillarity under the hypothesis of a continuous variation of density", J. Statist. Phys. 20 (2) (1979) 197–245.

[9] V. Ginzburg, L. Landau, On the theory of superconductivity, Zh. Eksp. Teor. Fiz. 20 (1950) 1064–1082. Translation in Collected papers of L.D. Landau, Pergamon, Oxford, 1965, pp. 546–568.

[10] J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. I.Interfacial free energy, J. Chem. Phys. 28 (1958) 258–267. [9] Kobayashi R 1993 Modelling and numerical simulations of dendritic crystal growth Physica D 63 410-23.

[11] Wang S-L, Sekerka R F, Wheeler A A, Murray B T, Coriell S R, Rraun R J and McFadden G B 1993 Thermodynamically-consistent phase-field models for solidification Physica D 69 189–200.

[12] Blas Echebarria, Quantitative phase-field model of alloy solidification PHYSICAL REVIEW E 70, 061604 (2004).

[13] Tomohiro Takaki, Phase-field Modeling and Simulations of Dendrite Growth, ISIJ International, Vol. 54 (2014), No. 2, pp. 437–444.

[14] H. Yin, S.D. Felicelli, Dendrite growth simulation during solidification in the LENS process, Acta Materialia 58 (2010) 1455–1465.

[15] Chang-sheng Zhu, Sheng Xu, Li Feng, Dan Han, Kai-ming Wang, Phase-field model simulations of alloy directional solidification and seaweed-like microstructure evolution based on adaptive finite element method, Computational Materials Science 160 (2019) 53–61.

[16] Nele Moelans, Bart Blanpain, Patrick Wollants, An introduction to phase-field modeling of microstructure evolution, Computer Coupling of Phase Diagrams and Thermochemistry 32 (2008) 268–294.

[17] Hsu YR, Lin MC, Lin HK, Chang YH, Lu CC, et al. (2018) Numerical simulation of nanopost-guided self-organization dendritic architectures using phase-field model. PLOS ONE 13(7)

[18] Won Y M, Thomas B G. Simple model of microsegregation during solidification of steel. Metallurgical and Material Transactions A, 2001, 32(7): 1755-1767

[19] El-Bealy M, Thomas B G. Prediction of dendrite arm spacing for low alloy steel casting processes[J]. Metallurgical and materials transactions B, 1996, 27(4): 689-693.

[20] Pierer R, Bernhard C. On the influence of carbon on secondary dendrite arm spacing in steel[J]. Journal of materials science, 2008, 43(21): 6938-6943.

[21] Weisgerber B, Hecht M, Harste K. Investigations of the solidification structure of continuously cast slabs[J]. Steel research, 1999, 70(10): 403-411.

[22] Young K P, Kerkwood D H. The dendrite arm spacings of aluminum-copper alloys solidified under steady-state conditions[J]. Metallurgical Transactions A, 1975, 6(1): 197-205.

[23] Ganguly S, Choudhary S K. Quantification of the Solidification Microstructure in Continuously-Cast High Carbon Steel Billets[J]. Metallurgical and Materials Transactions B, 2009, 40(3): 397-404.

[24] H. Xing, X. Dong, H. Wu, et al., Degenerate seaweed to tilted dendrite transition and their growth dynamics in directional solidification of non-axially oriented crystals: a phase-field study, Sci Rep. 6 (2016) 26625.

[25] N. Provatas, Q. Wang, M. Haataja, et al., Seaweed to dendrite transition in directionalsolidification, Phys. Rev. Lett. 91 (2003).

[26] N. Provatas, N. Goldenfeld, J. Dantzig, Adaptive mesh refinement computation of solidification microstructures using dynamic data structures, J. Comput. Phys. 148 (1999) 265–290.

[27] Ruo Li, On multi-mesh H-adaptive methods, J. Sci. Comput. 24 (2005) 321–341.

[28] X. Hu, R. Li, T. Tang, A Multi-mesh adaptive finite element approximation to phase field models, Comm. Comput. Phys. 5 (2009) 1012–1029.

[29] C.S. Zhu, P. Lei, R.Z. Xiao, et al., Phase-field modeling of dendritic growth under forced flow based on adaptive finite element method, Trans. Nonferr. Metals Soc. China 25(2015) 241–248.

[30] Seppo Louhenkilpi, Continuous Casting of Steel, Treatise on Process Metallurgy: Industrial Processes, 2014