TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

AUSTRIAN INSTITUTE OF TECHNOLOGY

Diploma Thesis

Grain growth simulation in aluminium casting processes

under the supervision of

Stephan Jäger, MSc
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Christoph Eisenmenger-Sittner

submitted in partial fulfillment of the requirements for the degree of Dipl.-Ing.
by

Bernhard Pruckner, BSc

5th August, 2020

Abstract

In this work the implementation of a grain growth simulation for an aluminum-copper alloy will be presented. Two seperate single-physics solvers will be coupled; a Cellular Automaton (CA) to simulate grain growth on a mesoscopic scale and a macroscopic solver, that simulates the solidification process using a finite-volume (FV) method and provides the temperature distribution. These solvers will be coupled together to a CAFV simulation, that provides the macroscopically calculated temperature distribution and the mesoscopic grain distribution for solidification processes of a melt. The time dependent temperature distribution given by the FV solver will be used by a CA algorithm to determine the grain growth. The data on the numerical grid of one solver, which is stored as a cloud of points, is interpolated and mapped onto the other coupled solver. Thus one solver has access to the output values of the other solver. Interpolation and mapping will be handled by a coupling library, which allows for a quick and easy way to establish a multi-physics simulation by coupling several single-physics solvers.

In dieser Arbeit wird die Implementierung einer Kornwachstumssimulation für eine Aluminium-Kupfer-Legierung vorgestellt. Zwei getrennte Einzelphysik-Solver werden gekoppelt; ein Zellularer Automat (CA) zur Simulation des Kornwachstums auf einer mesoskopischen Skala und ein makroskopischer Solver, der den Erstarrungsprozess mit einer Finite-Volumen (FV)-Methode simuliert und die Temperaturverteilung liefert. Diese Solver werden zu einer CAFV-Simulation gekoppelt, die die makroskopisch berechnete Temperaturverteilung und die mesoskopische Kornverteilung für Erstarrungsprozesse einer Schmelze liefert. Die durch den FV-Solver gegebene zeitabhängige Temperaturverteilung wird von einem CA-Algorithmus zur Bestimmung des Kornwachstums verwendet. Die Daten auf dem numerischen Gitter des einen Solvers, das als Punktwolke gespeichert wird, werden interpoliert und auf den anderen gekoppelten Solver abgebildet. Somit hat ein Solver Zugriff auf die Ausgabewerte des anderen Solvers. Interpolation und Mapping werden durch eine Kopplungsbibliothek gehandhabt, die einen schnellen und einfachen Weg zur Erstellung einer Multiphysik-Simulation durch Kopplung mehrerer Einzelphysik-Solver ermöglicht.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The knowledge of the microstructure of metal alloys is valuable for several reasons. It allows for specifying the mechanical properties of the solid, since the grain size effects the yield stress of a material [1]. The simulation of grain growth during solidification has become an important tool to provide the size and shape of grains in solidified metal alloys apart from experiments. It gives the possibility to investigate grain growth processes without depending on possibly complex and costly experiments. Further it provides essential information for forming processes simulations and computing the recrystallization. The grain growth simulation developed in this work will be a part of a bigger process simulation. The process includes the casting of molten metal alloys, the solidification and grain formation and the forming of the solid material. To conduct forming simulations, the grain structure of the solidified metal alloy is required.
The size and distribution of grains depend on the cooling rate, the temperature gradient during the solidification and thermophysical properties of the alloy, e.g. latent heat release during phase shift and the specific heat capacity. Therefore, the heat transport and the release of latent heat need to be computed numerically.
In order to establish a grain growth simulation, a macroscopic solver that provides the temperature distribution needs to be coupled with the mesoscopic grain growth algorithm. For the macroscopic part of the simulation, OpenFOAM [2], an open source Finite Volume (FV) code for fluid dynamics, will be used. The grain growth will be solved by a Cellular Automaton (CA) algorithm. These two separate single physics solver will be coupled by the coupling library preCICE [3].
The main objective of this thesis is to implement a coupled CAFV simulation that yields the temperature distribution and grain formation in a solidified aluminum-copper alloy.

1.2 State of the art

For simulating the nucleation and grain growth in solidification processes there are different models available, depending on which scale the process needs to be analysed. Models on a macroscopic scale yield the amount of solid and liquid fraction of a material in a specified domain, but fail to resolve individual grains on the scale of grain dendrites [4]. The mesoscopic models, that will be discussed in this work, describe grain growth at the length scales of the primary dendritic arms of grains, though they will not be resolved. Thus, the growth of individual grains is directly modelled. Dendritic growth is modelled by the means of CA algorithms, that resolve the primary dendrite arms. Here a growth velocity is assigned to the tip of each primary dendrite arm of a grain. Such a CA was first introduced by Rappaz and Gandin [5]. In [6] the CA algorithm is compared to analytically calculated grain growth. The growth rate of a specific dendrite arm of the grain depends on the local temperature gradient at that point. The arms in the direction of descending temperature will have a continuously rising growth velocity until their growth is stopped by reaching the boundary of the domain or another grain. In the other direction, the arms will grow until they reach the liquidus temperature isotherm. Thus the grain will preferably grow into the direction of the negative temperature gradient, which could be reproduced by the CA algorithm. The original algorithm was extended by a decentered grain algorithm [5], that shifts the center of grain growth to newly solidified cells of the CA and thus yields more realistic grain shapes. This algorithm was extended to three dimensions by Gandin and Rappaz [7]. In order to get the correct shape, the growth front has to be resolved at the typical spacing of secondary dendrite arms. The growth rate of the dendrite arms of the grain is obtained by using the KGT-model [8] by Kurz, Giovanola and Trivedi.
Rappaz and Thevoz [9] introduced another model that introduces a mushy zone in addition to liquid and solid states. The mushy zone is modeled around the solid core of the grain, which represents the area of primary and secondary dendrite arms, that are surrounded by interdendritic melt. The shape is modeled as simple spheres, which represent the size of the primary dendrite arms. This approach was further developed by Wu and Ludwig [10]. Here fluid flow and concentration of interdendritic melt has to be considered. Such approaches work with the LGK-model [11] by Lipton, Glicksman and Kurz. The LGK-model describes the growth of a dendrite arm in a steady state with regard to local undercooling and the supersaturation of the melt, which can be calculated from the temperature and the solute concentration.
There are two morphologies of microstructure to be commonly distinguished, columnar and equiaxed grain growth that are associated to heterogeneous and homogeneous solidification respectively. The growth of columnar morphologies can be predicted by the undercooling of the melt. For equiaxed morphologies the grain growth rate is not directly dependent on the speed of the isotherms [4]. Gandin and Rappaz [12] first coupled the CA with a FE code that solves the heat conduction equation. This approach is known as CAFE. The temperature field is stored on a coarse numerical grid. The local undercooling is then interpolated and the CA is performed on a finer resolved grid. Based on

[5, 7, 12, 8] a CA with a simple relation between growth rate and local undercooling was used by Rai, Helmer and Körner [13]. This CA serves as basis for the CA used in this work.

1.3 Structure of the work

Chapter 2 will give an overview of the theoretical background necessary to understand solidification as well as the software tools to simulate such processes. Both, the theory of continuum mechanics and heat transport as well as mesoscopic grain growth will be reviewed. In chapter 3, the numerical methods for implementing the theoretical models in physical simulations will be introduced. The FV method and the used FV solvers in OpenFOAM, that will be used for the macroscopic computations, will be described. The CA algorithm used to simulate grain growth in solidification processes will be introduced. The CA is embedded in palabos, which is a lattice-Boltzmann code for solving the transport equation. However, palabos will only be used as a software framework, since parallelization using MPI is already implemented in the code structure and can be easily applied. The lattice-Boltzmann method will not be used in the simulation. The CA and OpenFOAM will be coupled with the coupling library preCICE. In chapter 4 the setup of the solvers will be discribed. In chapter 5, the results of the solidification simulation will be shown and will be discussed in chapter 6. The focus will lay on the correct coupling of the macroscopic and mesocopic solvers, the efficiency of the coupled simulation and the plausibility of the grain distribution yielded by the CAFV simulation. Chapter 7 gives a short summary of the work. In chapter 8 an outlook to possible future work and improvements will be given.

CHAPTER 2

Theory

In this section the underlying theory of the grain growth simulation will be discussed. The simulation consists of a mesocopic grain growth algorithm that is implemented in a Cellular Automaton (CA) and a macroscopic model for heat and fluid transport using a finite-volume (FV) solver. The CA grain growth algorithm needs the temperature distribution, which is provided by the macroscopic solver.

2.1 Macroscopic - Continuum Mechanics

There are two models, that are used to compute the macroscopic temperature distribution. The first model is based on a simple Laplacian equation for temperature diffusion,

$$\rho \; c_p \frac{\partial}{\partial t} T(\boldsymbol{r}, t) - \nabla \cdot (\lambda \; \nabla T(\boldsymbol{r}, t) \;) = 0 \tag{2.1}$$

with temperature $T(\boldsymbol{r}, t)$, thermal conductivity λ, specific heat capacity at constant pressure c_p and the density of the material ρ. $\boldsymbol{r}$ is the vector of position and t denotes the time.

The second more sophisticated model considers heat transport, the release of latent heat during solidification and convection in the liquid regions. It was proposed by Voller and Prakash [14]. A mushy region is introduced. It models the gradual solidification of a certain volume. The solid fraction $f_s = V_s/V$ and liquid fraction $f_l = V_l/V$, with the volume that is liquid/solid $V_{l/s}$ and the total Volume V, are introduced [4]. It follows: $f_s + f_l = 1$. These quantities give the fraction of the material in a certain volume that is solid or liquid. The whole domain is considered as porous medium. The actual velocity $\boldsymbol{u}$ of the fluid is given by

$$\boldsymbol{u} = f_l \cdot \boldsymbol{u_l} + f_s \cdot \boldsymbol{u_s} \tag{2.2}$$

with the f_l and f_s denoting the liquid and solid fraction and $\boldsymbol{u_l}$ and $\boldsymbol{u_s}$ denoting the velocity of the liquid and the solid phase respectively. In this model the solidified part

of the material is considered to be fixed, therefore $\boldsymbol{u_s} = 0$.
The macroscopic solidification model is described by three conservation laws: mass conservation, conservation of momentum and conservation of enthalpy.

Conservation of mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{u} = 0 \tag{2.3}$$

The density is assumed to be constant over time, $\partial_t \rho = 0$. Therefore, the conservation of mass reduces to $\nabla \cdot \boldsymbol{u} = 0$. For the already solidified parts of the material $\boldsymbol{u_s} = 0$ applies.

Conservation of momentum:

$$\rho \frac{\partial(\boldsymbol{u})}{\partial t} + (\nabla \cdot \rho \boldsymbol{u})\boldsymbol{u} = \nabla \cdot (\mu \, \nabla \boldsymbol{u}) - \nabla \boldsymbol{p} + \boldsymbol{S_p} + \boldsymbol{S_b} \tag{2.4}$$

with the pressure $\boldsymbol{p}$, density ρ, dynamic viscosity μ, and source terms $\boldsymbol{S_p}$ and $\boldsymbol{S_b}$. $\boldsymbol{S_p}$ is introduced to account for the progressing solidification of the fluid and $\boldsymbol{S_b}$ describes natural convection due to buoyancy in a gravitational field. $\boldsymbol{S_p}$ is used to describe the blockage of fluid flow in the mushy regions, where a part of the alloy has already solidified. The mush is considered to be a porous medium and Darcy's law [15] is applied. It describes a fluid running through a porous medium. As solidification proceeds, the porosity decreases until the fluid is completely solid and the velocity gets down to 0. The source term

$$\boldsymbol{S_p} = A\boldsymbol{u} \tag{2.5}$$

is chosen to reflect this behavior. Based on Darcy's law the Carman-Kozeny equation describes the pressure gradient

$$\nabla P = \frac{-C(1 - f_l)^2}{f_l^3 \boldsymbol{u}} \tag{2.6}$$

with a constant C that depends on the morphology of the porous medium. From there a functional form for the factor A in equation 2.5 can be derived. It writes

$$A = \frac{-C(1 - f_l)^2}{f_l^3 + q}. \tag{2.7}$$

It gives a result of 0 for a solid fraction $f_s = 0$ and increases to a large numerical value for a solid fraction of 1. q is a constant with a small value to avoid dividing by 0 in numerical application. For increasing amount of f_s the momentum equation gets dominated by the source term S_p, approaching the behavior of Darcy's law. For complete solidification the velocities approach a very small numerical value close to 0. The source term $\boldsymbol{S_b}$ describes convection due to buoyancy as

$$\boldsymbol{S_b} = \frac{\rho \beta (h - h_{ref})}{c_p} \boldsymbol{g} \tag{2.8}$$

where β is a coefficient for thermal expansion, h_{ref} is a reference value for the sensible heat, ρ is the density of the melt and $\boldsymbol{g}$ is the gravitational acceleration.

Conservation of Enthalpy:
The total enthalpy H, meaning the total heat energy of the system, can be described as

$$H = h + \Delta H \tag{2.9}$$

with the sensible heat $h = c_p T$ and the latent heat released during the phase transition $\Delta H = L f_l$. It gives the amount of latent heat that is released, depending on how much of the fluid already has solidified. L is the total amount of latent heat per volume that can be released during the phase transition. The enthalpy conservation law denotes

$$\frac{\partial h}{\partial t} + \nabla \cdot \boldsymbol{u} h = \nabla \cdot (\alpha \ \nabla h) - S_h \tag{2.10}$$

with the thermal diffusivity $\alpha = \lambda / c_p \rho$, the sensible heat h and the enthalpy source term S_h, which gives the amount of latent heat that is released during the phase transition. It writes

$$S_h = \frac{\partial \Delta H}{\partial t} + \nabla \cdot (\boldsymbol{u} \Delta H). \tag{2.11}$$

2.2 Mesoscopic - Dendritic Grain Growth

In this work, we look at dendritic grain growth of an $Al - Cu$ binary alloy. Dendritic growth describes the growth of tree-like structures in an undercooled liquid. The dendrite grows in primary arms that can branch in secondary arms and further. The growth direction and rate are determined by the temperature gradient of the surrounding undercooled liquid, the cooling rate and the crystallographic direction of the dendrite. The local undercooling of the melt ΔT at the solid-liquid interface consists of three parts, namely

$$\Delta T = \Delta T_t + \Delta T_c + \Delta T_r. \tag{2.12}$$

The terms on the right hand side of equation 2.12 are the thermal undercooling ΔT_t, the solutal undercooling ΔT_c and the shift of the liquidus line due to the finite radius of the dendrite tip ΔT_r.
When the first component of the alloy solidifies the remaining components are rejected into the surrounding melt [4]. The increased concentration of the solute in the vicinity of the solid-liquid interface shifts the liquidus temperature at the interface to higher temperatures. This effect is described by the solutal undercooling ΔT_c. Similarly, the thermal undercooling ΔT_t describes the rejection of solidification heat into the surrounding melt.
The CA grain growth algorithm is based on an algorithm suggested by Gandin and Rappaz [5, 7, 6] which is based on the KGT-model proposed by Kurz, Giovanola and Trivedi [8]. In the KGT model [8] dendritic growth only depends on the local thermal

undercooling. The change in solute concentration around the dendritic tip due to the rejection of solute into the melt is not considered. The rejected solidification heat diffuses much faster in the melt than rejected solute. Therefore, the macroscopic finite element solver, which computes the heat diffusion does not need to have a spatial resolution as high as the solver of the CA managing the grain growth. This chapter deals with the underlying theory. The macroscopic solver and the CA will be discussed in the following chapter.

KGT Model

In the KGT model the growth rate of dendritic grains in the solute is determined from the local thermal undercooling of the melt. This model uses a stability criterion to determine the growth rate of the tips of the main dendritic arms of the grain. In the stability model a planar interface between the solid phase and the undercooled melt is considered. The solid interface advances into the melt. The planar interface is subject to small sinusoidal disturbances with certain wavelengths λ. Using linear perturbation theory, one wants to get the smallest wave number at which the interface is stable, meaning the amplitudes of the disturbances decay and vanish over time. The dendrite tip operates within marginal stability, meaning the tip grows as large as possible without splitting or becoming unstable. The product of growth velocity of the tip and the square of the minimal wavelength $v\lambda_s^2$ is constant for small growth rates. The radius of the tip R is equal to the minimal wavelength of the solid-liquid interface

$$R = \lambda_s. \tag{2.13}$$

The rest of the dendritic interface is unstable and it's growth is time dependent. Sinusoidal disturbances are produced, get amplified over time and build dendritic side arms. The marginally stable wavelength and therefore the tip radius has been derived to be

$$R = 2\pi\sqrt{\frac{\Gamma}{mG_c\xi_c - G_t}} \tag{2.14}$$

here Γ is the Gibbs-Thomson parameter, which is the ratio of solid-liquid interface energy to melting entropy, m is the liquidus slope, G_c is the concentration gradient at the interface in the liquid phase and G_t is the temperature gradient at the interface. ξ_c is the solutal stability function:

$$\xi_c = 1 + \frac{2k}{1 - 2k - \sqrt{1 + \left(\frac{2\pi}{P_c}\right)^2}} \tag{2.15}$$

Here $P_c = vR/2D$ is the solute Péclet number. Here it gives the ratio of growth rate of the solid-liquid interface fluid to the diffusion within the fluid. v is the growth velocity of the interface, D the diffusion coefficient of the solute in liquid. At low growth rates,

ξ_c is close to unity. k is the partition coefficient and is assumed to be the constant. The local thermal undercooling is given by

$$\Delta T(\boldsymbol{r}) = T_l - T(\boldsymbol{r}) \tag{2.16}$$

Here T_l is the liquidus Temperature, and $T(\boldsymbol{r})$ the local temperature at position $\boldsymbol{r}$.
A relation between the growth velocity of the dendrite tips and the local thermal undercooling was derived using the KGT model. Usually it is approximated by a second [6] or third [13] order polynomial function. The second order polynomial approximation is only valid for small undercoolings[6].

$$v = A \cdot \Delta T^2 + B \cdot \Delta T^3 \tag{2.17}$$

The parameters A and B can obtained by phase field simulations [16] or analytically solving the KGT model [8].

CHAPTER 3

Methods

In this chapter, the numerical methods of the coupled grain growth simulation will be discussed. The goal of this thesis is to create a coupled CAFV simulation consisting of a CA and a FV solver. The main objective is to move the calculation of the temperature field away from the mesoscopic scale and carry out this part of the simulation on a macroscopic scale. This way computational time can be reduced. The temperature field is provided by an FV solver, that simulates temperature diffusion on a macroscopic model. The macroscopic FV solver with its coarse mesh is coupled with the mesocopic CA with a much finer resolved grid.

3.1 Macroscopic - Continuum Mechanics

3.1.1 Time Discretization - Finite Difference Method

in OpenFOAM the time integration is managed by the finite difference method (FDM). The temporal domain is discretized into an evenly spaced grid. A function is represented by discrete point values on the grid nodes. FDM will be discussed based on the example of a simple one-dimensional temperature diffusion equation in spatial x-direction

$$\frac{\partial T}{\partial t} = \alpha \cdot \frac{\partial^2 T}{\partial x^2}. \tag{3.1}$$

The time derivative is replaced by the difference quotients of first and second order respectively, using the Taylor expansion of the function. OpenFOAM uses the FDM for the time discretization. The spatial discretization is managed by the finite volume method FVM and will be discussed in the following sections. The time discretization denotes

$$\frac{\partial T}{\partial t} = \frac{T_t - T_{t-1}}{\Delta t}. \tag{3.2}$$

with the time step index t and the spatial and temporal spacing Δx and Δt. Depending at which timestep the spatial derivative is calculated, one will acquire an explicit (forward

Euler) or an implicit (backward Euler) scheme. An explicit scheme can be solved straight forward for each grid point. Equation 3.1 will yield

$$T_{t+1} = T_t + \alpha \cdot \left.\frac{\partial^2 T}{\partial x^2}\right|_t \tag{3.3}$$

which can be solved directly by simply iterating over all grid points i each time step t. Using the implicit scheme equation 3.1 will yield

$$T_{t+1} = T_t + \alpha \cdot \left.\frac{\partial^2 T}{\partial x^2}\right|_{t+1} \tag{3.4}$$

This will lead to a matrix equation, which needs to be solved iteratively.
Implicit schemes have the advantage of being stable at larger time step sizes, compared to explicit schemes. In theory a simple implicit scheme like equation 3.4 is unconditionally stable. The inversion of the matrix in each time step is solved iteratively. If the time step is to big the results are not accurate anymore, especially for convective processes. The time step chosen for the macroscopic OpenFOAM solvers (as discussed in chapter 4.6.4) are chosen to maintain stability and accuracy of the results.

3.1.2 Spatial Discretization - Finite Volume Method

Spatial discretization is managed by FVM. It is an integral method where the numeric domain is split into finite volumes and the fluxes of physical values (eg. temperature, velocity, etc) are stored at the boundary faces of these volumes. In the FVM conservation laws are inherently satisfied, therefore it has become a standard method for computational fluid dynamics [17]. A very brief overview is given in this section. As an example we look at the conservation of mass equation 2.3 from section 2.1.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{u} = 0 \tag{3.5}$$

Each finite volume satisfies the conservation law. The change of the conserved quantity in each volume is defined by the flux through the faces of each volume and the temporal change in the density of the fluid

$$\int_V \frac{\partial}{\partial t} \rho \, dV + \int_{\partial V} \rho \boldsymbol{u} \cdot \boldsymbol{n} \, dA = 0. \tag{3.6}$$

In the case of incompressible fluids, which is assumed in this work, the density is constant and equation 3.6 reduces to

$$\int_{\partial V} \rho \boldsymbol{u} \cdot \boldsymbol{n} \, dA = 0. \tag{3.7}$$

This yields for each finite volume with index i the following equation for the mean value of the conserved quantity, placed at the center of the volumes:

$$\frac{1}{V_i} \int_{\partial V_i} \rho \boldsymbol{u} \cdot \boldsymbol{n} \, dA = 0 \tag{3.8}$$

The surface integral is discretized into the sum of the fluxes over all faces of the finite volume.

$$\frac{1}{V_i}\int_{\partial V_i} \rho \boldsymbol{u}\cdot\boldsymbol{n}\ dA = \sum_{f=0}^{N}(\rho \boldsymbol{u_f}\ \boldsymbol{A_f}) \tag{3.9}$$

The right hand side shows the summation over all N faces of a volume and the surface area vector of a face f, ΔA_f and the velocity vector on that face u_f.

FVM in OpenFOAM

OpenFOAM [2] is a free open source solver for computational fluid dynamics developed by the OpenFOAM Foundation [18]. In this work the version OpenFOAM 5 was used. Spatial discretization is managed by FVM. The solver used in this thesis uses an implicit FVM. There are various discretization schemes for differential operators available. As an example we look at the energy and momentum conservation laws, equation 2.10 and 2.4. There are diffusion, divergence and gradient terms. As an example we first look at a steady-state diffusion equation

$$\nabla\cdot(\alpha\ \nabla\Phi) = -S^{\Phi} \tag{3.10}$$

where Φ is a scalar variable e.g. enthalpy, α is a diffusion coefficient and S^{Φ} a source term for Φ. With defining the diffusion flux according to Fick's law of diffusion [19] $\boldsymbol{J}^{\Phi,D} = -\alpha\nabla\Phi$ the diffusion term can be rewritten as

$$\nabla\cdot\boldsymbol{J}^{\Phi,D} = S^{\Phi} \tag{3.11}$$

Through the Gaussian divergence theorem the integral, that needs to be solved, can be rewritten as

$$\int_V \nabla\cdot\boldsymbol{J}^{\Phi,D} = \int_{\partial V}\boldsymbol{J}^{\Phi,D}d\boldsymbol{A} \tag{3.12}$$

with the face normal vector $\boldsymbol{A}$. The flux is evaluated at the face center with index c of a finite volume face. Having only the discrete values at the face center the integral turns into a sum. The sum of all fluxes over all N faces equals the value of the source term in one finite volume.

$$\left[\sum_{f=0}^{N}(\boldsymbol{J}^{\Phi,D})_n\cdot\boldsymbol{A_f}\right]_C = S^{\Phi} = \int_V S^{\Phi}dV = S^{\Phi}_C V_C \tag{3.13}$$

with the volume of one finite volume cell V_C, the surface vector of a face $\boldsymbol{A_f}$.
The gradient $\nabla\Phi$ is calculated by

$$\nabla\Phi_C = \frac{1}{V_C}\sum_{f=0}^{N}\Phi_f\boldsymbol{A_n} \tag{3.14}$$

with Φ_C being the conserved quantity stored at the center of the volume V_C, Φ_f the values at the faces of the volume and the face vectors $\boldsymbol{A_n}$. To compute the value Φ_f,

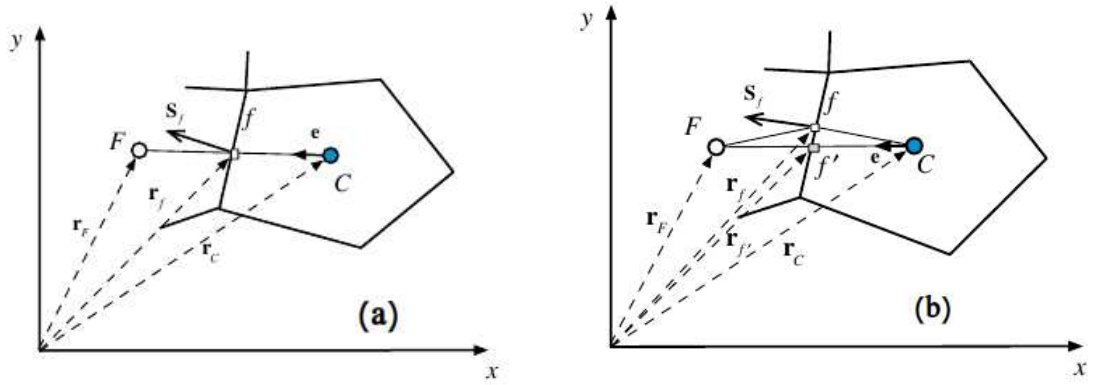


Figure 3.1: Discretization scheme for gradient and surface value in a 2D finite volume grid. a: orthogonal grid. b: non-orthogonal grid.

it needs to be interpolated, since values are normally stored at the center of a volume. The simple solution is to average the center values of the two adjacent cells C and F, shown in figure 3.1a,

$$\Phi_f = g_C \Phi_C + (1 - g_C)\Phi_F. \tag{3.15}$$

g_C is a geometric weighing factor, which denotes

$$g_C = \frac{| \boldsymbol{r_F} - \boldsymbol{r_f} |}{| \boldsymbol{r_F} - \boldsymbol{r_C} |}. \tag{3.16}$$

In a regular spaced orthogonal grid the value is simply $\Phi_f = (\Phi_C + \Phi_F)/2$. In a non-orthogonal grid the value calculated this way usually does not lie at the face centroid point f, but on a point slightly shifted away f', as shown in figure 3.1b. The value obtained that way using equation 3.15 is denoted $\Phi_{f'}$. To get the true face gradient a correction term needs to be added to equation 3.15

$$\Phi_f = \Phi_{f'} + g_C \, \nabla\Phi_C \cdot (\boldsymbol{r_f} - \boldsymbol{r_C}) + (1 - g_C) \, \nabla\Phi_F \cdot (\boldsymbol{r_f} - \boldsymbol{r_F}). \tag{3.17}$$

Next we look at the divergence operator in the convective term

$$\nabla \cdot (\rho \boldsymbol{u} \Phi) \tag{3.18}$$

with density ρ, velocity $\boldsymbol{u}$ and any scalar variable Φ. Similarly to the diffusive flux a convective flux can be defined as $\boldsymbol{J}^{\Phi,C} = \rho \boldsymbol{u} \Phi$. In the same way as it was shown before the convective flux is summarized over all faces of a finite volume.

3.1.3 Implementation of Macroscopic Solidification Model in OpenFOAM

Two different models were used to calculate the temperature distribution. The first uses the Laplacian equation for temperature diffusion using equation 3.1. The second model is more sophisticated including the phase change at a given melting temperature, the latent heat release during the phase shift and fluid flow based on the theory outlined in chapter 2.2. For non pure materials, such as alloys, the phase shift takes place in the area between solidus and liquidus line of the phase diagram. Here the latent heat is released over a certain temperature interval around the melting temperature, to model a mushy zone, where solid and liquid phases coexist. A liquidus fraction is introduced, which describes the percentage of liquid in a given volume. The basic code structure is shown for the Laplacian equation equation 3.1 and the conservation law of energy (equation 2.10).
Temperature diffusion equation:

```
1   fvm::ddt(T)
2 - fvm::laplacian(alpha, T)
3 ==
4   0
```

T is the temperature and *alpha* the diffusion coefficient.
Energy conservation law:

```
1   fvm::ddt(h)
2 - fvm::div(mDot, h)
3 ==
4     fvm::laplacian(alpha, h)
5   + fvm::Sp(S_h)
6   - fvc::(S_c)
```

h is the sensible heat, $mDot$ is a variable for the mass flow rate corresponding to the velocity in equation 2.10, S_h is the source term describing the latent heat released during phase transition and S_c a (possible) constant sink. Two operators are used in the code. The fvc :: namespace ("finite volume calculus") solves equations with an explicit schemes. The fvm :: namespace ("finite volume matrix") is used for implicit schemes.

Applications
OpenFOAM offers a variety of applications for different physical problems. In this work two applications were used: the basic 'laplacianFoam' and the more sophisticated 'buoyantPimpleFoam' with the additional 'solidificationMeltingSource' as fvOption.

laplacianFoam
This application solves the temperature diffusion equation equation 3.1. The only material specific variable is the temperature diffusion coefficient α. Solidification and the release of latent heat as well as fluid flow are not included in this model. laplacianFoam was mainly used for testing purposes of the coupled CAFV algorithm.

buoyantPimpleFoam
This application was used to model temperature diffusion and fluid flow. The additional $solidificationMeltingSource$ functionality is included to model phase changes, release of latent heat and fluid flow in the mushy region based on the theory discussed in chapter 2.2. Turbulent fluid flow, that is modeled with Reynolds-Averaged Navier-Stokes equations [20], can also be included into the application if desired. The material specific parameters, that need to be defined in this model are listed in chapter 4.6.4.

3.2 Mesoscopic - Grain Growth

3.2.1 Cellular Automaton for Dendritic Growth

A Cellular Automaton (CA) grain growth algorithm has been proposed by Gandin and Rappaz [7]. It is designed to model the growth of the envelope of the dendritic grain on the mesoscopic scale of the primary dendritic arms. The numeric domain is divided into a regular grid of square cells. Each grid cell is assigned a grain state value, determining whether it is liquid or solid. A single dendrite is modeled to be of an octahedral shape. The main half diagonals are defined by the <100> directions and correspond to the primary dendritic arms. The edges connecting the tips of the main diagonals correspond to the envelope of the secondary dendritic arms, as shown in figure 3.2a. In two dimensions the grain envelope is a square with the primary dendrite arms along the <10> directions, shown in figure 3.2b. Under uniform thermal conditions, each dendrite arm will grow with the same speed. In the presence of a temperature gradient however, the grain will grow preferably into the direction of the negative temperature gradient.

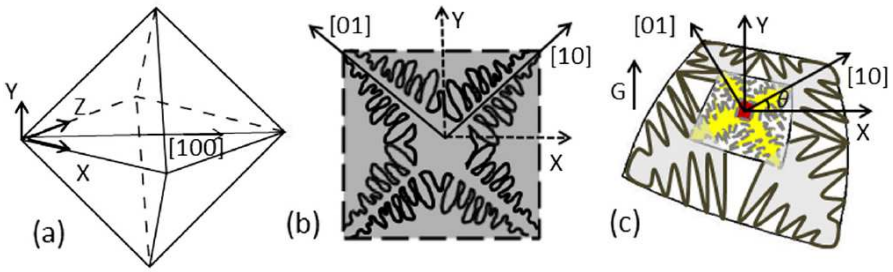


Figure 3.2: The dendritic grain envelope for uniform thermal conditions in 3D (a) and 2D (b), and in presence of a temperature gradient G in 2D (c).

Figure 3.2c shows the dendritic growth in presence of a temperature gradient along the y-axis with temperatures getting lower into negative y-direction.
Figure 3.3 shows the basic concept of the growth algorithm under uniform thermal conditions. The CA is made up of a regular grid of square cells with the grid spacing Δx. Starting point for the grain growth is a nucleation cell with the index ν. This cell has a solid grain state as well as a given crystallographic orientation. The orientation is expressed by the angle θ in respect to the x-axis of the CA. The square grows only at the tip of the main diagonals with a growth velocity depending on the local undercooling of the melt, according to equation 2.17. If the grain envelope grows, it overpasses the centers of the neighbouring cells with index μ_i. The four direct neighbors have the indices $i = 1, ..., 4$ and the second nearest neighbors have the indices $i = 5, ..., 8$. In 3D each cell has 6 direct neighbors and 20 second nearest neighbors. The square envelope of the nucleus after a certain time step t_n is denoted S^n_ν. The according half size of the square envelope at time t_n is denoted L^n_ν and can be calculated by

$$L^n_\nu = \frac{1}{\sqrt{2}} \cdot \int_{t_n}^{t} v(\Delta T_\nu(\tau))\, d\tau \tag{3.19}$$

L^n_ν depends on the growth velocity of the tips, which is dependent on the local undercooling of the surrounding melt, see equation 2.17. Fig. 3.3a shows the capture of the direct neighboring cells. If the envelope passes the center of a cell, it is considered captured by the initial cell ν. The grain state and orientation of the initial call ν are assigned to the newly captured cells $\mu_{i=1,...,4}$. Now new envelope squares are placed on the captured cells. For cell μ_1 for example, the length of the new square envelope is given by

$$L^n_{\mu_1} = min(L^n_{face1}, \sqrt{2}\Delta x) + min(L^n_{face2}, \sqrt{2}\Delta x) \tag{3.20}$$

L^n_{face1} and L^n_{face2} are the distances from the center of the captured cell μ_1 to the corners of the initial envelope. In the next step the new square is placed in such a way, that one of its corners coincides with the corner of the initial envelope, that is closest to the center of the captured cell μ_1. The length of the new square is limited to the maximum value of $L^n_{\mu_1} = 2 \cdot \sqrt{2}\Delta x$. This is the maximum size to capture the neighboring cells $\mu_{i=1,...,8}$. This condition ensures non-uniform growth of the grains in the presence of a temperature gradient. The squares of the direct neighbor cells in figure 3.3a are all the same size as the initial square and all overlap. However, this is not the case anymore with the squares of the second nearest neighboring cells $\mu_{i=5,...,8}$. The centers of these new squares are shown in figure 3.3b as $C_{\mu_5,...,\mu_8}$. The grain growth algorithm continues with the growth of envelope squares of newly captured cells. In the grain growth algorithm that is used in this work, the growth rate of the grain envelope only depends on the local thermal undercooling. Once the undercooling of the surrounding melt is big enough, the initial nucleation sites will start to grow. The undercooling at a certain grid cell denotes

$$\Delta T_{i,j,k} = T_l - T_{i,j,k} \tag{3.21}$$

and depends on the local temperature $T_{i,j,k}$ and the liquidus Temperature T_l (see equation 2.16). The liquidus temperature of the binary alloy is derived from the melting

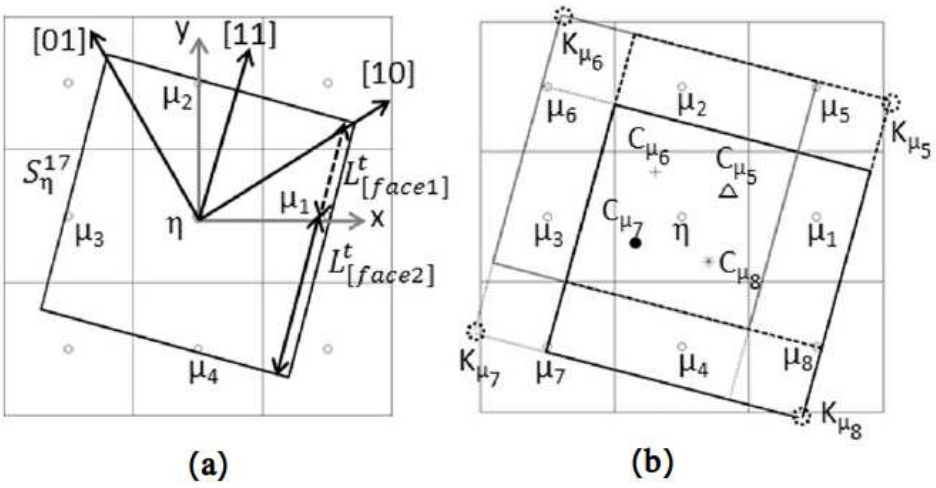


Figure 3.3: Growth algorithm of the square grain envelopes.

temperature of pure aluminum T_{melt} and the liquidus slope. The liquidus slope is linearly approximated. Depending on the concentration of the second component of the alloy, one gets a liquidus temperature different from the melting temperature of pure aluminum

$$T_{liquidus} = T_{melt} + m_L \cdot \rho_{solute}. \tag{3.22}$$

ρ_{solute} is the initial concentration of the second component of the binary alloy. δ_T is the prescribed undercooling that was assigned to each initial nucleation site. The grain growth algorithm starts when the undercooling reaches a positive value. δ_T is a random value based on a normal density distribution, that is assigned to each nucleation cell. Therefore, grain growth at the nucleation sites will start not at a sharply defined temperature value, but rather within a temperature interval. Classic nucleation theory predicts that crystals in a undercooled liquid will not form as soon as the temperature drops below liquidus temperature, but with a certain probability in the presence of seed crystals. If the undercooling condition is satisfied, the respective cell is a potential starting point for grain growth. The growth velocity of the tips of the grain envelope is governed by equation 2.17. To prevent the algorithm to become unstable due to high growth velocities in the case of high undercooling, the time step size Δt needs to be defined according to

$$\Delta t = t_{rel} - \frac{\Delta x^2}{2\alpha}. \tag{3.23}$$

with the thermal relaxation time t_{rel}, the grid spacing Δx and the diffusion coefficient α.

Initialization

Crystal grain growth in solidifying materials starts at nucleation sites, which need to

be initialized throughout the grid, before the grain growth algorithm can start. Cells within the boundaries of the numeric domain $[x_0, x_1]$ are randomly chosen based on a uniform density distribution, with following probability mass function

$$f(i; x_0, x_1) = \frac{1}{x_1 - x_0 + 1}. \tag{3.24}$$

To these cells a solid grain state is assigned. At the boundary of the domain the density of nucleation sites is higher, recognizing the fact, that the shell of the containment, will serve as nucleation site. Each nucleation site also gets a specific crystallographic orientation. The crystallographic orientation gives the orientation of the Euler angles of the grain in respect to coordinate system of the three dimensional CA. These are random values between 0° and 90°. Additionally a initial undercooling can be assigned to the chosen grain cells as well. The initial undercooling is a randomly chosen value within a certain interval, based on a normal density distribution [21]

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3.25}$$

with the mean value μ and standard deviation σ that take different values for the boundary and the bulk of the numeric domain.
The initialized nucleation sites are the starting points for the grain growth algorithm. The grain growth at a nucleation site in the mesh starts, once the condition

$$T_{i,j,k} = T_{liquidus} - \delta_T \tag{3.26}$$

is met. Here $T_{i,j,k}$ is the temperature at coordinates i, j, k, $T_{liquidus}$ is the liquidus temperature and δ_T is the initial undercooling assigned to the nucleation site.

3.2.2 Palabos Framework

The mesoscopic Cellular Automaton algorithm is implemented in palabos (**PA**rallel **LA**ttice **BO**ltzmann **S**olver), a framework for computational fluid dynamics that is based on the lattice-Boltzmann method. The palabos code structure allows for easy parallel implementation of custom routines, which has been the main reason the CA has been implemented in this solver. The most important data structures used in the simulation are $MultiScalarField3D$ and $MultiTensorField3D$. These data structures naturally support parallel computation and split up into 'atomic blocks' when the code is run parallel on multiple processors. The $MultiScalarField3D$ is a class for manipulating scalar fields and is used for the temperature field. The temperature field, that is calculated by OpenFOAM, is written onto the $MultiScalarField3D$. This way it can be split into smaller domains, corresponding to the number of processors used. In the same way MultiTensorField3D allows handling vector fields. For two dimensional cases the respective 2D datatypes are used.

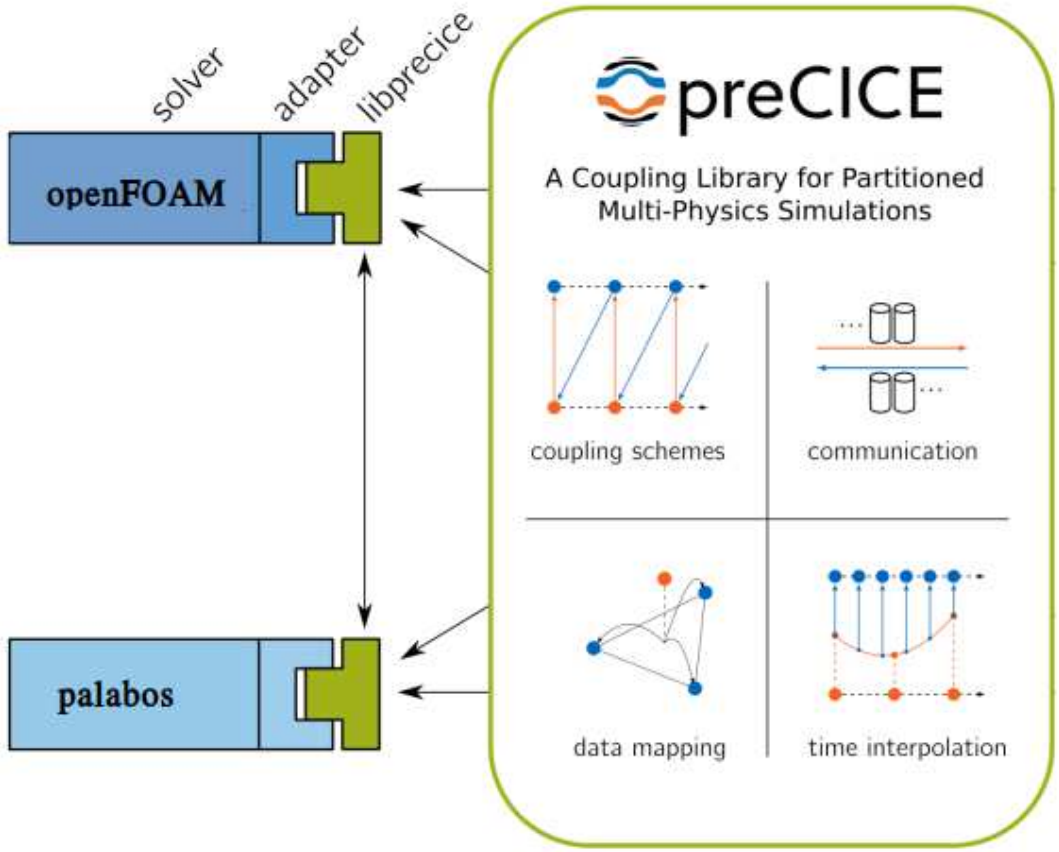


Figure 3.4: The basic outline of the coupling of two separate solvers by preCICE.

3.3 preCICE - Coupling Macro- and Mesoscopic Solvers

The mesoscopic CA solver and the macroscopic OpenFOAM solver are coupled with the coupling library preCICE [3]. It is a library to couple two ore more single physics solvers to obtain a multi-physics simulation. It is mainly used for fluid-structure interactions and heat transfer, but not restricted to these cases. Figure 3.4 outlines the basic functionality of preCICE. To couple palabos and OpenFOAM, an adapter needs to be implemented into each solver. The adapter calls the preCICE library functions, that handle the transfer of the coordinate meshes and scalar or vector fields on the grid from and to preCICE. preCICE is configured via an .xml file, where the name of the participating solvers, coupling schemes, type of mapping, numerical methods and other things are specified.

3.3.1 Interpolation and Mapping of Data

The coordinate meshes of each solver are treated by preCICE as unstructured point clouds. Each point has its respective coordinates as well as the scalar or vector data from the solver assigned to. The data fields are then mapped from the grid of the delivering solver to the grid of the receiving solver. Several mapping methods are available. The simplest is the nearest-neighbor-mapping. Here the value on each mesh point is simply passed to its nearest neighbor on the other mesh. Obviously this is only a valid method as long as the spatial resolutions of meshes are identical. For the simulations described in this work, the radial basis function mapping was used. The data points of the mesh of the delivering solver are interpolated to a globally defined function, which is then read out at the respective coordinates of the receiving mesh. The global function is constructed from a sum of N radial basis functions $\phi(r)$ for N mesh points.

$$f(x) \approx \sum_{i=1}^{N} \lambda_i \ \phi(|r|) \tag{3.27}$$

The values of the scalar field at the mesh points serve as the centres of the interpolation and they are weighted by the coefficient λ_i. The most common choice for a radial basis function is the Gaussian function $\phi(r) = exp(-(s\ |r|)^2)$. For this work the compact polynomial function $(1 - |r|)^2$ was used. To reduce the computational effort, a cutoff radius can be defined, outside which the radial basis function becomes zero. This will decrease accuracy in favor of computational time. To obtain a sufficient accuracy the cutoff radius should be at least 3 to 5 times the spacing of the interpolated mesh (see section 5.5). For performance reasons, the coarsest grid of all the coupled solvers should be chosen. The radial basis of one mesh point c_i then only takes non-zero values inside an area spanned by the cutoff radius.

$$\phi(|r|) = \begin{cases} \phi(|r|), & \text{if } |x - c_i| < r_{cutoff} \\ 0, & \text{otherwise} \end{cases} \tag{3.28}$$

Depending on the properties of the scalar or vector field, one can choose between consistent and conservative mapping. This is especially important when using meshes with different spacings. For normalized quantities such as temperature or pressure, consistent mapping is required. The value at one particular node of the coarse grid is the same as the value at the corresponding node of the fine grid (considering the interpolation done before). For quantities, that are absolute like forces, conservative mapping is needed. The value on a node of the coarse mesh is an aggregation of all nodes of the fine mesh, that correspond to that coarse node. In this work, the temperature field was mapped with the consistent mapping method.

3.3.2 Coupling Schemes

Coupling schemes describe the order at which the participating solvers are executed within a preCICE coupling step. In a serial scheme the solvers are executed one after

the other, whereas in a parallel scheme the solvers are executed simultaneously. Both, serial and parallel, can be run explicitly or implicitly.
In an explicit scheme, both solvers are executed only once per time step. The order of execution needs to be defined in the xml-configuration file. With an implicit scheme, each participant is executed multiple times per time step until a predefined convergence condition is satisfied. In this work a serial-explicit scheme was chosen. For the development and first application of the coupled grain growth simulation, the simple serial-explicit coupling was sufficient. Since there is just a one-way coupling of the temperature field and the convergence of the temperature field calculation is already managed by OpenFOAM, there is no need for an implicit coupling scheme.

3.3.3 Time Step Management

The time step size of the participating solvers usually are different. preCICE manages different step sizes. The preCICE time step is defined as a fixed value. If the participating solvers have smaller time steps than preCICE, they subcycle. After each subcycle the solver forwards the current time to preCICE, which then calculates the remaining time to reach a full preCICE time step. The remainder can also be smaller then the participant's time step size. Data between solvers is only exchanged, after one full preCICE time step is completed. The simulations run until a defined number of time steps or a defined end time is reached. For our coupled simulation the preCICE time step is the same as the time step of the macroscopic OpenFOAM solver $\Delta t_{preCICE} = \Delta t_{OpenFOAM}$. Thus, after each preCICE time step, the newly calculated temperature field is forwarded to the grain growth algorithm in palabos, which then runs multiple subcylces with the current macroscopic temperature distribution.

3.3.4 preCICE Adapter

In this section the basic structure of a preCICE adapter is outlined. The adapter is implemented in the coupled solvers and contains all functions of the preCICE library, managing the coupling. For the code of the adapter see appendix A.
The two main steps of the adapter are setting up a preCICE-mesh or more precisely a point cloud, which corresponds to the mesh of the solver and mapping the scalar and vector fields onto that point cloud. All scalar and vector values of each mesh point of the participating solver are mapped onto each corresponding point of the preCICE point cloud. The scalar and vector fields on the point cloud are then mapped onto the point clouds of the other participating solver. This of course can go both ways. After receving the mapped data fields, the values on the point cloud are read out and written onto the solver mesh. This process is repeated for each time step.

OpenFOAM Adapter

There is a preCICE adapter for OpenFOAM available, however some adjustments had to be made for the purpose of the CAFV simulation. By default, the adapter is set up to couple adjacent numerical domains, that have contact on one or more boundary

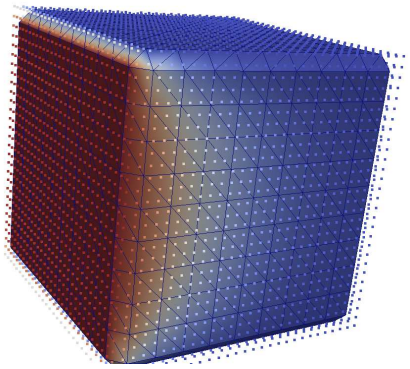


Figure 3.5: Visualisation of the mesh handling in OpenFOAM. The boundary patches are plotted as points. The volume mesh is shown as a surface plot inside.

faces. Each domain belongs to one participating solver. Data would only be transferred at the boundary faces from one solver to the other. These boundary fields are the two-dimensional patches of the OpenFOAM mesh structure, with face centered values. The three dimensional volume mesh is not accessed by preCICE. Figure 3.5 shows the mesh structure in OpenFOAM.

For the simulations done in this work various adjustments were implemented in the OpenFOAM adapter. The two participating solvers, OpenFOAM and palabos, operate in the same numerical domain, therefore the data needs to be transferred over the whole volume. The mesh points on all boundary faces of the domain are accessed through the OpenFOAM patches. Additionally also the mesh points throughout the whole volume of the domain without the patches are transferred to preCICE. All mesh points are stored into a buffer. The temperature field can be read from or written on each individual mesh point.

Datatypes

OpenFOAM, palabos and preCICE use different datatypes to store the mesh and the fields on that mesh. When implementing the preCICE adapter into the individual solvers, this needs to be considered. palabos uses the `MultiScalarField3D` and `MultiTensorField3D` datatypes to store fields on the mesh. Similarly OpenFOAM uses the `volVectorField` and `volScalarField` datatypes for storing vector and scalar fields on the mesh. In preCICE scalar and vector fields are stored in `double` arrays called `BlockScalarData` and `BlockVectorData`. In both solvers each of element of the specific datatype objects needs to be called separately and the value written to or read from the corresponding element of the `double` array of preCICE. It's important to note, that preCICE can only handle `double` and `int` datatypes.

Coupling Two Solvers

Since OpenFOAM is a finite volume code it has its data sitting at the center of a finite

volume, whereas the data in palabos is located on the cell nodes. The grid points in the two solvers do not overlap. The mesh spacing of the two meshes in palabos and OpenFAOM is also different. The temperature field from the coarse OpenFOAM mesh is mapped on a very fine palabos mesh. Therefore the temperature field points from OpenFOAM need to be interpolated to the positions in the palabos grid. The interpolation is handled by the radial basis function mapping functionality of preCICE, as outlined in section 3.3.1. At the edges of the numeric domain, data needs to be extrapolated by preCICE. Since OpenFOAM is a finite volume solver, there is no data sitting at the edge of the numeric domain. However, the finite element solver palabos needs data on the edges. The necessary extrapolation to the edges from the nearby face- and volume-centered data points is also handled by preCICE.

3.3.5 Outline of Coupled Simulation

Figure 2.6 shows the flow chart of the coupled CAFV simulation. Both solvers are managed by the preCICE library. The macroscopic FV solver OpenFoam has a large time step, which is also the global time step of preCICE. The mesoscopic CA operates at a much smaller time step. At the beginning of one global iteration, OpenFoam also iterates one time and passes the calculated temperature field over to the CA. The CA then subcycles, meaning it runs through so many iterations until it has reached the same time as the global iteration with large time step. Then the second global iteration starts. As soon as the simulation time is reached preCICE ends the global iterations and finalizes the simulation. For the code structure of a preCICE adapter implemented into a coupled solver, see appendix A.

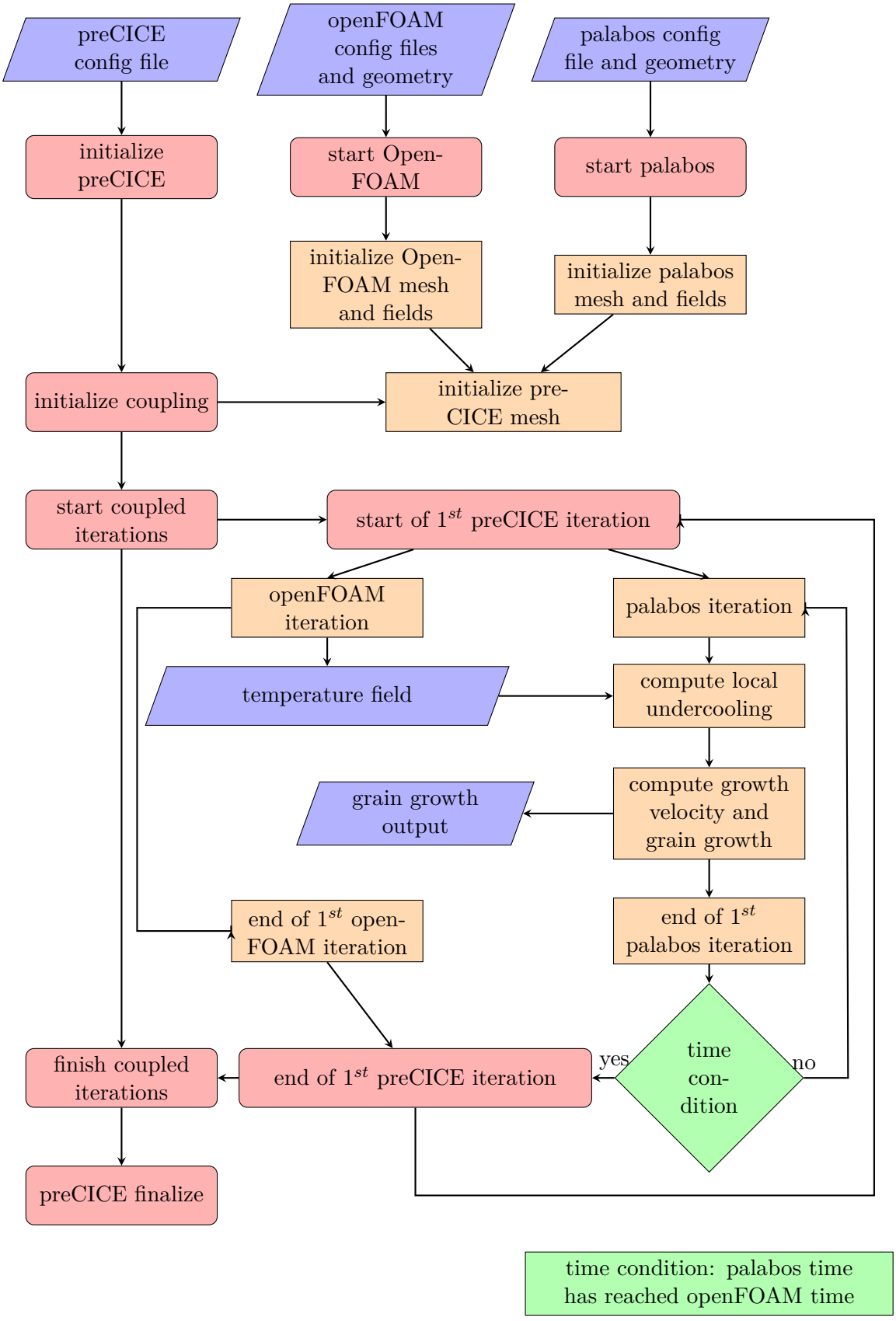


Figure 3.6: Flowchart of the coupled simulation.

CHAPTER 4

Setup

In this chapter the setups of the grain growth simulations will be presented. First the geometries, that are used in the simulations, are illustrated. Second the OpenFOAM setup is shown. This includes the specific solvers that are used as well as the applied boundary and initial conditions. Third the palabos setup is presented. In the fourth section the thermophysical parameters of the simulations are shown. In the fifth section the preCICE setup is presented. Finally the sixth section deals with the setup for every specific case, that is simulated.

4.1 Mesh and Geometry

Two geometries are used, a cube and a cylinder. Figure 4.1 and Figure 4.2 show the schematic cubic and cylindrical geometries respectively, with labelled boundary faces. The simulations using the cubic geometry were performed to ensure the functionality of the coupled simulation. The cylindrical geometry was used for physically relevant simulations. Simulations are performed with differently sized geometries. All geometries, dimensions and mesh resolutions that were used in the simulations are listed in section 4.6. The cylindrical shape cannot be discretized into a regular rectangular grid. Several mesh-specific parameters provide information about the mesh quality. The numerical properties of a mesh can be tested via an integrated OpenFOAM tool (OpenFOAM command: *checkMesh*). The reference grid spacing gives the nominal grid spacing for a rectangular domain of the same size. The aspect ratio is the ratio between the biggest and the smallest faces of the bounding box of one cell in the mesh. The mesh non-orthogonality gives the angle between the line connecting the centers of two neighboring cells and the face normal of their common face. In a rectangular mesh the non-orthogonality would be 0. The lower the non-orthogonality, the better the numeric results. Non-orthogonality values in the range $[0; 70]$ are considered appropiate, and need no further numeric treatment. Values in the range of $[70; 90]$ need non-orthogonality correctors, described in the next section. Values > 90 do not allow a stable simulation. The

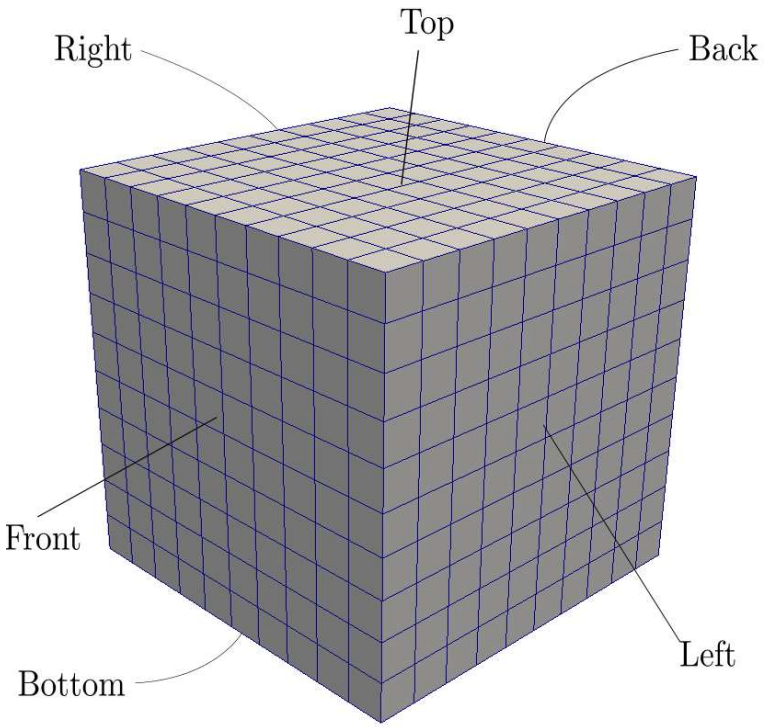


Figure 4.1: Cubic geometry with labeled boundary faces.

skewness measures the distance between two points. The fist point is the intersection of the line connecting two cell centers and the face normal of their common face. The second point is the face center of the common face of the two cells. It is the difference between the cell and a equilateral cell with an equivalent volume. The lower the skewness value, the better. The maximal value for skewness admissible in OpenFOAM is 4 for the inner volume and 20 for boundary faces. These mesh parameters are shown for each individual case in section 4.6.

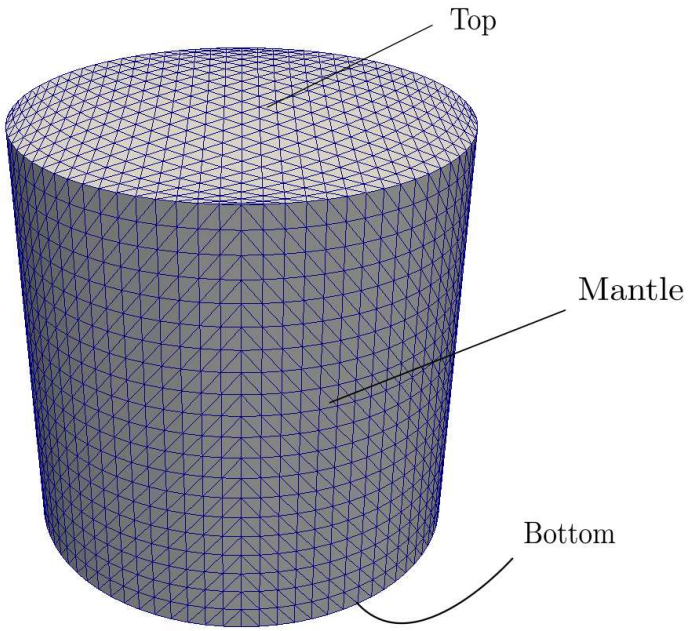


Figure 4.2: Cylindrical geometry with labeled boundary faces.

4.2 OpenFOAM setup

Two different OpenFOAM solvers are used in the simulations, the *laplacianFoam* solver and the *buoyantPimpleFoam* solver with the additional *solidificationMeltingSource* function. *laplacianFoam* only solves the simple temperature diffusion equation. The *bouyantPimpleFoam* solver additionally includes the release of latent heat during phase transition and fluid flow within the liquid regions (see section 1.1 and 2.1). This will produce physically more realistic results than just solving the simple temperature diffusion equation. Those simulations using the cubic geometry, which were only performed to verify the correct coupling of the OpenFOAM solvers with the Cellular Automaton (CA) within the palabos framework, only use the *laplacianFoam* solver. Simulations using the cylindrical geometry are performed using the *laplacianFoam* solver as well as *buoyantPimpleFoam* with *solidificationMeltingsource*.

The liquid fraction is initialized with value $f_l = 1$, meaning all of the domain is liquid. The velocity is initialized to be zero at every point in the volume and also zero at the boundary faces during the whole simulation.

The boundary conditions, that are used in the simulations, are *fixed value* and *fixed gradient* conditions. *fixed value* sets a certain constant temperature value at the boundary and *fixed gradient* sets a constant temperature gradient value $\nabla_n T$, that models a constant heat loss through the boundary face. If the heat loss is set to be zero, it is an

adiabatic boundary condition.

Discretization Schemes

The propagation in time is done via an implicit Euler scheme. The macroscopic time step is chosen to ensure the stability of the implicit-Euler time integration scheme of OpenFOAM. Therefore, it depends on the cell size of the macroscopic mesh. The time step sizes for each simulation are shown in section 4.6.

The gradient and divergence operators are discretized via a linear Gauss scheme. *Gauss* refers to the standard finite volume discretization of Gaussian integration, which is described in chapter 3. *linear* refers to the used interpolation scheme, that is used to interpolate cell-centered values to face centers. The scheme for the Laplacian is extended by a correction. This term ensures, that second order accuracy is maintained for a non-orthogonal mesh, like the cylindrical mesh. Table 4.1 shows the discretization schemes used for temporal propagation and the spatial differential operators.

Table 4.1: Discretization schemes used in OpenFOAM.

time	implicit Euler
gradient	linear Gauss
divergence	linear Gauss
Laplacian	linear Gauss (corrected)

4.3 palabos Setup

palabos uses a cartesian grid. The domain is a rectangle that envelopes the cylindrical OpenFOAM mesh completely. All cells of the palabos mesh are flagged based on wheter they lie outside, inside or right at the boundary of the OpenFOAM geometry. Figure 4.3 shows the cylindrical OpenFOAM geometry and palabos mesh, which envelopes the OpenFOAM geometry. The 2D cut in the horizontal xy plane depicts the 3 different flags, that can be assigned to palabos cells based on their position. The cells at the boundary of the OpenFOAM domain are shown in red, those in the inner volume are marked white and all cells outside are marked blue. All white and red cells are considered in the CA grain growth algorithm. The grid spacing Δx of the CA should be the same scale as the secondary dendrite arm spacing of the grains in order to accurately resolve the grains. The mesoscopic time step of the CA is chosen to be of a size, that a growing grain will not cover a distance bigger than one cell per time step (see section 3.3). Table 4.2 shows the numeric setup of the CA, which is the same for all performed simulations.

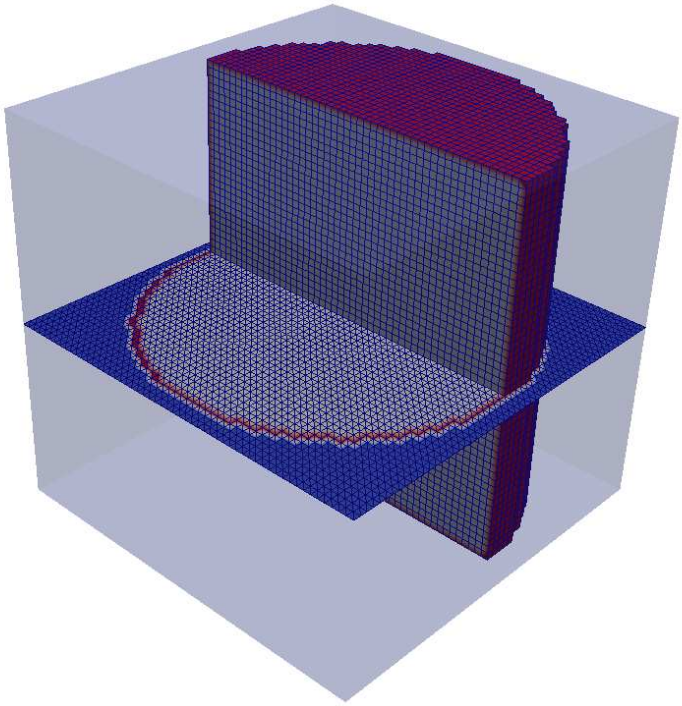


Figure 4.3: OpenFOAM geometry (cylinder) and surrounding palabos mesh. The horizontal 2D cut shows the flagged cells within the palabos mesh. The boundary cells are marked differently than those in the inner volume.

Table 4.2: Numerical parameters of the Cellular Automaton

CA time step:	$\Delta t_{CA} = 2.8 \cdot 10^{-4} s$
CA grid spacing:	$\Delta x = 2 \cdot 10^{-4} m$

4.3.1 Nucleation Sites

At the beginning of a simulation the nucleation sites have to be initialized in the Cellular Automaton (CA) (see section 2.3.2). Certain cells in the CA grid are chosen to be nucleation cells, based on a uniform density distribution. The density of nucleation sites differs between boundary and the inner volume of the mould. The boundary has a higher density of nucleation site, as grains are more likely to form at the wall of the containment. An initial undercooling is assigned to all nucleation cells. It determines the temperature, at which the grain growth algorithm starts. Table 4.3 shows the parameters for the density distributions.

Table 4.3: Distribution parameters of nucleation sites and local undercooling.

initial undercooling (volume)	normal distribution	$\mu = 5.5;\ \sigma = 0.5$
initial undercooling (surface)	normal distribution	$\mu = 0.5;\ \sigma = 0.05$
nucleation sites (volume)	density	$\rho = 1 \cdot 10^{10} m^{-3}$
nucleation sites (surface)	density	$\rho = 1 \cdot 10^{10} m^{-2}$

4.3.2 KGT model

The grain growth algorithm of the CA in palabos is based on the KGT model (see section 2.2). The growth velocity only depends on the local undercooling and two parameters (see equation 2.17). The parameters are listed in table 4.4.

Table 4.4: Distribution parameters of nucleation sites and local undercooling.

quadratic KGT parameter	$A = 8.26 \cdot 10^{-6}\ ms^{-1}K^{-2}$
cubic KGT parameter	$B = 8.18 \cdot 10^{-5}\ ms^{-1}K^{-3}$

4.4 preCICE Setup

The preCICE setup is handled via a separate preICE configuration file. The participating solvers are listed and it is determined which solvers transmit and receive data. The interpolation method for data mapping is determined and the global time step, which coincides with the biggest time step of the participating solvers, are specified. Table 4.5 lists the configurations used in the following simulations. For complete configuration file see appendix B.

Table 4.5: preCICE configuration

transmitting solver	openFOAM
receiving solver	palabos
transmitted data	temperature field (from openFOAM)
mapping scheme	radial basis function mapping
support radius	$r = 5 \cdot \Delta x_{openFOAM}$ (see individual case setups)
preCICE timestep	$\Delta t_{preCICE} = \Delta t_{openFOAM}$ (see individual case setup)

4.5 Thermophysical Parameters

All simulations are done for a binary aluminum-copper alloy $Al - 4.7wt\%Cu$. Table 4.6 lists all thermophysical parameters for the performed simulations. The input parameters are required by both, the mesoscopic CA as well as the macroscopic OpenFOAM

solvers. However, not all parameters are needed for those simulations, that use the *laplacianFoam* solver. In those cases only the thermal conductivity of the liquid phase, density and specific heat capacity for constant pressure are required. The fluid constant

Table 4.6: Thermophysical parameters [22, 23] for the coupled grain growth simulation

symbol	**parameter name**	**value**
m_{mol}	molar mass	26.982 $kg\ mol^{-1}$
R	gas constant (incompressibility)	3000 $J\ mol^{-1}K^{-1}$
ρ	density	2743 $kg\ m^{-}3$
c_p	specific heat coefficient (constant pressure)	1179 $J\ kg^{-1}K^{-1}$
H_f	latent heat release during solidification	397000 J
μ	dynamic viscosity	0.013 $kg\ m^{-1}s^{-1}$[24]
Pr	Prandtl number	0.199
T_{melt}	melting temperature for pure aluminium	933.5 K
$T_{liquidus}$	liquidus temperature of the binary alloy	917.332 K [25]
$T_{solidus}$	solidus temperature of the binary alloy	841.15 K [25]
β	thermal expansion coefficient	$1.2 \cdot 10^{-4}\ K^{-1}$
T_{ref}	reference temperature for the CA	950 K
C	initial solute concentration of copper	4.7 $wt\%$
D_s	solute diffusivity solid phase	$8 \cdot 10^{-9}\ m^2s^{-1}$
D_l	solute diffusivity liquid phase	$3 \cdot 10^{-13}\ m^2s^{-1}$
λ_s	thermal conductivity solid phase	153 $W\ m^{-1}K^{-1}$
λ_l	thermal conductivity liquid phase	77 $W\ m^{-1}K^{-1}$
m_l	liquidus slope	-344 [25]
k_{part}	partition coefficient	0.145
Γ	Gibbs-Thomson parameter	$2.14 \cdot 10^{-7}\ m\ K$

R models the compressibility of a fluid. The numerical value of $R = 3000$ gives an incompressible behavior of a perfect fluid, as suggested by OpenFOAM tutorials.

4.6 Simulation Cases

In this section the specific setup for each performed simulation is shown. Table 4.7 gives a short overview, listing the case, the geometry used and the applied macroscopic solver. There are two different geometries, a cube and a cylinder. Two different macroscopic models are used, a simple Laplacian temperature diffusion model and a more sophisticated macroscopic solidification model, that includes the release of latent heat during phase transition and fluid flow. The mesoscopic CA is the same for all cases.

Table 4.7: Overview of the simulated cases

Case 1	cube	Laplacian diffusion model
Case 2	cylinder	Laplacian diffusion model
Case 3	cylinder	phase transition + latent heat
Case 4	cylinder	phase transition + latent heat

4.6.1 Case 1 - Cube, Laplacian Diffusion Model

The setup for this uniformly discretized cube is shown in table 4.8. This case is studied to ensure the functionality of the coupled simulation. The fixed temperature boundary conditions apply a constant temperature to two boundary faces. To the other boundary faces a zero gradient condition is applied, meaning there is no heat flux through the surface. This will lead to a stationary state of a constant temperature gradient in z-direction.

Table 4.8: Setup of Case 1

General Setup	
OpenFOAM solver:	$laplacianFOAM$
time step (OpenFOAM):	$\Delta t_{OF} = 2.8 \cdot 10^{-2} s$
time step (CA):	$\Delta t_{CA} = 2.8 \cdot 10^{-4} s$

Mesh Properties	
geometry:	cube
size:	$x = y = z = 10^{-2} m$
mesh spacing (OpenFOAM):	$\Delta x = 10^{-3} m$
mesh spacing (CA):	$\Delta x = 2 \cdot 10^{-4} m$

Initial Condition	
whole volume	fixed value: $T = 933K$

Boundary Conditions	
Top	fixed value: $T = 933K$
Bottom	fixed value: $T = 900K$
Front	adiabatic: $\nabla_n T = 0 K m^{-1}$
Back	adiabatic: $\nabla_n T = 0 K m^{-1}$
Left	adiabatic: $\nabla_n T = 0 K m^{-1}$
Right	adiabatic: $\nabla_n T = 0 K m^{-1}$

4.6.2 Case 2 - Cylinder, Laplacian Diffusion Model

Case 2 is that of a cylindrical bolt. The setup is shown in table 4.9. The *laplacianFoam* solver is applied. This case is studied to again verify correct coupling and the flagging procedure in palabos. Here a constant negative temperature gradient is applied for boundary conditions.

Table 4.9: Setup of case 2

General Setup	
OpenFOAM solver:	$laplacianFOAM$
time step (OpenFOAM):	$\Delta t_{OF} = 2.8 \cdot 10^{-3} s$
time step (CA):	$\Delta t_{CA} = 2.8 \cdot 10^{-4} s$
Mesh Properties	
geometry:	cylinder
size:	$radius = 10^{-2} m; height = 10^{-2} m$
reference mesh spacing (OpenFOAM):	$\Delta x = 10^{-3} m$
mesh spacing (CA):	$\Delta x = 2 \cdot 10^{-4} m$
Mesh Quality (OpenFOAM)	
max aspect ratio	7.4839
mesh non-orthogonality	average: 18.9636°; max: 64.503°
max skewness	0.9781
Initial Condition	
whole volume	fixed value: $T = 933K$
Boundary Conditions	
Top	fixed gradient: $\nabla_n T = -10 K m^{-1}$
Bottom	fixed gradient: $\nabla_n T = -60 K m^{-1}$
Mantle	fixed gradient: $\nabla_n T = -4 K m^{-1}$

4.6.3 Case 3 - Cylinder, Latent Heat Release

Case 3 uses the same geometry and boundary conditions. as case 2, but here the *bouyantPimpleFoam* solver with the $solidificationMeltingSource$ function is applied. The setup is shown in table 4.10.

Table 4.10: Setup of case 3

General Setup

OpenFOAM solver:	$bouyantPimpleFoam$
time step (OpenFOAM):	$\Delta t_{OF} = 2.8 \cdot 10^{-3} s$
time step (CA):	$\Delta t_{CA} = 2.8 \cdot 10^{-4} s$

Mesh Properties

geometry:	cylinder
size:	$radius = 10^{-2} m; height = 10^{-2} m$
reference mesh spacing (OpenFOAM):	$\Delta x = 5 \cdot 10^{-4} m$
mesh spacing (CA):	$\Delta x = 2 \cdot 10^{-4} m$

Mesh Quality (OpenFOAM)

max aspect ratio	11.4514
mesh non-orthogonality:	max: 75.1435°, average: 20.5073°
max skewness	0.9944

Initial Condition

quantity	**area**	**applied condition**
temperature	whole volume	fixed value: $T = 933K$
velocity	whole volume	fixed value: $\boldsymbol{v} = (0,0,0) ms^{-1}$
liquid fraction	whole volume	fixed value: $f_l = 1$

Boundary Conditions

quantity	**boundary face**	**applied boundary condition**
temperature	Top	fixed gradient: $\nabla_n T = -10\ Km^{-1}$
	Bottom	fixed gradient: $\nabla_n T = -60\ Km^{-1}$
	Mantle	fixed gradient: $\nabla_n T = -4\ Km^{-1}$
velocity	Top	fixed value: $\boldsymbol{v} = (0,0,0) ms^{-1}$
	Bottom	fixed value: $\boldsymbol{v} = (0,0,0) ms^{-1}$
	Mantle	fixed value: $\boldsymbol{v} = (0,0,0) ms^{-1}$
liquid fraction	Top	zero gradient: $\nabla_n f_l = 0 m^{-1}$
	Bottom	zero gradient: $\nabla_n f_l = 0 m^{-1}$
	Mantle	zero gradient: $\nabla_n f_l = 0 m^{-1}$

4.6.4 Case 4 - Cylinder, Latent Heat Release

Case 4 has the same setup as case 3, except for the boundary conditions. The heat loss through the boundary faces is smaller and the cooling rate of the melt is slower. The simulation is performed twice. The first time with the mean initial undercooling that has been used for the other simulations as well (see table 4.3). For the second simulation the mean initial undercooling has been reduced to $\mu = 0.5$.

Table 4.11: setup of case 4

General Setup

OpenFOAM solver:	$bouyantPimpleFoam$
time step (OpenFOAM):	$\Delta t_{OF} = 2.8 \cdot 10^{-3} s$
time step (CA):	$\Delta t_{CA} = 2.8 \cdot 10^{-4} s$

Mesh Properties

geometry:	cylinder
size:	$radius = 10^{-2} m; height = 10^{-2} m$
reference mesh spacing (OpenFOAM):	$\Delta x = 5 \cdot 10^{-4} m$
mesh spacing (CA):	$\Delta x = 2 \cdot 10^{-4} m$

Mesh Quality (OpenFOAM)

max aspect ratio	7.4839
mesh non-orthogonality:	max: 64.503°, average: 18.9636°
max skewness	0.9781

Initial Condition

quantity	**area**	**applied condition**
temperature	whole volume	fixed value: $T = 933K$
velocity	whole volume	fixed value: $\boldsymbol{v} = (0,0,0) m s^{-1}$
liquid fraction	whole volume	fixed value: $f_l = 1$

Boundary Conditions

quantity	**boundary face**	**applied boundary condition**
temperature	Top	fixed gradient: $\nabla_n T = -5\ K m^{-1}$
	Bottom	fixed gradient: $\nabla_n T = -25\ K m^{-1}$
	Mantle	fixed gradient: $\nabla_n T = -5\ K m^{-1}$
velocity	Top/Bottom/Mantle	fixed value: $\boldsymbol{v} = (0,0,0) m s^{-1}$
liquid fraction	Top/Bottom/Mantle	zero gradient: $\nabla_n f_l = 0 m^{-1}$

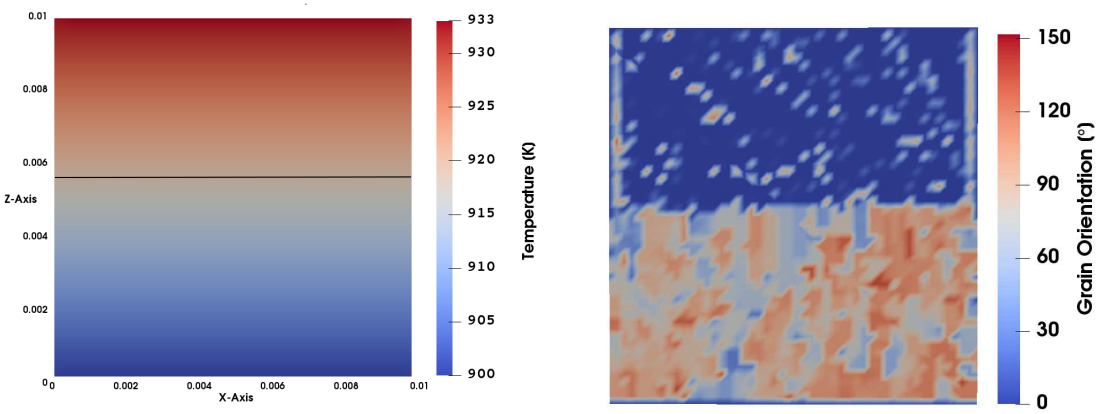
CHAPTER 5

Results and Discussion

In this chapter the results of the acquired CAFV simulations will be discussed. A main goal of this work is the application of the CAFV algorithm within bigger process simulations. The results of cases 1 to 4, as described in the previous chapter, will be shown. The results from the macroscopic solver and the results from the mesoscopic solver at corresponding time steps will be shown. Results from the macroscopic solver include the temperature distribution, velocity field and liquid fraction, depending on the model, that has been used. After that the mesoscopic grain distribution will be shown. This will be done for multiple time steps in the simulation.
Cases 1 and 2 will be focused on the correct coupling of the two solvers and the mapping of the temperature field. Here the results of the simple Laplacian diffusion model for both geometries, cubic and cylindrical, will be shown. Cases 3 and 4 show the results of the CAFV simulation using the physically more accurate model, that also considers release of latent heat and fluid flow.

5.1 Case 1 - Cube, Laplacian Diffusion Model

The first step in development was to ensure the correct mapping and interpolation functionality of preCICE and the implemented adapters. The coarse OpenFOAM mesh is mapped onto the finer palabos mesh using the radial-basis-function (rbf) interpolation scheme. Figure 5.1 shows a 2D cut through the xz-plane of the cubic domain. Figure 5.1 a shows the temperature field provided by OpenFOAM with the marked liquidus line, $T_{liquidus} = 917.332K$. Figure 5.1 b shows the corresponding grain distribution calculated by the CA. Case 1 does not simulate a physically accurate solidification process. It was performed to validate the correct coupling of the macro- and mesoscopic solvers. To test the mapping of the temperature field via preCICE the simulation was performed several times using different support radii for the rbf mapping. A 1D outline in z-direction of the temperature field, that is received by palabos, was performed. This was compared



(a) Temperature field, with marked liquidus temperature $T_{liquidus} = 917.332K$.

(b) Grain distribution

Figure 5.1: 2D cut of the cube of case 1 in the xz-plane, showing a) the temperature field and b) the corresponding grain distribution.

to the temperature field provided by OpenFOAM. The results are shown in figure 5.2 for the area near the boundary. The temperature field coming from OpenFOAM as well as the temperature field received by palabos for several support radii $(1, 2, 4, 6, 8, 10$ times the grid spacing of the coarse OpenFOAM grid) are plotted.

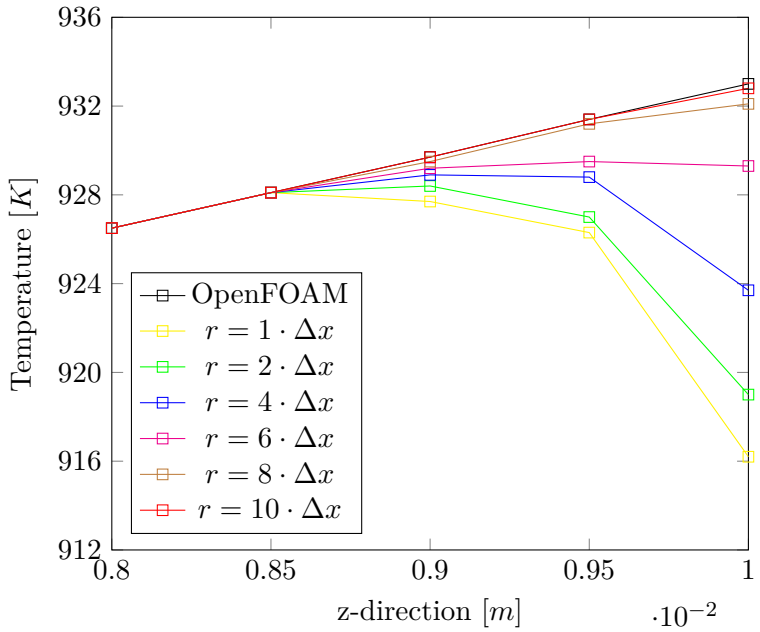
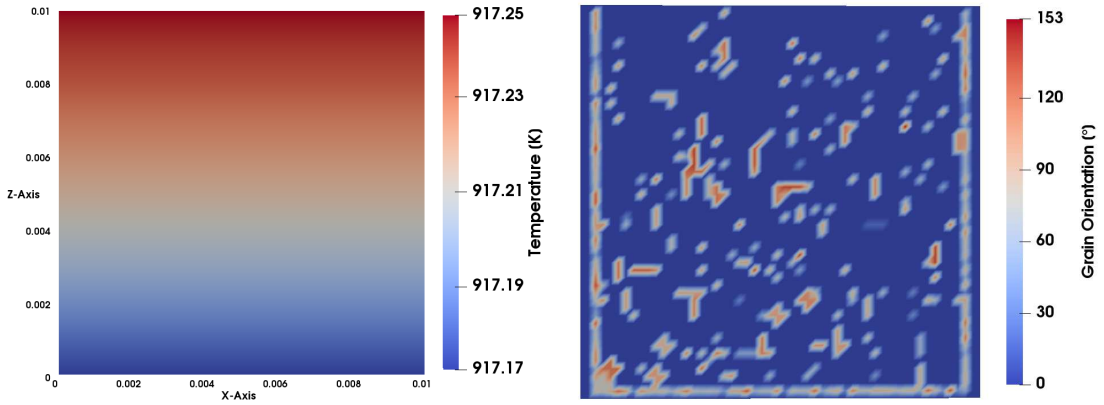


Figure 5.2: 1D lineout of the temperature field near the upper boundary for several boundary radii.

5.2 Case 2 - Cylinder, Laplacian Diffusion Model

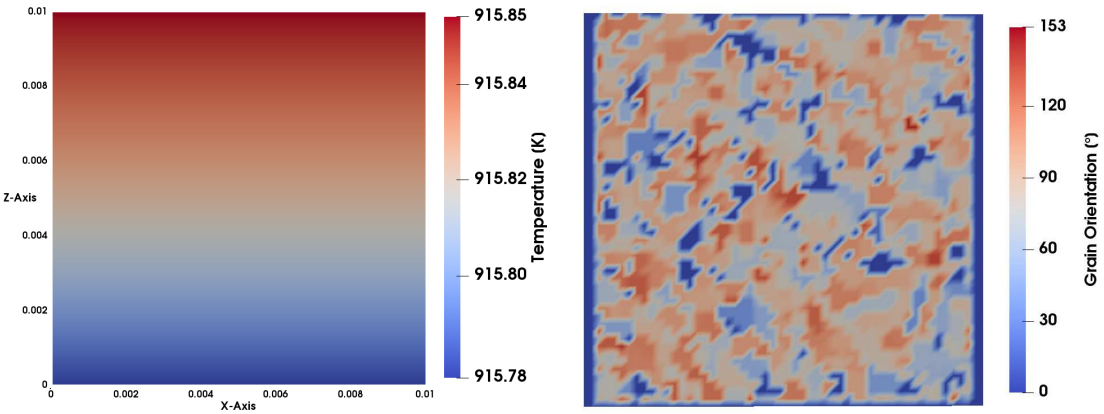
In case 2 we look at a cylindrical shape with physically more realistic boundary conditions. The macroscopic solver, however, is still the simple Laplacian diffusion solver. Figure 5.3 shows a 2D cut through the xz-plane of the cylindrical domain at the beginning of grain formation. Figure 5.3 a shows the temperature field provided by OpenFOAM and figure 5.3 b shows the corresponding grain distribution calculated by the CA. Figure 5.4 shows the same 2D cuts at a later time when the alloy is completely solidified. To



(a) Temperature field

(b) Grain distribution

Figure 5.3: 2D cut of the cylinder of case 2 in the xz-plane at the beginning of grain formation, showing a)the temperature field and b) the corresponding grain distribution.



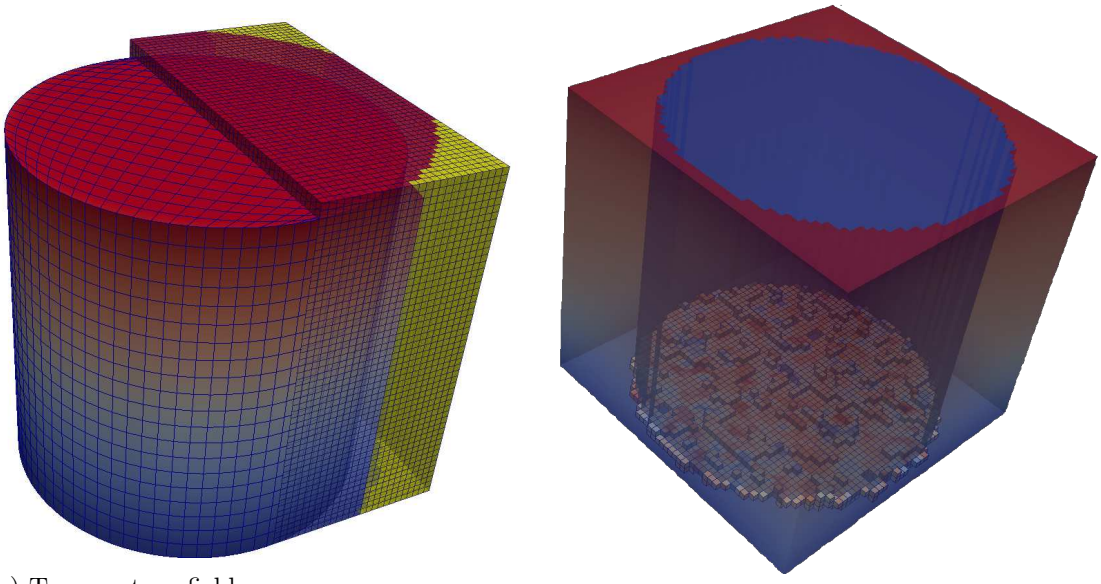
(a) Temperature field

(b) Grain distribution

Figure 5.4: 2D cut of the cylinder of case 2 in the xz-plane at complete solidification, showing a)the temperature field and b) the corresponding grain distribution.

couple the cylindrical OpenFOAM domain with the regular rectangular palabos grid, all

cells of the palabos grid were flagged in respect to their position. Only those cells at the boundary and inside of the cylindrical domain take part in the grain growth algorithm of the CA. Figure 5.5 a shows the coupling of the OpenFOAM and palabos mesh. The temperature field provided by OpenFOAM is extrapolated by the rbf mapping scheme of preCICE beyond the boundaries of the cylinder. The yellow cells at the corners of the palabos mesh lie outside of the range of the rbf mapping scheme and did not receive any temperature at all. Figure 5.4 b shows the grains that grow only in those palabos cells, that lie inside of the cylindrical domain.



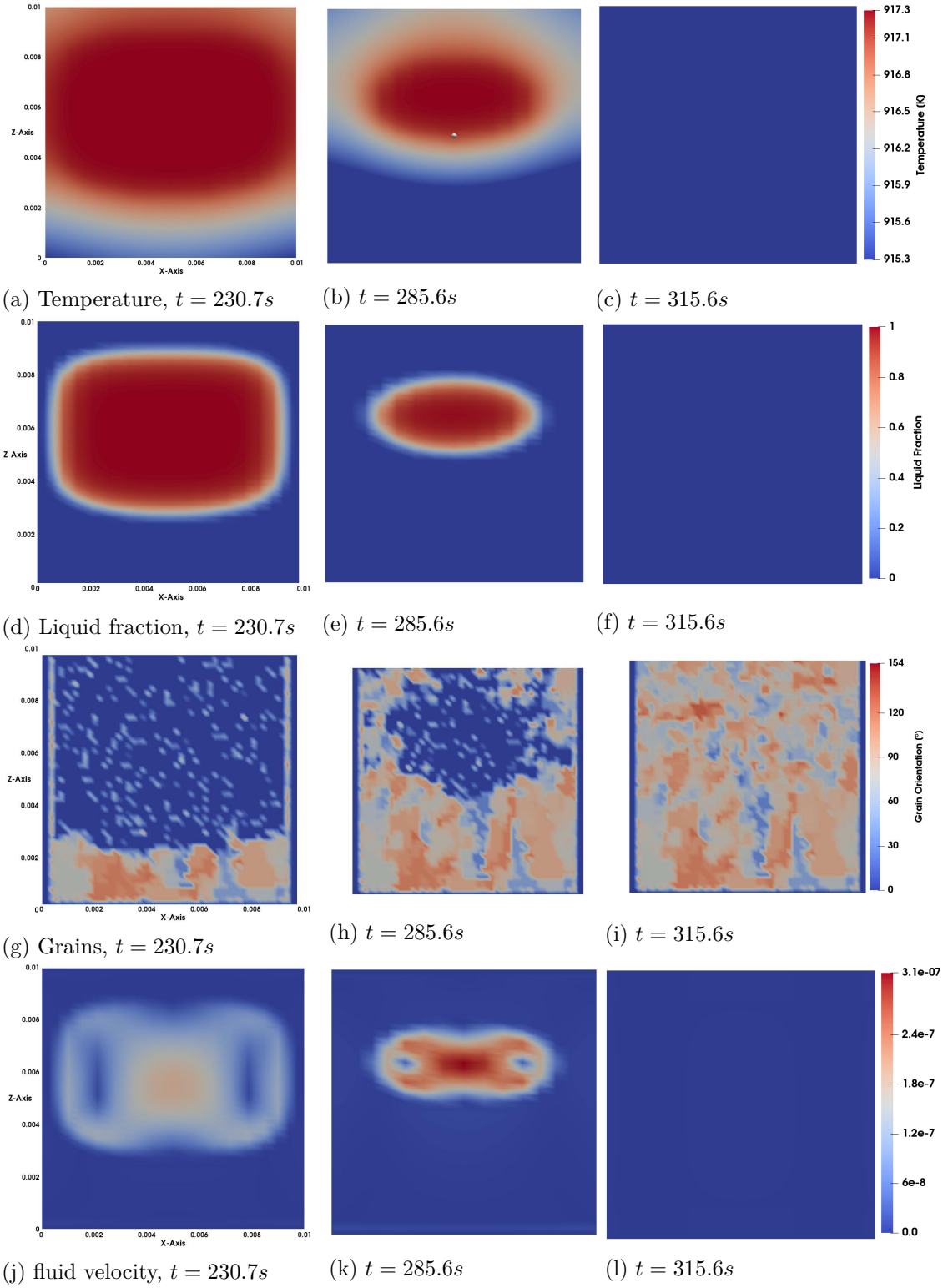
(a) Temperature field

(b) Grain distribution

Figure 5.5: Cylindrical OpenFOAM geometry that is engulfed by the regular palabos grid. a) shows the temperature mapping and extrapolation onto the palabos cells, that lie outside of the OpenFOAM domain. b) shows the grains only growing inside of the cylindrical OpenFOAM domain.

5.3 Case 3 - Cylinder, Latent Heat Release

In case 3 we look at a cylindrical shape with the same boundary conditions as in case 2. The macroscopic solver is the more sophisticated *buoyantPimpleFoam* solver, that incorporates the realease of latent heat and fluid flow. Figure 5.6 shows a 2D cut through the xz-plane of the cylindrical domain at three different times. Figure 5.6 a-c show the temperature distribution, d-f show liquid fraction, g-i show the grain distribution generated by the CA and j-i show the fluid velocity of the liquid melt.



(a) Temperature, $t = 230.7s$ (b) $t = 285.6s$ (c) $t = 315.6s$

(d) Liquid fraction, $t = 230.7s$ (e) $t = 285.6s$ (f) $t = 315.6s$

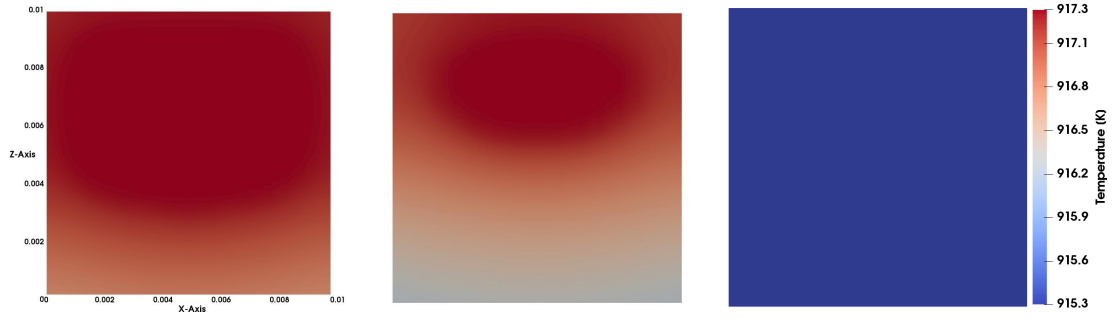
(g) Grains, $t = 230.7s$ (h) $t = 285.6s$ (i) $t = 315.6s$

(j) fluid velocity, $t = 230.7s$ (k) $t = 285.6s$ (l) $t = 315.6s$

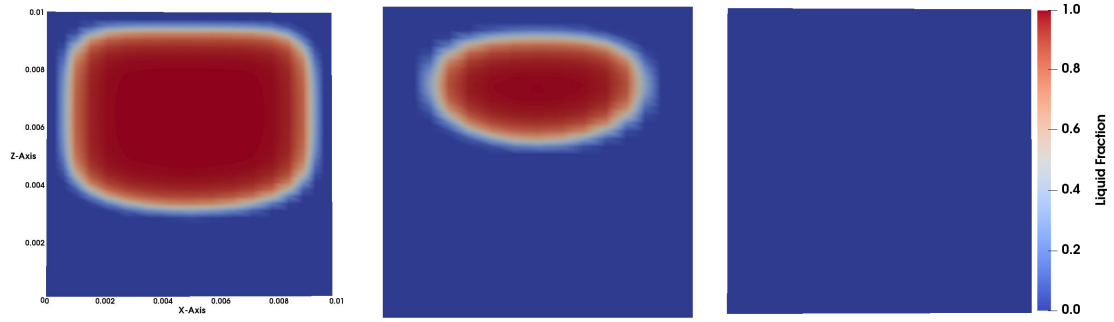
Figure 5.6: 2D cut of the cylinder of case 3 in the xz-plane, showing the temperature field (a,b,c), the liquid fraction (d,e,f), the grain distribution (g,h,i) and the fluid velocity (j,k,l) at three different times.

5.4 Case 4 - Cylinder, Latent Heat Release

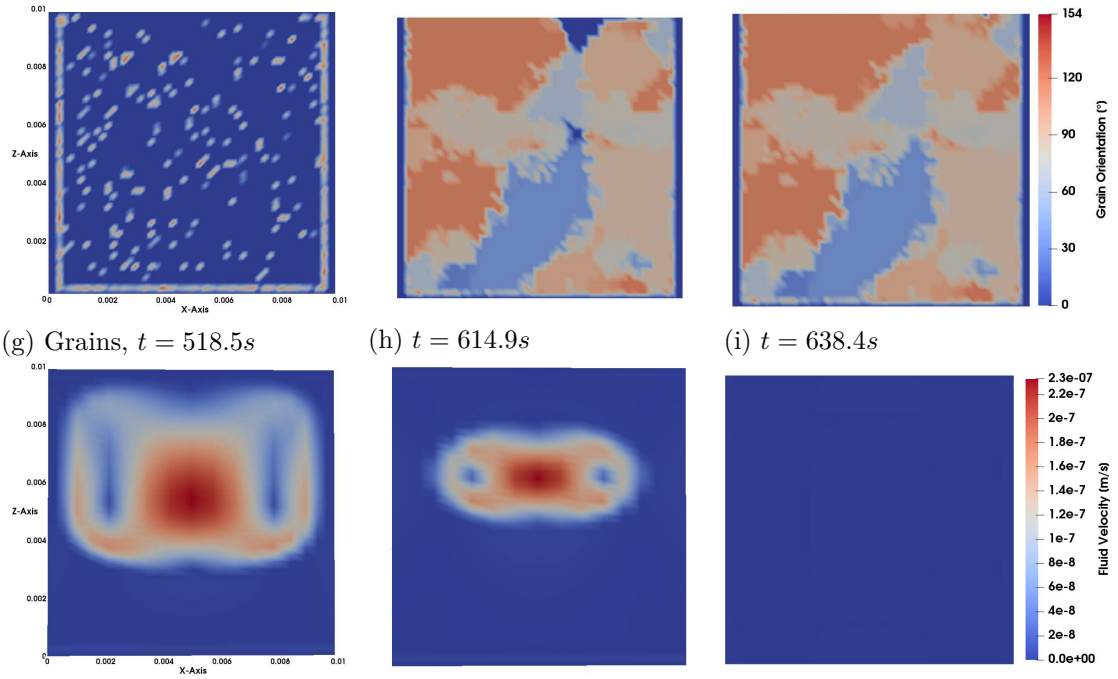
Case 4 has different boundary conditions than case 3. There is less heat loss through the boundary faces, therefore a slower cooling rate and a smaller temperature gradient. The simulation of case 4 was performed twice. The first run displayed in figure 5.7 was performed with the mean initial undercooling, that has also been used in the previous cases (see table 4.3). The second run displayed in figure 5.8 was performed with a smaller mean initial undercooling of $\mu = 0.5$.
Figure 5.7 a) shows the temperature distribution, b) the velocity and c) the liquid fraction, all provided by the macroscopic solver. Figure 5.7 d) shows the grain distribution generated by the CA. Figure 5.7 displays the beginning of solidification and Figure 5.7 shows complete solidification at a later time. Figure 5.8 shows the grain distribution in case of a smaller mean undercooling of the nucleation sites.



(a) Temperature, $t = 518.5s$ (b) $t = 614.9s$ (c) $t = 638.4s$



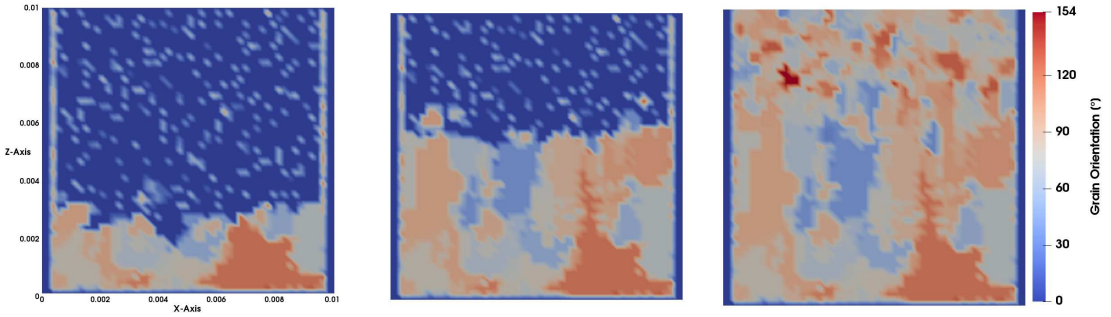
(d) Liquid fraction, $t = 518.5s$ (e) $t = 614.9s$ (f) $t = 638.4s$



(g) Grains, $t = 518.5s$ (h) $t = 614.9s$ (i) $t = 638.4s$

(j) Fluid velocity, $t = 518.5s$ (k) $t = 614.9s$ (l) $t = 638.4s$

Figure 5.7: 2D cut of the cylinder of case 4 in the xz-plane showing the temperature field (a,b,c), the liquid fraction (d,e,f), the grain distribution (g,h,i) and the fluid velocity (j,k,l) at three different times.



(a) Grains, $t = 518.5s$ (b) $t = 614.9s$ (c) $t = 638.4s$

Figure 5.8: 2D cut of the cylinder of case 4 in the xz-plane showing the grain distribution. The mean initial undercooling assigned to the nucleation sites was $\mu = 0.5$ - lower than in figure 5.7.

5.5 Data Mapping

Figure 5.1 shows qualitatively the correct coupling of the CAFV simulation. The grains are located in an area with a temperature below the liquidus temperature of $T_{liquidus} = 917.332K$.

Figure 5.2 shows the temperature field, that has been mapped to palabos by preCICE, using rbf interpolation. In the inner volume of the numeric domain, there are virtually no deviations, regardless of the support radius. Errors only show at discontinuities of the interpolated function, such as the boundaries of the mesh. For a smooth function interpolation errors will be very small even for small support radii. However, in the presence of abrupt changes, non-continuously differentiable sections and the edges of the numeric domain, the mapping function will show errors that grow inversely proportional to the support radius. Grain growth in CAFV simulation mainly starts at the boundaries of the numeric domain, since the walls of the mould serve as nucleation site. Therefore providing accurate temperature values in these areas is especially important. To obtain acceptable accuracy at the boundaries of the domain, a minimum support radius of 4 to 6 times the grid spacing of the coarse grid is needed. Figure 6.1 shows the mapping error in relation to the support radius at the boundary of the domain. At boundary edges of the domain preCICE needs to extrapolate the volume and face centered values of the OpenFOAM mesh to obtain values for the finite elements of the palabos mesh. Figure

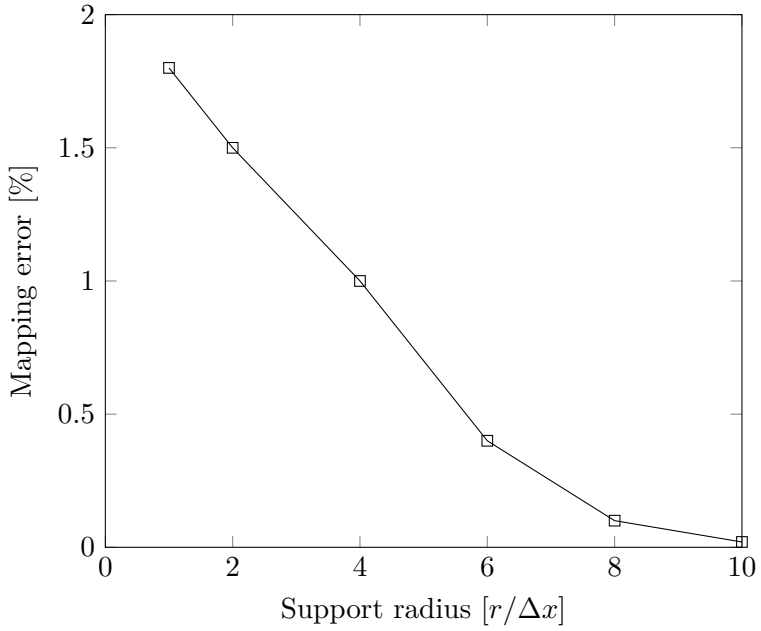


Figure 5.9: Mapping error of radial-basis-function mapping scheme of preCICE in relation to the support radius.

5.5 a shows the temperature mapping from the cylindrical OpenFOAM mesh onto the

regular palabos mesh. The palabos cells, that lie outside of the cylinder, receive a temperature value, that has been extrapolated by preCICE using the rbf mapping scheme. The yellow cells at the corners of the palabos mesh lie outside of the range of the rbf extrapolation and did not receive a temperature value. The reach of the extrapolated field depends on the support radius of the rbf method. Figure 5.5 b shows that only those palabos cells inside of the cylinder take part in the grain growth algorithm of the CA.

5.6 Solver Coupling

In this section the correct coupling of the macroscopic solver providing the temperature field and the mesoscopic CA executing the grain growth algorithm is analysed. Figure 5.1 a shows the temperature distribution with the liquidus temperature marked. Figure 5.1 b shows the grains growing in the area with temperatures below the liquidus temperature line. The solid-liquid interface does not reach to the liquidus temperature line, which is in accordance to the underlying theory. This confirms the correct coupling of the two solvers. The temperature field delivered by OpenFOAM is correctly mapped onto the palabos mesh. The grains grow in areas with a local undercooling, meaning the temperature of the melt lies below the liquidus temperature of the alloy of $T_{liquidus} = 917.332K$. A closer look at the grain distribution reveals, that the grains preferably grow into the positive z-direction which is the direction of the negative temperature gradient. This is also consistent with the theory of grain growth in undercooling melts. The nucleation sites located in areas with temperatures above $T_{liquidus}$, have not been starting points of grain growth.

5.7 Effects of Macroscopic Models

In Case 2 more realistic boundary conditions were applied, however, the macroscopic solver still uses the simple Laplacian diffusion solver. Figures 5.3 and 5.4 show temperature and grain distribution for case 2 at two different times. Neglecting latent heat release during solidification leads to a very fast drop of temperature, due to the high amount of heat lost through the boundaries. This leads to an instant drop of temperature over the whole domain with a very small temperature gradient. This shows, that the Laplacian solver cannot deliver physically accurate results for macroscopic solidification processes, at least not with boundary conditions meant to be applied to the macroscopic solidification solver including the release of latent heat during phase shift. In case 2 this leads to a very high cooling rate with a very small temperature gradient. Figures 5.3 b and 5.4 b show the grains growing according to these conditions. Due to the very small temperature gradient, grains growth starts rather uniformly over the whole domain. Due to the very high cooling rate, almost all nucleation cells become a starting point of grain growth before they could have been outgrown by a neighbouring grain. These conditions lead to a high number of small grains, that do not show a preferred growth direction.

For cases 3 and 4 the main plausibility test is the comparison of the liquid fraction delivered by the macroscopic solver with the grain distribution of the CA. Areas with a small liquid fraction are almost completely solidified, according to the calculation of the macroscopic solver. If the grain distribution of the CA matches those areas of small liquid fraction, it means that the simulation results can be considered physically plausible, at least qualitatively. Figure 5.6 shows the solidification process of case 3 at 3 different time steps for each, temperature field, liquid fraction, grain distribution and fluid velocity. Figures 5.6 a,d,g,j show the solidification at an early stage, b,e,h,k show solidification at a more advanced stage and c,f,i,l show complete solidification. Comparing the liquid fraction with the grain distribution shows, that the grains form in areas with a liquid fraction of $f_l \leq 0.1$. The tips of some grains are reaching into areas with higher liquid fractions. At the second time, as depicted in figure 5.6 e, a small area in the upper half of the cylinder is still liquid, with a liquid fraction of $f_l = 1$, while the areas near the boundaries of the cylinder are already solidified. Figure 5.6 h shows a matching grain distribution. The grains cover the parts with $f_l = 0$ quite accurately. Figures 5.6 a-c show the corresponding temperature dsitribution. Though the temperature of most of the domain is lower than the liquidus temperature, grains only start to form at the bottom of the cylinder with a certain local undercooling. Figures 5.6 j,k show the limited fluid flow in the area of a liquid fraction of $f_l \geq 0.8$. Fluid flow has a small increasing effect on the cooling rate, by transporting hot melt from the inner volume of the cylinder to the boundaries.

The agreement of the grain distribution with the temperature field and liquid fraction cannot be completely accurate. There will always be a mismatch of the solid-liquid interface in the CA and in OpenFOAM. There are several reasons for mismatches: First the different spatial resolution of the coupled grids. The coarse OpenFOAM grid certainly cannot resolve the solid-liquid interface produced in the CA. Second the different models used. The macroscopic solvers work with mean values of solid and liquid fractions over each finite volume and therefore cannot simulate an actual solid-liquid interface. Third the way of coupling. The coupling implemented in this work is a simple one way coupling. OpenFOAM does not get any information back from the CA.

5.8 Effects of Boundary Conditions

Case 4 differs from case 3 by different boundary conditions, that cause a slower cooling rate and a smaller temperature gradient, as shown in figure 5.7. Thus the temperature gradient in the cylinder at any given time is smaller than in case 3. Due to the slower cooling, only a few nucleation sites serve as starting points for grain growth. At a time, where a considerable amount of the domain already is solidified according to the liquidus fraction, there have still not been any grains formed. Once the undercooling condition is satisfied for a few nucleation sites, those grains have enough time, to grow over other nucleation sites. The resulting grain distribution displayed in figures 5.7 g,h,i shows bigger and less numerous grains than in case 3. Grains also will grow partly in

areas of high liquid fraction, whereas the area with low liquid fraction is not fully filled with grains yet. This mismatch is possibly due to the very small temperature gradient. The differences in temperature over the whole domain are smaller than the mean initial undercooling and grain growth is likely to start at any nucleation site throughout the domain. At very slow cooling rates, the sources of error discussed in the previous section seem to have a bigger effect, thus leading to more inaccurate results.

5.9 Effects of Initial Undercooling

A way to tackle these problems, is to reduce the mean initial undercooling of the nucleation sites. The initial undercooling assigned to nucleation sites leads to grain growth not starting precicely once the local temperature is below the liquidus temperature, but rather in a certain temperature interval below the liquidus temperature. If the temperature difference over the whole domain is smaller than this interval, due to a slow cooling rate, grains can possibly form anywhere, not matching the liquidus and solidus fraction of the macroscopic solver anymore. The grain distribution with a smaller mean initial undercooling is schown in figure 5.8. Now the local undercooling to trigger the grain growth algorithm at any given nucleation site does not need to be as low as before. Grains will start to grow within a temperature interval closer to the liquidus temperature. Grain growth will be more sensitive to small temperature changes. This leads to more nucleation sites eligible to serve as starting points for grain growth, thus producing more smaller grains and a grain distribution that better coincides with the liquid fraction of the macroscopic solver, similar to case 3. Values of the mean initial undercooling have to be determined experimentally [4, 21]. If the experimentally obtained values do not support the reduction of the mean initial undercooling, then better results can only be obtained by choosing boundary conditions that lead to a more rapid cooling of the melt.

CHAPTER 6

Summary and Outlook

The main objective of this work was the implementation of a CAFV simulation, for modelling dendritic grain growth during solidification, by establishing a working coupling of a macroscopic finite volume solver and a mesoscopic Cellular Automaton. For the macroscopic part of the simulation a solidification solver of OpenFOAM has been used. This solver models the solidification of a liquid including the release of latent heat during phase transition and fluid flow of the liquid. The resulting temperature field is stored on a coarse numerical grid. The coupled CA is operating on a much finer numerical grid. The grain growth algorithm is derived from the KGT model. It models a rectangular grain envelope at the length scale of the primary dendrite arms of the grain. The growth rates of the dendrite grains only depend on the local undercooling and therefore on the temperature field calculated by the macroscopic solver. The CA is implemented into the palabos framework, because the palabos code structure is designed for parallel processing and allows for implementation of additional custom routines. The two solvers are coupled by the coupling library preCICE. Grain growth and heat diffusion are performed in separate solvers to reduce the computational cost of the simulation. The macroscopic solver has larger time steps and a larger grid spacing, which is sufficient for the macroscopic solidification model. The grain growth algorithm of the CA requires a much finer grid with smaller timesteps.
The temperature field is passed from OpenFOAM to palabos. The values of the coarse grid from OpenFOAM are mapped onto the much finer grid of the CA by radial-basis-function interpolation. Several tests were performed to determine the accuracy of the data mapping on preCICE. At the boundaries of the numeric domain as well discontinuities of the temperature field preCICE produces interpolation errors, that are inversely proportional to the chosen support radius. The tests show reasonably accurate interpolation for radii of 4-6 times the grid spacing of the coarse grid.
The CAFV simulations show plausible results for high cooling rates. The grain distribution produced by the CA, conincides with the temperature field and the liquid fraction coming from the macroscopic solver. For low cooling rates, the resulting grain distribu-

tion shows less numerous and bigger grains. Here grain growth does not concide well with the liquid fraction from the macroscopic solver. Reducing the mean initial undercooling can help to tackle the problems arising at low cooling rates. The values for the mean initial undercooling need to be determined experimentally.

Ultimately, the coupled CAFV simulation developed in this work will be part of a larger process simulation, providing valuable information of the grain structure for further forming simulations. Several improvements can be implemented in the future to tackle the sources of errors discussed in the previous chapters.
A two way coupling can be established. The liquid fraction can be calculated in the CA by counting the number of solidified cells in a certain volume, that matches the finite volumes of the macroscopic solver in size. This way the macroscopic solver will get an accurate liquid fraction value, which in turn will lead to a more accurate modelling of the latent heat release.
Also, the CA can be improved by using other grain growth models. The LGK model by Lipton, Glicksman and Kurz [11], introduces a growth rate, that not only depends on the local undercooling, but also on the local concentration of the solute. In this model the rejection of solute at the solid-liquid interface out of the crystal into the liquid is considered. Since the solute concentration depends on the mesoscopic grain growth, this would also call for a two way coupling of the participating solvers. The macroscopic solver would provide temperature and solute concentration but also would have to receive the new concentration of the solute from the CA every time step.

CHAPTER 7

Acknowledgement

The author would like to thank the State of Upper Austria for financial support of this research work in the frame of the project PSHeRo (#WI-207-289120/16) within the strategic programme „Innovative Upper Austria 2020“. Furthermore the author would like to thank the thesis advisor Stephan Jäger for his valuable support and feedback throughout the project, Matthias Hartmann for providing precious support writing the thesis and all members of the Leichtmetallkompetenzzentrum Ranshofen (LKR), where this project was performed.

Bibliography

[1] W. F. Smith, J. Hashemi. Foundation of materials science and engineering. page 242, 2006.

[2] H. G. Weller, G. Tabor, H. Jasak, C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, 12(6), 1998.

[3] H.-J. Bungartz, F. Lindnder, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann. precice - a fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250–258, 2016.

[4] M. Rappaz. Modelling of microstructure formation in solidification processes. *International Materials Reviews*, 34(3), 1989.

[5] M. Rappaz, Ch.-A. Gandin. Probabilistic modelling of microstructure formation in solidification processes. *Acta mater.*, 41(2):345–360, 1993.

[6] Ch.-A. Gandin, R. J. Schaefer, M. Rappaz. Analytic and numerical predictions of dendritic grain envelopes. *Acta mater.*, 44(8):3339–3347, 1996.

[7] Ch.-A. Gandin, M. Rappaz. A 3d cellular automaton algorithm for the prediction of dendritic grain growth. *Acta mater.*, 45(5):2187–2195, 1997.

[8] W. Kurz, B. Giovanola, R. Trivedi. Theory of microstructural development during rapid solidification. *Acta metall.*, 34(5):823–830, 1986.

[9] M. Rappaz, Ph. Thevoz. Solute diffusion model for equiaxed dendritic growth: Analytical solution. *Acta Metall.*, 35(12):2929–2933, 1987.

[10] M. Wu, A. Ludwig. Modeling equiaxed solidification with melt convection and grain sedimentation. *Acta Mater.*, 57:5621–5631, 2009.

[11] J. Lipton, M. E. Glicksman, W. Kurz. Dendritic growth into undercooled alloy melts. *Materials Science and Engineering*, 65:57–63, 1984.

[12] Ch.-A. Gandin, M. Rappaz. A coupled finite element-cellular automaton model for the prediction of dendritic grain structures in solidification processes. *Acta mater.*, 42(7):2233–2246, 1994.

[13] A. Rai, H. Helmer, C. Körner. Simulation of grain structure evolution during powder bed based additive manufacturing. *Additive Manufacturing*, 13:124–134, 2017.

[14] V. R. Voller, C. Prakash. A fixed grid numerical modelling methodology for convection-diffusion mushy region phase-change problems. *Int. J. Heat Mass Transfer*, 30(8):1709–1719, 1987.

[15] S. Whitaker. Flow in porous media: A theoretical derivation of darcy's law. *Transport in Porous Media*, 1:3–25, 1986.

[16] J. Bragard, A. Karma, Y. H. Lee, M. Plapp. Linking phase-field and atomistic simulations to model dendritic solidification in highly undercooled melts. *Interface Science*, 10:121–136, 2002.

[17] F. Moukalled, L. Mangani, M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics*, volume 113. Springer International Publishing Switzerland, 2016.

[18] The openfoam foundation. `https://openfoam.org/`.

[19] A. Fick. On liquid diffusion. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 10(63):30–39, 1855.

[20] O. Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London. A*, 186:123–164, 1895.

[21] Ch.-A. Gandin, J.-L-Desbiolles, M. Rappaz, Ph. Thevoz. A three-dimensional cellular automaton-finite element model for the prediction of solidification grain structures. *Metallurgical and Materials Transactions A*, 30, 1999.

[22] aluminum properties. `https://www.webelements.com/aluminium/`.

[23] copper properties. `https://www.webelements.com/copper/`.

[24] N. Konstantinova, A. Kurochkin, P. Popel. Viscosity and volume properties of the al-cu melts. *EPJ Web of Conferences*, 15, 2011.

[25] X.-Y. Yan, Y.A. Chang, S.-L. Chen, F. Zhang, S. Daniel. Calculated phase diagrams of aluminum alloys from binary al–cu to multicomponent commercial alloys. *Journal of Alloys and Compounds*, 320:151–160, 2001.

Appendix

A preCICE Adapter

Listing 1: Basic code structure of the preCICE adapter.

```
#include "precice/SolverInterface.hpp"
using namespace precice;

SolverInterface( "GrainSolver", rank, size );
precice.configure("precice-config.xml");

int dim = precice.getDimensions();
int vertexSize = Nx*Ny*Nz;
int* vertexIDs = new int[vertexSize];
int meshID = precice.getMeshID("GrainMesh");
int temperatureID = precice.getDataID("Temperature", meshID);

double *temperature = new double[vertexSize];
double* mesh = new double[vertexSize*dim];

precice.setMeshVertices(meshID, vertexSize, grid, vertexIDs);

double dt_precice = precice.initialize();
precice.initializeData();

while (precice.isCouplingOngoing()){
    dt = min(dt_precice, dt);

    precice.readBlockScalarData(temperatureID, vertexSize, vertexIDs, temperature);

    ParticipatingSolver (...) ;

    precice.writeBlockScalarData(dataID, vertexSize, vertexIDs, data);

    dt_precice = precice.advance(dt);
}

precice.finalize()
```

First the preCICE header file needs to be linked. Then the API is initialized with `SolverInterface`(4), which takes the name of the participating solver, as well as the rank and size of the current thread as input arguments. `GrainSolver` is the name of the coupled CA. If the simulation is not run in parallel, rank and size are 0 and 1 respec-

tively. The `configure()` method reads in the .xml configuration file. Next, various preCICE variables are initialized (7-14). `dim` is the physical dimension of the simulation (2D or 3D), `vertexSize` is the number of points in the mesh. The mesh array defines the point cloud, that is handed over to preCICE. Each point gets an entry for it's x-,y-, and z- coordinates. For the mesh and the temperature array, there is also a seperate ID variable, that is assigned with the corresponding name from the xml configuration file. To hand the mesh over to preCICE, the preCICE library function `setMeshVertices` is called (16). After the mesh is defined, preCICE can be initialized using the `initialize()` library function (18) and the value of the preCICE time step is called. `initializeData()` (19) maps data between the solvers before the first time step, if initial values are needed. After the initialization, the main loop starts (21-30). The `isCouplingOngoing()` function checks, whether the maximum number of steps or the end time has already been reached. Scalar or vector data coming from the other solver is received by preCICE with `readBlockScalarData()` or `readBlockVectorData()` respectively. For writing data to be passed on to the coupled solvers, `writeBlockScalarData()` or `writeBlockVectorData()` are called. `ParticipatingSolver()` represents any solver-specific functions that are called during the current time step.
To advance the simulation for one time step, the `advance` function is called. It passes the time step `dt`, that was used for the current iteration to preCICE and receives the time step for the next iteration. This can either be the current time step or the remainder, that is left for the global preCICE time step to be completed. In the latter case, the solver would subcycle. For the current iteration the smaller time step is used. This ensures, that the solver would not advance beyond the current preCICE-time step. After the main loop is exited, preCICE is ended by the `finalize()` function call (33).

B preCICE configuration

Listing 2: preCICE configuration file.

```
<?xml version="1.0"?>

<precice-configuration>
    <log>
        <sink type="stream" output="stdout" filter="%Severity% > debug"
            enabled="true"/>
    </log>

    <solver-interface dimensions="3">

        <data:scalar name="Temperature"/>

        <mesh name="openfoamMesh">
            <use-data name="Temperature"/>
        </mesh>

        <mesh name="palabosMesh">
            <use-data name="Temperature"/>
        </mesh>

        <participant name="openfoamSolver">
            <use-mesh name="openfoamMesh" provide="yes"/>
            <write-data name="Temperature" mesh="openfoamMesh"/>
        </participant>

        <participant name="palabosSolver">
            <use-mesh name="openfoamMesh" from="openfoamSolver"/>
            <use-mesh name="palabosMesh" provide="yes"/>
            <read-data name="Temperature" mesh="palabosMesh"/>
            <mapping:petrbf-compact-polynomial-c0 support-radius="0.0025"
                constraint="consistent"
              direction="read" from="openfoamMesh" timing="initial"
                  to="palabosMesh"/>
        </participant>

        <m2n:sockets distribution-type="gather-scatter" from="openfoamSolver"
            to="palabosSolver"/>

        <coupling-scheme:serial-explicit>
            <timestep-length value="0.0028" method="fixed"/>
```

```
37              <max-time value="600"/>
38              <participants  first ="palabosSolver" second="openfoamSolver"/>
39              <exchange data="Temperature" mesh="openfoamMesh"
                    from="openfoamSolver" to="palabosSolver" initialize="yes" />
40          </coupling-scheme:serial-explicit>
41
42      </solver-interface>
43
44  </precice-configuration>
```

Here the preCICE configuration file for the simulations performed in this work is shown. `<solver-interface dimensions="3">` contains the specifications for the coupled three dimensional simulation. `<data:scalar name="Temperature"/>` specifies the data, that will be mapped from solver to solver. In this case it is the scalar field of the temperature. `<mesh>` specifies the meshes of the participating solvers and the data fields that are stored on the meshes. In this case it is the meshes of openFOAM and palabos. The temperature field is stored on those meshes. `<participant>` defines the participating solvers and how they handle the meshes and data. Here the data on the openFOAM mesh is defined to be passed to the receiving palabos solver. The openFOAM mesh is mapped onto the palabos mesh and the data is read by palabos. `<mapping:petrbf-compact-polynomial-c0>` defines the parameters for the radial basis function mapping. `<m2n:sockets>` specifies the data transmission between processes. `<coupling-scheme:serial-explicit>` defines the coupling scheme that is used. For the simulations in this work serial explicit coupling was used (see section 3.3.2). Here the global preCICE time step, which is the same as the largest time step of the participating solvers (in this case openFOAM), the runtime of the simulation, the participating solvers and data fields and meshes, that are mapped, are specified.