ROYAL INSTITUTE
OF TECHNOLOGY

MASTER OF SCIENCE THESIS

Modeling of Subcooled Nucleate Boiling with OpenFOAM

Edouard MICHTA

Supervisors:
Henryk ANGLART
Kristian ANGELE

Division of Nuclear Reactor Technology
Royal Institute of Technology
Stockholm, Sweden, February 2011

TRITA-FYS 2011:11 ISSN 0280-316X ISRN KTH/FYS/–11:10–SE

ABSTRACT

Within the course of this master thesis project, subcooled nucleate boiling in a vertical pipe has been modeled using CFD. The modeling has been carried out within the OpenFOAM framework and a two-phase Eulerian approach has been chosen. The code can be used to predict the distribution of the local flow parameters, i.e. the void fraction, the bubble diameter, the velocity of both liquid and gas, the turbulent intensity as well as the liquid temperature. Special attention has been devoted to the phenomena which govern the void fraction distribution in the radial direction. Two different solvers have been implemented and the simulations have been performed in two dimensions.

Firstly, isothermal turbulent bubbly flow is mechanistically modeled in a solver named *myTwoPhaseEulerFoamAdiabatic*. The conservation equations of mass and momentum are solved for the two phases, taking special care in the modeling of the interfacial forces. The turbulence phenomena are described by a classical k-ϵ model in combination with standard wall functions for the near-wall treatment. Furthermore, an interfacial area concentration equation is solved and two different models for its sink- and source terms (corresponding to bubble coalesence and bubble breakup) have been investigated.

Secondly, a solver named *myTwoPhaseEulerFoamBoiling* has been developed based on the first solver in order to model a heated wall leading to subcooled nucleate boiling and subsequent condensation in the subcooled liquid. Additional terms accounting for the phase change have been included in the mass and momentum conservation equations as well as in the interfacial area equation. Assuming the gas phase being at saturation conditions, only one energy equation for the liquid phase needs to be solved.

The adiabatic solver has been validated against the DEDALE experiment and the simulation results showed satisfactory agreement with the measured data. The predictions obtained from *myTwoPhaseEulerFoamBoiling* have been compared to the DEBORA experimental data base. They are qualitatively similar but rather high quantitative discrepancies exist. Grid dependence tests revealed that the latter solver depends on the near-wall grid resolution, a yet unresolved issue related to the application of the wall heat flux as the boundary condition. However, the results were shown to be insensitive to small variations in the applied inlet conditions.

ACKNOWLEDGEMENTS

Summarising a 18-month experience in one single page is impossible. There are too many people I would like to thank for having helped me in my work or shared very nice moments with. Long story short...

First of all, I would like to thank Henryk Anglart for having continuously helped and guided me throughout this thesis. I am also very grateful to all the colleagues at the Nuclear Reactor Technology department for the nice atmosphere I discovered there, as well as the interesting discussions we could have during our bi-weekly meeting. *Vielen Dank* especially to Roman for having standed me coming to his office every hour to ask him silly questions rearding OpenFOAM or Linux, but also for these great pingis nights I will miss so much and for the camels...

Furthermore, I am indebted to Vattenfall for the financial support but also for having welcomed me so warmly at their facility in Älvkarleby. Particularly, I would like to express my gratitude to Kristian Angele for his constant availability and for having given me valuable input or feedback in my work. The friendly and respectful relation we have had is probably one of the main reasons of the successfull outcome of this project.

In addition, I had a great year living almost 24/7 with Hervé, Marc, Mathias and Pauline (who by the way saved us from eating a Paneng every day...) and with whom I did so many memorable trips and organised an unforgettable corridor party! Also I must mention all the nukes for the year and a half we have spent together at Albanova, so Fredrik, Luca, Marti, Paul and all the others, I hope to see you soon again! As well I want to thank all my friends at Supaero, and I would like to recall this unique experience we shared with a few of them on a boat to Tallinn... A big *Thank ya!* to the 9BD crew, notably to Bastien, Bastien, JB, Alexane and Marc, for all the amazing nights quite often ending at Max... Besides, I am also very glad having been a team member of *Los Cachondos*, so thanks guys for the games, the beers and my first title in Sweden! Still about football, I will always remember this match with PE, Val and Alexandre played around 4 am, I am pretty sure you will never forget it neither... I am also very glad I got to know Eline better in the last months of my stay in Sweden, she will certainly remain a friend for long. There are still too many people I have not cited: my awesome roomie Johan of course but also Märta-Linn, Ludvig, Julia, Josefine and all the others I cannot mention for brevity reasons.

Finally, I want above all to thank my family, who has always supported my choices and has given me this unique opportunity to discover such a beautiful country that is Sweden and that I fell in love with.

CONTENTS

NOMENCLATURE

Symbol	Dimensions	Description
		Latin Symbols
a_i	m^{-1}	Interfacial area concentration
A_q	-	Quenching wall area ratio
$A_w^{'''}$	m^{-1}	Contact area with the wall per unit volume
C_D	-	Drag coefficient
C_f	-	Fanning friction factor $C_f = \frac{\tau}{\frac{1}{2}\rho_l U_l^2}$
C_h	-	Stanton number $C_h = \frac{Nu}{RePr}$
C_L	-	Lift coefficient
C_t	-	Turbulence response coefficient
c_p	$J/kg/K = m^2/s^2/K$	Specific heat capacity
C_{WL}	-	Wall lubrication coefficient
D	m	Pipe diameter
D_S	m	Bubble Sauter mean diameter
d_w	m	Bubble departure diameter
Eo	-	Eötvös number $Eo = \frac{g(\rho_l - \rho_g)D_s^2}{\sigma}$
f	$Hz = s^{-1}$	Detachment frequency
$\vec{g}$	m/s^2	Gravity
G	$kg/m/s^3$	Buoyant production of turbulent kinetic energy per unit volume
h_c	$W/m^2/K = kg/s^3/K$	Heat transfer coefficient for single-phase convection
h_{li}	$W/m^2/K = kg/s^3/K$	Heat transfer coefficient for condensation
i	J/kg	Specific enthalpy
$\bar{\bar{I}}$	-	Identity tensor
i_{fg}	$J/kg = m^2/s^2$	Latent heat
k	m^2/s^2	Turbulent kinetic energy
$\vec{M}$	$N/m^3 = kg/m^2/s^2$	Interfacial forces per unit volume
$\vec{n}_r$	-	Wall normal vector
$N^{''}$	m^{-2}	Nucleation site density
p	Pa	Pressure
Pr	-	Prandtl number $Pr = \frac{c_p \mu}{\lambda}$
$\vec{q}$	$W/m^2 = kg/s^3$	Conductive heat flux density
$q_w^{''}$	$W/m^2 = kg/s^3$	Wall heat flux density
$\bar{\bar{R}}$	$N/m^2 = kg/m/s^2$	Stress tensor
Re_b	-	Bubble Reynolds number $Re_b = \frac{\lVert\vec{U}_g - \vec{U}_l\rVert D_S}{\nu_l}$
t	s	Time
T	K or $°C$	Temperature
$\vec{U}$	m/s	Velocity

Continued on next page

Continued from previous page

Symbol	Dimensions	Description
$U^{'}$	m/s	Root mean square of velocity fluctuations
U^{+}	-	Dimensionless velocity $U^{+} = \frac{U_l}{\sqrt{\frac{\tau_w}{\rho_l}}}$
We	-	Weber number $We = \frac{\rho_l U_l^{'} D_S}{\sigma}$
x	m	Distance from the wall
x^{+}	m	Dimensionless distance from the wall $x^{+} = \frac{x\sqrt{\frac{\tau_w}{\rho_l}}}{\nu_l}$
y/D	-	Elevation to pipe diameter ratio
	Greek Symbols	
α	-	Phase volume fraction
Γ_{gl}	$kg/m^3/s$	Evaporation rate per unit volume
Γ_{lg}	$kg/m^3/s$	Condensation rate per unit volume
ϵ	m^2/s^3	Turbulent energy dissipation
κ	-	von Karman constant $\kappa = 0.42$
λ	$W/m/K = kg \cdot m/s^3/K$	Thermal conductivity
μ	$Pa \cdot s = kg/m/s$	Dynamic viscosity
μ^*	-	Dimensionless dynamic viscosity
ν	m^2/s	Kinematic viscosity
ρ	kg/m^3	Density
σ	kg/s^2	Surface tension
τ	$N/m^2 = kg/m/s^2$	Shear stress
ϕ	m^{-3}	Source/sink of bubble number density
	Subscripts	
$_{cr}$	Critical	
$_{g}$	Gas	
$_{i}$	Interface	
$_{l}$	Liquid	
$_{m}$	Mixture	
$_{max}$	Maximum	
$_{sat}$	Saturation	
$_{w}$	Wall	
	Superscripts	
^{BK}	Breakup	
^{bulk}	Bulk	
^{c}	Convection	
^{CO}	Coalescence	
^{D}	Drag	
^{e}	Evaporation	
$^{k-\epsilon}$	From the k-ϵ model	
^{L}	Lift	
^{NUC}	Nucleation	
^{q}	Quenching	
^{t}	Turbulent	
^{T}	Tranpose	
^{TD}	Turbulent dispersion	
^{VM}	Virtual mass	
^{WL}	Wall lubrication	

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Background

The increasing world population requires to augment the global energy production. That has become one of the biggest challenges in today's society. In the context of fossil fuel shortage and global warming, the need of sustainable, economical and safe technology has arisen. Renewable technologies such as wind power, photovoltaic cells or geothermal energy for example are of interest but some of them have to be further developed. Nuclear power appears to be one of the most mature solutions in order to fulfill the above criteria.

Even though nuclear power sometimes still faces the public opinion's reluctancy, most countries have admitted that it will represent an important share also in tomorrow's energy supply and they want to take part to what is by some experts called "the nuclear renaissance ". For instance, Sweden revised a previous law forbidding new build and a planned complete phase-out by 2010 and authorized the replacement of current reactors with new reactors.

Among the existing types of reactors, Light Water Reactors (LWR) are the most commonly used, particularly Boiling Water Reactors (BWR) and Pressurized Water Reactors (PWR). A basic knowledge concerning these designs is a pre-requisite for this work but out of purpose in this report.

1.2 Subcooled nucleate boiling

Convective boiling in vertical heated channels has many industrial applications, including the core of LWR where heat is supplied by the nuclear reactions in the fuel. This is a very complex phenomenon which can be divided into several regimes, depending on the local flow conditions, as shown in Figure 1.1. In the case of a LWR, where water is subcooled at the inlet,the first stage in the boiling process is called subcooled nucleate boiling. Before subcooled nucleate boiling occurs, the heat transfer is governed by single-phase forced convection. Subcooled nucleate boiling designates the evaporation of liquid in micro-cavities adjacent to the heated wall, also called nucleation sites, while the bulk liquid remains subcooled. When the wall temperature exceeds sufficiently the local saturation temperature, liquid evaporates and bubbles are formed at the nucleation sites. They grow until they reach a critical size and then detach from the heated wall. The bubbles migrate to the bulk of the fluid, which is subcooled, and condensate, heating the liquid phase. Subcooled boiling is said to be partial if the nucleation sites cover only a part of the heated wall. As the wall temperature increases, the area where single-phase convection occurs decreases and the area covered by nucleation sites increases. When nucleation sites cover the whole surface, the regime is said to be fully-developed.

Modeling of subcooled nucleate boiling is a first attempt towards handling simulations of the boiling crisis, also known as Critical Heat Flux (CHF). At a heat flux equal to the CHF, there is a very sudden deterioration of the heat transfer coefficient, leading to a sharp increase of the wall temperature since the heat flux imposed by fission reactions in the fuel remains unchanged. Therefore, the fuel rods structure may be damaged and radioactive material may be released to the primary system, which constitutes a serious infringement to the

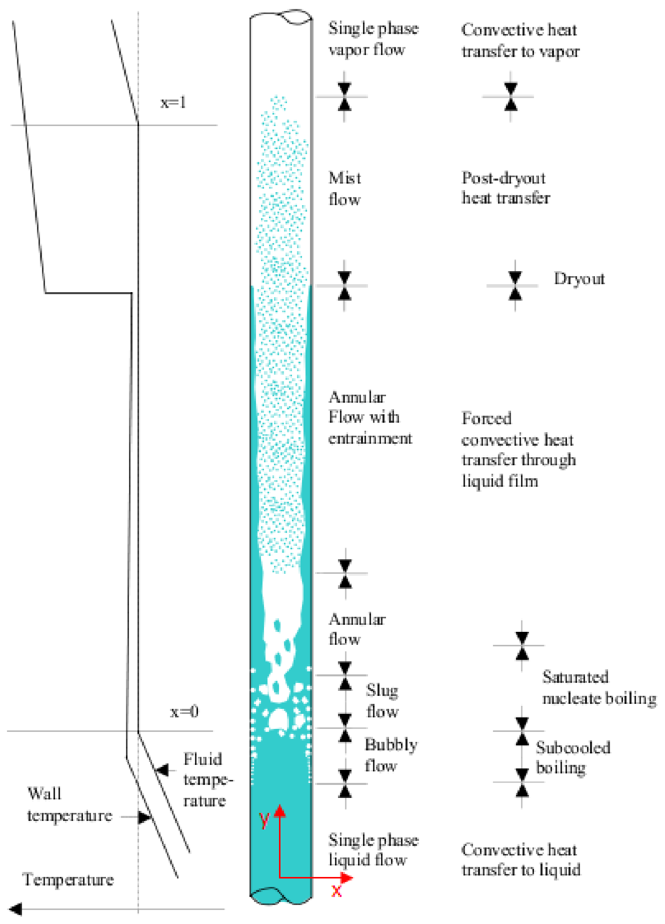
Single phase vapor flow
Convective heat transfer to vapor
x=1
Mist flow
Post-dryout heat transfer
Dryout
Annular Flow with entrainment
Forced convective heat transfer through liquid film
Annular flow
Saturated nucleate boiling
Slug flow
x=0
Bubbly flow
Subcooled boiling
Fluid tempe-rature
Wall temperature
y
Single phase liquid flow
Convective heat transfer to liquid
x
Temperature

Figure 1.1: Flow patterns and heat transfer regimes in a heated channel, from Anglart [1]

defense-in-depth principle. At low thermodynamic equilibrium qualities, which is the in a PWR core or the lower part of a BWR core, the CHF occurs when the boiling mechanism evolves from nucleate boiling to film boiling which is referred to as Departure from Nucleate Boiling (DNB), and represented by the point C in Figure 1.2. DNB sets an upper limit to the power that can be generated by the fuel.

In the case of PWR, even if this limit seems of no concern during normal operation, it is essential to ensure that safety margins will also be respected in case of transients, incidents or accidents. Indeed, subcooled boiling can even occur in the higher part of a PWR core, and it would generate a lithium borate deposition which affects significantly the power distribution. This phenomenon is called axial offset anomaly (AOA). The nuclear safety analysis of BWR is also influenced by the system reactivity. Subcooled nucleate boiling causes a highly inhomogeneous void fraction distribution in the axial direction (along the flow) since the channel is heated, but also in the cross-sectional direction due to the migration of steam bubbles. The presence of bubbles and their distribution induces a non-negligible reactivity feedback. Consequently, it is crucial to be able to predict the void fraction distribution if one wants to couple thermal-hydraulics and neutronics. The codes used today for this purpose are still old system level codes with a quasi one-dimensional representation of the flow and heat transfer in the core. Clearly there is a need for advanced models of the complex physics.

1.3 Computational fluid dynamics (CFD)

Reproducing the flow conditions of a power plant in an experimental facility is very difficult and expensive. The development of Computational Fluid Dynamics (CFD) together with the evolution of modern powerful computers the last decades has been a huge breakthrough within the industrial community. One can nowadays simulate many (but not all!) non-trivial fluid dynamics problems through the modeling of heat transfer, turbulence or chemical reactions at a varying degree of accuracy. Still, such models always need to be validated against experimental data. Any CFD software consists of three parts :

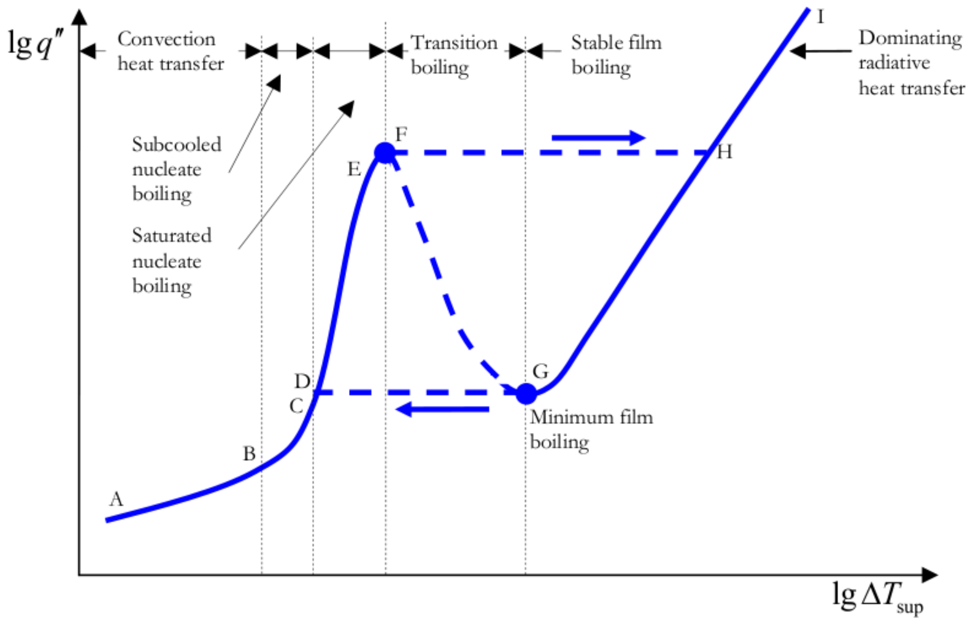

Figure 1.2: Boiling curve, influence of the wall heat flux on the wall superheat, from Anglart [1]

- **Pre-processing**: the geometry is defined and divided into discrete cells (the mesh), the initial and boundary conditions as well as physical properties of the fluid are set up
- **Solver**: the governing equations are discretized (both in time and space) and solved applying an iterative process
- **Post-processing**: the results are displayed thanks to various user-friendly visual tools

OpenFOAM ® (OF) is one of many existing CFD software packages available on the market. Its creation was initiated at the Imperial College in London in the late 1980's and the first version was released in 2004. OF is provided with many discretization schemes and a certain number of predefined solvers that can be applied for various fluid dynamics problems. OF has many advantages:

- **object-oriented**: one can select solvers, reflecting the physics one wants to model, independantly from discretization schemes or cases on which to apply the solvers. This feature gives OF en enhanced flexibility.
- **open-source**: the code is transparent and one can easily add or modify equations to create or improve solvers. This characteritic makes OF very interesting from an academic and research point of view.
- **free of charge**: there is no license restriction and companies can use OF without paying any fee.

1.4 Objectives of the work

The present study aims to develop a new solver for subcooled nucleate boiling within the OpenFOAM framework. A multi-dimensional modeling using a two-fluid Eulerian approach has been chosen. This work comports two main development stages. The first one is dedicated to the development of an adiabatic solver without phase change *myTwoPhaseEulerFoamAdiabatic*, and after validation of the latter, it has been further developed to account for boiling and subsequent condensation. This last version constitutes a new solver called *myTwoPhaseEulerFoamBoiling*. In addition to standard conservation equations, a one-group interfacial area concentration transport equation has been implemented and for *myTwoPhaseEulerFoamBoiling* boiling has been modeled according to Kurul and Podowski [19]. The solvers will be respectively tested and validated against adiabatic and diabatic experiments of vertical upward flows in cylindrical pipes.

These solvers should be taken as a basis for further implementation. They constitutes a skeleton for subcooled nucleate boiling modeling in OpenFOAM but do not intend to be complete and sophisticated. The solvers are based on several assumptions, notably of **incompressible phases, saturated gas phase, spherical bubbles and no bubble-sliding**. The closure relations present here, even if widely used in many software packages, remain perfectible. But since OF is an open-source code, more complex and accurate models could be easily implemented at a later stage.

Regarding the structure of this report, Chapter 2 describes the mathematical model. Chapter 3 presents briefly how OF works and describes its structure and then finally the code is described as well as the details of the tested cases. Chapter 4 deals with the results and the validation of the solvers. In Chapter 5, a discussion about possible future improvements is conducted. Last, in Chapter 5, the conclusions of this work are presented.

CHAPTER 2 MATHEMATICAL MODEL DESCRIPTION

This chapter is dedicated to a thorough description of the equations used in the solver. It begins with general conservation equations valid for a two-phase Eulerian modeling, and continues with the successive additional equations neccesary for the closure of the mathematical system of equations.

2.1 Governing flow equations

The code solves successively the different conservation equations (mass, momentum, energy), assuming incompressibility of each of the phases. In this study, the continuous phase is liquid and the dispersed phase is gaseous. The continuity equation can be written as:

$$\frac{\partial \alpha_k}{\partial t} + \nabla \cdot \left(\alpha_k \vec{U}_k \right) = \frac{\Gamma_{ki} - \Gamma_{ik}}{\rho_k} \tag{2.1}$$

where k denotes the phase and can be either l or g (liquid or gas) and i is the non-k phase. α, ρ and $\vec{U}$ represent the phase volume fraction, density and velocity respectively. The notation Γ_{gl} corresponds to the evaporation rate per unit volume, which within the scope of this study is due to nucleation at the wall, whereas Γ_{lg} is the condensation rate per unit volume, which is the only possible phase change at a phase interface since water remains subcooled and gas is saturated. This equation may be surprising because physically, condensation and evaporation can not occur at the exact same location. But in numerical calculations, it is possible to have a near-wall cell which contains the two phases and in which could occur both condensation of the gas and evaporation of a part of the liquid phase.

One can also observe that it is not required to solve the continuity equation for both phases. It is sufficient to solve one of them, for example for the gas phase, and calculate the liquid fraction according to:

$$\alpha_l = 1 - \alpha \tag{2.2}$$

where $\alpha = \alpha_g$.

The momentum equation for phase k is given by:

$$\frac{\partial \alpha_k \vec{U}_k}{\partial t} + \nabla \cdot \left(\alpha_k \vec{U}_k \vec{U}_k \right) = -\nabla \cdot \left(\alpha_k \left(\bar{\bar{R}}_k + \bar{\bar{R}}_k^t \right) \right) - \frac{\alpha_k}{\rho_k} \vec{\nabla} p + \alpha_k \vec{g} + \frac{\vec{M}_k}{\rho_k} + \frac{\Gamma_{ki} \vec{U}_i - \Gamma_{ik} \vec{U}_k}{\rho_k} \tag{2.3}$$

In Eq.2.3, the first term of the right hand side (RHS) is the combined (or effective) viscous and Reynolds (or so-called turbulent) stress. The second term of the RHS corresponds to the pressure drop in the channel. The third term of the RHS is gravity. The fourth term is the interfacial force per unit volume and will be the object of a specific modeling. Finally, the last term represents the gain or loss of momentum due to phase change. Incidentally, the latter has been seen too many times wrongly derived in the litterature in the form of various extravagant combinations of phase change rates and velocities.

According to Boussinesq [6], the turbulent stress tensor is analogous to that of Newtonian fluids and the resulting effective stress appears as a function of fluid properties and velocity:

$$\bar{\bar{R}}_k^{eff} = \bar{\bar{R}}_k + \bar{\bar{R}}_k^t = -\left(\nu_k + \nu_k^t\right)\left(\nabla\vec{U}_k + \left(\nabla\vec{U}_k\right)^T - \frac{2}{3}\bar{\bar{I}}\nabla\cdot\vec{U}_k\right) + \frac{2}{3}\bar{\bar{I}}k_k \tag{2.4}$$

where the identity tensor is identified as $\bar{\bar{I}}$. k_k designates the turbulent kinetic energy of phase k and ν_t its turbulent kinematic viscosity. The calculation of turbulent parameters will be explained later.

The last conservation equation concerns energy. As mentioned previously, one of the assumptions is that the gas is always at saturated conditions. It would be irrelevant to solve an energy equation for the gaseous phase. Only one energy equation is required for the liquid phase. In comparison with heat flux densities considered, potential and kinetic energies of the interface can be neglected. Also, for bubbly two-phase flows, the only possible heat transfer from the gas to the liquid is by condensation. These assumptions are justified by Kurul and Podowski [19]. Then, the energy conservation equation written in terms of specific enthalpy takes the following form:

$$\frac{\partial\left(\left(1-\alpha\right)i_l\right)}{\partial t} + \nabla\cdot\left(\left(1-\alpha\right)i_l\vec{U}_l\right) = -\frac{1}{\rho_l}\nabla\cdot\left[\left(1-\alpha\right)\left(\vec{q}_l + \vec{q}_l^t\right)\right] + \frac{\left(1-\alpha\right)}{\rho_l}\frac{Dp}{Dt} + \frac{\Gamma_{lg}i_{g,sat} - \Gamma_{gl}i_l}{\rho_l} + \frac{q_w^{''}A_w^{'''}}{\rho_l} \tag{2.5}$$

where $\vec{q}_l$ and $\vec{q}_l^t$ denote respectively the molecular and turbulent heat fluxes inside phase l. Here, the notation $\frac{D}{Dt}$ represents the total derivative. The third term is due to phase change and the last one represents the heat from the wall where $q_w^{''}$ is the wall heat flux density and $A_w^{'''}$ refers to the contact area with the wall per unit volume.

Now, if one uses Fourier's law of conduction inside phase l, the molecular heat flux can be tramsformed to:

$$\vec{q}_l = -\frac{\lambda_l}{c_{pl}}\vec{\nabla}i_l \tag{2.6}$$

where λ and c_p are the thermal conductivity and the specific heat respectively.

A similar expression can be written for the turbulent energy exchange if one uses a mixing length model :

$$\vec{q}_l^t = -\frac{\lambda_l^t}{c_{pl}}\vec{\nabla}i_l \tag{2.7}$$

where λ^t is the turbulent thermal conductivity obtained as:

$$\lambda_l^t = \frac{c_{pl}\nu_l^t}{\rho_l Pr_l^t} \tag{2.8}$$

where Pr_l^t is the turbulent Prandtl number of phase l. There are several empirical correlations and analytical relations to determine it [19], [17], but it is always in the same range in the frame of this work, slightly below 1. A constant value of 0.9 has been chosen for Pr_l^t.

2.2 Interfacial forces

The interfacial forces acting on a bubble are caused by the liquid which surrounds it. So, in agreement with the third of Newton's law of motion, it can be written:

$$\vec{M}_g = -\vec{M}_l \tag{2.9}$$

It is very important to select adequate models describing this force. Very often, this force is assumed to be the sum of different contributions, each one of them corresponding to a separate physical phenomenon:

$$\vec{M}_g = \vec{M}_g^D + \vec{M}_g^L + \vec{M}_g^{WL} + \vec{M}_g^{TD} + \vec{M}_g^{VM} \tag{2.10}$$

2.2.1 Drag force

The first term $\vec{M}_g^D$ is the drag force. This force represents the resistance opposed to the motion of a bubble in the fluid. Its direction is therefore the opposite of the bubble's relative velocity. The drag depends, as one could expect, on the bubble's shape and size. Several formulations exist but the Ishii-Zuber correlation [2] has been chosen here, since it is valid for densely distributed fluid particles, such as bubbles:

$$\vec{M}_g^D = -\frac{3}{4}\frac{C_D}{D_S}\rho_l \alpha \|\vec{U}_g - \vec{U}_l\| \left(\vec{U}_g - \vec{U}_l\right) \tag{2.11}$$

where D_S is the Sauter diameter, equal to the actual bubble diameter since bubbles are assumed to be sphericle within this work, and C_D defined as:

$$C_D = \max\left(24\frac{1+0.15Re_{bm}^{0.687}}{Re_{bm}}, 0.44\right) \tag{2.12}$$

$$Re_{bm} = \frac{\rho_l \|\vec{U}_g - \vec{U}_l\| D_S}{\mu_m} \tag{2.13}$$

$$\mu_m = \mu_l \left(1 - \frac{\alpha}{\alpha_{max}}\right)^{-2.5\alpha_{max}\mu^*} \tag{2.14}$$

Here, α_{max} is the void fraction at maximum packing equal to 0.52 and μ^* is the dimensionless dynamical viscosity:

$$\mu^* = \frac{\mu_g + 0.4\mu_l}{\mu_g + \mu_l} \tag{2.15}$$

The Ishii-Zuber drag model presented above is valid for densely distributed sphericle fluid particles, with a void fraction $\alpha < 0.52$ and at the condition that $0.2 < Re_b < 100000$

2.2.2 Lift force

The second term $\vec{M}_g^L$ is the lift force which must be treated carefully since it plays an important role and has a large effect on the radial distribution of bubbles. The following general formulation was proposed by Drew and Lahey ([10]):

$$\vec{M}_g^L = C_L \rho_l \alpha \left(\vec{U}_g - \vec{U}_l\right) \wedge \left(\nabla \wedge \vec{U}_l\right) \tag{2.16}$$

where C_L is the lift coefficient which has been investigated by many authors. Here the Tomiyama model was chosen:

$$C_L = \begin{cases} \min\left[0.288\tanh\left(0.121Re_b\right), f\left(Eo_d\right)\right] & \text{if } Eo_d < 4 \\ f\left(Eo_d\right) & \text{if } 4 \le Eo_d \le 10 \\ -0.27 & \text{if } Eo_d > 10 \end{cases} \tag{2.17}$$

$$f\left(Eo_d\right) = 0.001509Eo_d^3 - 0.0159Eo_d^2 - 0.0204Eo_d + 0.474 \tag{2.18}$$

The involved quantities are the bubble Reynolds number Re_b and a modified Eötvös number (describing the ratio between buoyancy forces and those from surface tension) Eo_d for the maximum horizontal dimension of the bubble d_h, that is calculated using a correlation developed by Wellek et al. [36]:

$$Re_b = \frac{\|\vec{U}_g - \vec{U}_l\| D_S}{\nu_l} \tag{2.19}$$

$$d_h = D_s\left(1 + 0.163Eo^{0.757}\right)^{1/3} \tag{2.20}$$

$$Eo = \frac{g\left(\rho_l - \rho_g\right) D_s^2}{\sigma} \tag{2.21}$$

$$Eo_d = \frac{g\left(\rho_l - \rho_g\right) d_h^2}{\sigma} \tag{2.22}$$

This model is widely used and accepted, particularly because it captures the sign change of the lift force for big bubbles, as shown in Figure 2.1. Indeed small bubbles are pushed towards the wall, but above a critical bubble diameter, they are pushed in the other direction, towards the centerline of the channel. For an air/water system at normal conditions, this critical diameter is around 5.8 mm.

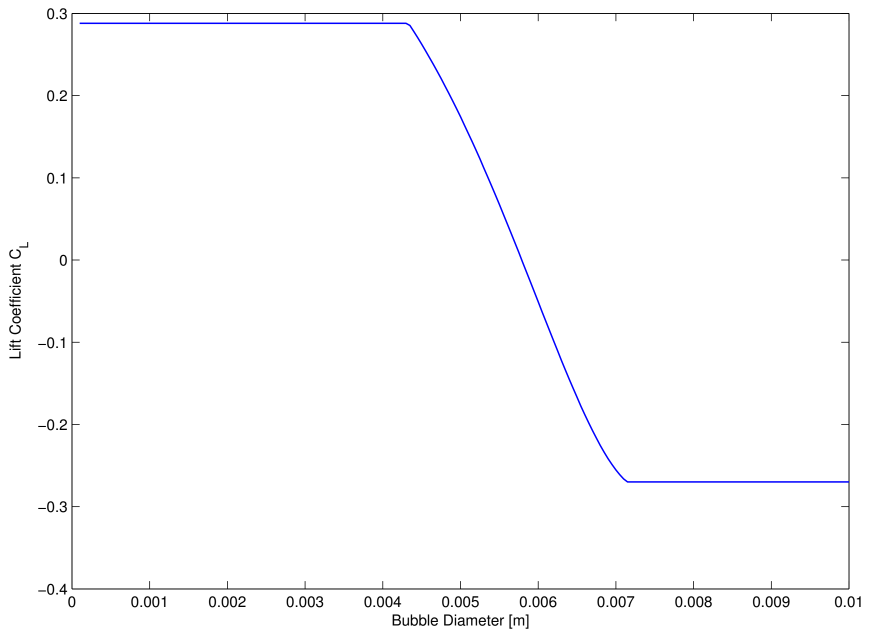

Figure 2.1: Influence of the bubble diameter on the Tomiyama lift coefficient for water at atmospheric pressure

2.2.3 Wall lubrication force

Furthermore, experiments on adiabatic two-phase flow showed evidence of the existence of a repulsive force pushing bubbles away from the wall. If bubbles were concentrated close to the wall, they were never adjacent to it. The force which is responsible for this wall peaking phenomenon is the so-called wall lubrication force, denoted by $\vec{M}_g^W$ in Eq. 2.10. Antal [3] was the first to derive an analytical expression of the wall lubrication force. Tomiyama improved this model for pipe geometries [33], using a correlation whose constants are based on glycerol/air flow experiments. However the model turned out to be valid for other two-phase systems such as air/water:

$$\vec{M}_g^{WL} = \frac{1}{2} C_{WL} \rho_l \alpha D_S \left(\frac{1}{x^2} - \frac{1}{(D-x)^2} \right) \|(\vec{U}_g - \vec{U}_l) \cdot \vec{y}\|^2 \vec{n}_r \tag{2.23}$$

$$C_{WL} = \begin{cases} 0.47 & \text{if } Eo < 1 \\ e^{-0.933Eo+0.179} & \text{if } 1 \leq Eo \leq 5 \\ 0.00599Eo - 0.0187 & \text{if } 5 < Eo < 33 \\ 0.179 & \text{if } 33 < Eo \end{cases} \tag{2.24}$$

where C_W is the wall lubrication force coefficient, D the pipe diameter, x the distance to the wall, $\vec{y}$ the axial direction and $\vec{n}_r$ the normal to the wall pointing towards the fluid. This model is however geometry-dependent and limited to pipes. Frank [11] developed a more general formulation based on Tomiyama wall lubrication force coefficient and a damping function independent of the geometry, but this improved model is not considered in the solver.

2.2.4 Turbulent dispersion force

The fourth component in in Eq. 2.10 is the turbulent dispersion force, referred to as $\vec{M}_g^{TD}$. The turbulent dispersion force accounts for the turbulent fluctuations of liquid velocity and the effect that has on the gas bubbles. Gosman et al. [12] proposed the following relation:

$$\vec{M}_g^{TD} = -\frac{3}{4} \frac{C_D}{D_S} \frac{\nu_l^t}{Pr_l^t} \rho_l \|(\vec{U}_g - \vec{U}_l)\| \vec{\nabla} \alpha \tag{2.25}$$

The turbulent dispersion plays a major role in the radial void fraction distribution. For small bubbles in adiabatic flow, one can observe the so-called wall-peaking phenomenon where most of the bubbles agglomerate close to the wall but not in contact with it. The turbulent dispersion force strongly influences the sharpness of this void wall peak, meaning its extent in the x-direction. For big bubbles whose lift force push them towards the centerline, the turbulent dispersion is the only force to limit the so-called core voiding which designates the agglomeration of bubbles around the centerline of the channel.

2.2.5 Virtual mass force

Finally the last force component in Eq. 2.10 is the virtual mass force, or added mass force, $\vec{M}_g^{VM}$. The virtual mass force takes its origin from the relative acceleration of one phase to the other. Therefore, this force is often neglected when running steady state calculations. It is considered in this study even though its contribution vanishes fast. According to Zuber ([40]), the virtual mass force for spherical bubbles is equal to:

$$\vec{M}_g^{VM} = -\frac{1}{2}\rho_l \frac{1+2\alpha}{1-\alpha}\alpha \left(\frac{D\vec{U}_g}{Dt} - \frac{D\vec{U}_l}{Dt} \right) \tag{2.26}$$

This force has been implemented in our solver so that the user can possibly use it. However, it has been found to be the origin of many instabilities and has therefore been neglected in Eq.2.10.

2.3 Boiling model

2.3.1 Wall heat flux partitioning

Over the past 50 years, different subcooled nucleate boiling models have been developed. On the one hand, one-dimensional empirical correlations, based on experiments, connect the wall heat flux and the wall superheat as a function of global parameters like pressure. One can for example refer to Jens-Lottes [16] or Thom et al. [31] correlations. Yet, these approaches do not lend themselves to CFD applications since they do not allow the calculation local parameters such as the evaporation rate and the bubble departure diameter.

On the other hand, mechanistic models based on local parameters (temperature, void fraction or for example turbulence intensity) could be used without any difficulty in CFD. One of them was presented by Kurul and Podowki [19] and it is the model applied in this solver. The basic idea in the model is that the heat transfer originates from three different mechanisms between the heated wall and the liquid phase:

$$q_w^{''} = q_w^{c''} + q_w^{e''} + q_w^{q''} \tag{2.27}$$

where $q_w^{c''}$, $q_w^{e''}$ and $q_w^{q''}$ denote respectively single-phase convective heat flux density, evaporation heat flux density and quenching heat flux density. A schematic view of the boiling process is shown in Figure 2.2. At the heated wall, bubbles are formed at the nucleation sites due to evaporation of liquid at the wall and one therefore assumes that part of the wall heat flux density is directly used to transform the liquid into vapor. Bubbles grow and reach a critical size at which they detach from the wall. Once a bubble leaves the wall, the volume previously occupied by it is filled with cooler liquid, which will receive heat from the wall. The heat transfer to this cooler liquid is called quenching. The ratio of the wall influenced by quenching A_q is thus closely depending on the evaporation and the bubble lift-off size. Finally, the rest of the wall $1 - A_q$ is governed by single-phase convection.

The evaporation heat flux can easily be calculated as a function of other key parameters: the nucleation site density $N^{''}$, the detachment frequency f and the bubble detachment (or departure or lift-off) diameter d_w. It is given by:

$$q_w^{e''} = \frac{\pi}{6} d_w^3 \rho_v f N^{''} \left(i_{g,sat} - i_l \right) \tag{2.28}$$

DelValle and Kenning [9] derived an analytical solution for the quenching heat flux, assuming a transient heat transfer in a liquid cylinder with a diameter equal to d_w. They showed, if one makes the hypothesis that the growth time is much shorter than the time between two subsequent nucleations, that:

$$A_q = \pi N'' d_w^2 \tag{2.29}$$

$$q_w^{q''} = A_q \frac{2\lambda_l (T_w - T_l)}{\sqrt{\frac{\pi}{f}\frac{\lambda_l}{\rho_l c_{pl}}}} \tag{2.30}$$

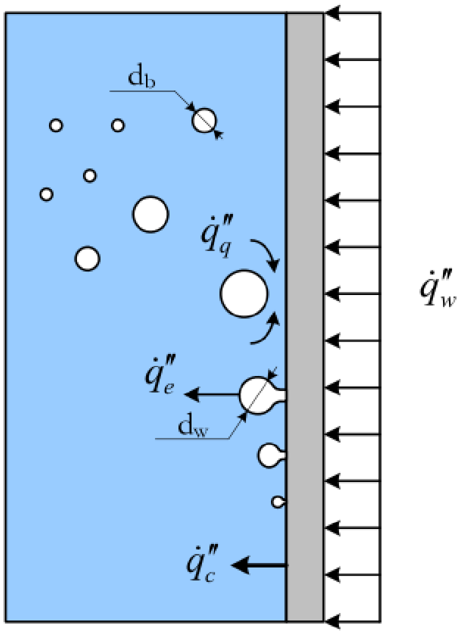

Figure 2.2: Heat flux partitioning according to Kurul and Podowski [34]

Concerning the single-phase convective heat flux, it is simply formulated by the following formula:

$$q_w^{c''} = (1 - A_q)h_c(T_w - T_l) \tag{2.31}$$

where h_c is the heat transfer coefficient for single-phase convection heat transfer. Kurul and Podowski [19] do not recommend a specific closure law for this coefficient, so it will be calculated as it has been done by KTH-NRT in the CFD-software CFX, using a approach based on the liquid Stanton number C_{hl}:

$$h_c = C_{hl}\rho_l c_{pl}\|\vec{U}_l\| \tag{2.32}$$

where C_{hl} is linked to the Fanning friction factor C_f through the following mono-dimensional correlation:

$$C_{hl} = \frac{C_f^2}{1 - 1.783C_f} \tag{2.33}$$

and the friction factor is implicitly suggested as:

$$C_f = \frac{1}{\frac{\ln(\widetilde{Re}C_f)}{0.435} + 5.05} \tag{2.34}$$

$$\widetilde{Re} = \frac{3.2586 \cdot 10^{-4}[m]\|\vec{U}_l^{bulk}\|}{\nu_l} \tag{2.35}$$

2.3.2 Nucleation site density

The nucleation site density depends mostly on the material properties of the wall and the wall superheat, which is the temperature difference between the wall temperature and the saturation temperature of the liquid. The data published by Lemmert and Chawla [20] leads to the following correlation:

$$N'' = [210\,(T_w - T_{sat})]^{1.805} \tag{2.36}$$

where T_{sat} is the saturation temperature. The temperatures must be expressed in K or $°C$ and the result obtained is given in m^{-2}.

2.3.3 Detachment frequency

During subcooled nucleate boiling, when approaching DNB, the growth of the bubbles is controlled by inertia effects and it can be analytically proven that $f^2 d_w$ is constant at constant heat flux. In particular, Ceumern-

Lindenstjerna [7] came to the result:

$$f = \sqrt{\frac{3}{4}\frac{(\rho_l - \rho_g)g}{\rho_l d_w}} \tag{2.37}$$

2.3.4 Bubble detachment diameter

The determination of the bubble detachment diameter is doubtlessly a key characteristic in modeling of subcooled nucleate boiling. Performing an energy balance for a bubble growing at a heated surface, and fitting some constants to experimental water data, Unal [35] expressed the bubble detachment diameter as:

$$d_w = \frac{2.42 \cdot 10^{-5} p^{0.705} a}{\sqrt{b\Phi}} \tag{2.38}$$

where:

$$a = \frac{(T_w - T_l)\lambda_l \gamma}{2\rho_g i_{fg}\sqrt{\pi \frac{\lambda_l}{\rho_l c_{pl}}}} \tag{2.39}$$

where i_{fg} is the latent heat and:

$$\gamma = \sqrt{\frac{\lambda_w \rho_w c_{pw}}{\lambda_l \rho_l c_{pl}}} \tag{2.40}$$

$$b = \frac{T_{sat} - T_l}{2\left(1 - \frac{\rho_g}{\rho_l}\right)} \tag{2.41}$$

$$\Phi = \begin{cases} \left(\frac{v_l^{bulk}}{0.61}\right)^{0.47} & \text{if } v_l^{bulk} > 0.61 \\ 1 & \text{if } v_l^{bulk} < 0.61 \end{cases} \tag{2.42}$$

and v_l^{bulk} represents the axial component of the liquid phase in the bulk. In the Unal correlation, all parameters must be given in the $S.I$, meaning that the pressure must be given in Pa, the velocity in m/s and the returned result for the bubble diameter is expressed in m.

But specific care must be taken while using this correlation since its range of validity is the following:

$$\begin{cases} 0.1 < p < 17.7Mpa \\ 0.47 < q_w < 10.64MW/m^2 \\ 0.08 < v_l^{bulk} < 9.15m/s \\ 3.0 < T_{sat} - T_l < 86K \\ 0.08 < d_w < 1.24mm \end{cases} \tag{2.43}$$

For low subcooling, other correlations must be used. At very low subcooling, below $2K$, one must switch to the Tolubinsky and Kostanchuk expression [32] which predicts an exponential decrease of the detachment diameter with the subcooling:

$$d_w = 1.4 \cdot 10^{-3} \exp\left(-\frac{T_{sat} - T_l}{45}\right) \tag{2.44}$$

Here again, all units are those from the $S.I.$

If the subcooling is between 2 K and 3 K, then the detachment diamater is calculated using a linear interpolation. In order to be able to use the same correlation for the whole range of subcooling, Borée et al. [5] modified the parameter b of the Unal correlation and defined it as a function of the Stanton number. However this improvement has not been taken into account in the present solver. It must also be recalled that the Unal correlation was established for water/vapor system only, but Manon [24] demonstrated that the results given by this correlation were in agreement with the experiments even when Freon 12 (R12) is used as coolant.

2.3.5 Wall superheat

Having a look a the boiling model that has been explained in the previous subsections, all variables only depend on physical properties and local parameters (velocity, temperature for both phases, pressure) which can be retrieved from the CFD software and the wall superheat, which is so far the only unknown in this boiling model.

The wall superheat is solved using a Newton-type iterative method. An initial guess is made for the wall superheat $T_w - T_{sat}$ and the different heat flux contributions are calculated, then the wall superheat is adjusted if Eq.2.27 is not fulfilled.

2.4 Interfacial mass transfer

2.4.1 Evaporation rate

Evaporation occurs at the heated wall only and can be deduced easily since the evaporation heat flux contribution is known from the previous section. The evaporation rate per unit volume is expressed as:

$$\Gamma_{gl} = \frac{q_w^{e''}}{i_{g,sat} - i_l} A_w^{'''} = \frac{\pi}{6} d_w^3 \rho_v f N^{''} A_w^{'''} \tag{2.45}$$

2.4.2 Condensation rate

Once bubbles are created and moves towards the centerline, they are surrounded by subcooled water and condense. In the case of subcooled nucleate boiling, condensation is indeed the only possible physical phase change at the interface for cells which are not in contact with the wall. The interfacial mass transfer related to condensation of vapor bubbles in the bulk coolant is found as:

$$\Gamma_{lg} = \frac{h_{li}(T_{sat} - T_l) a_i}{i_{fg}} \tag{2.46}$$

where a_i represents the interfacial area concentration and h_{li} the heat transfer coefficient for condensation at the interface described by Wolfert et al. [37], assuming that bubbles are at saturated conditions as:

$$h_{li} = \rho_l c_{pl} \sqrt{\frac{\pi}{4} \frac{\|\vec{U}_g - \vec{U}_l\|}{D_S} \frac{\lambda_l}{\rho_l c_{pl}} \frac{1}{1 + \lambda_l^t / \lambda_l}} \tag{2.47}$$

2.5 Interfacial area concentration transport equation

The interfacial area concentration (IAC) corresponds to the area of the gas bubbles per unit volume. For spherical bubbles, due to geometrical reasons, it is directly proportional to the void fraction and inversely proportional to the bubble diameter:

$$a_i = 6 \frac{\alpha}{D_S} \tag{2.48}$$

The IAC is influenced by many different phenomena. It can increase or decrease due to sudden phase change (condensation in the bulk or nucleation at the wall). Coalescence and breakup mechanisms also affect the IAC. Coalescence consists of several bubbles joining together to form a larger bubble, decreasing the area for the same volume of void so that the IAC decreases. This is provoked by collisions driven by turbulence. The opposite phenomenon is breakup of bubbles which results from the interaction between bubbles and turbulent eddies. In the case of breakup, one bubble splits into two or several smaller bubbles and the IAC increases as the consequence.

There is a spatial and temporal evolution of the IAC, which has to be described by an additional equation. Since most of the interfacial forces are functions of the bubble diameter, and indirectly of the IAC, this extra equation must be modeled accordingly. Yao and Morel ([39]) derived an analytical form:

$$\frac{\partial a_i}{\partial t} + \nabla \cdot \left(a_i \vec{U}_g \right) = -\frac{2}{3} \frac{a_i}{\alpha} \Gamma_{lg} + \frac{36\pi}{3} \left(\frac{\alpha}{a_i} \right)^2 \left(\phi_n^{CO} + \phi_n^{BK} \right) + \pi d_w^2 \phi_n^{NUC} \tag{2.49}$$

where ϕ_n^{CO} and ϕ_n^{BK} are a bubble sink term by coalescence and a bubble source term by breakup respectively and ϕ_n^{NUC} is the source term by nucleation at the wall, equal to:

$$\phi_n^{NUC} = fN^{''}A_w^{'''} \tag{2.50}$$

Many people have investigated these parameters. Wu et al. [38] and Ishii and Kim [14] derived expressions for these two terms and also introduced another term in Eq.2.49 reflecting the wake entrainment, meaning the acceleration of the following bubble in the wake of the preceding bubble. The Yao and Morel model [39] was chosen in the NEPTUNE code which has been implemented in the frame of a collaboration between the main french nuclear authorities and companies (CEA, Areva, EDF, IRSN). For a good description of the differences between the different models referred to here, see Delhaye [8]. In the present work, the Hibiki and Ishii model [13] is considered as the standard model in the present solvers. According to Hibiki and Ishii, the coalescence and breakup terms are found to be:

$$\phi_n^{CO} = -\Gamma_C \frac{\alpha^2 \epsilon_l^{1/3}}{D_S^{11/3}(\alpha_{max}-\alpha)} \exp\left(-K_C \frac{D_S^{11/3}\rho_l^{1/2}\epsilon_l^{1/3}}{\sigma^{1/2}}\right) \tag{2.51}$$

where:

$$\begin{cases} \Gamma_C = 0.005 \\ K_C = 1.29 \end{cases} \tag{2.52}$$

and,

$$\phi_n^{BK} = \Gamma_B \frac{\alpha(1-\alpha)\epsilon_l^{1/3}}{D_S^{11/3}(\alpha_{max}-\alpha)} \exp\left(-K_B \frac{\sigma}{D_S^{5/3}\rho_l \epsilon_l^{2/3}}\right) \tag{2.53}$$

where:

$$\begin{cases} \Gamma_B = 0.007 \\ K_B = 1.37 \end{cases} \tag{2.54}$$

However, the Yao and Morel model has also been created and can be selected if required by the user instead of the standard Hibiki and Ishii formulation. Yao and Morel derived the following expressions for the coalescence and breakup terms:

$$\phi_n^{CO} = -K_{C1} \frac{\alpha^2 \epsilon_l^{1/3}}{D_S^{11/3}} \frac{1}{\frac{\alpha_{max}^{1/3}-\alpha^{1/3}}{\alpha_{max}^{1/3}} + K_{C2}\alpha\sqrt{\frac{We}{We_{cr}}}} \exp\left(-K_{C3}\frac{We}{We_{cr}}\right) \tag{2.55}$$

where:

$$\begin{cases} K_{C1} = 2.86 \\ K_{C2} = 1.922 \\ K_{C3} = 1.017 \\ We_{cr} = 1.24 \end{cases} \tag{2.56}$$

and,

$$\phi_n^{BK} = K_{B1} \frac{\alpha\left(1-\alpha\right)\epsilon_l^{1/3}}{D_S^{11/3}} \frac{1}{1+K_{B2}\left(1-\alpha\right)\sqrt{\frac{We}{We_{cr}}}} \exp\left(-\frac{We_{cr}}{We}\right) \tag{2.57}$$

where:

$$\begin{cases} K_{B1} = 1.6 \\ K_{B2} = 0.42 \end{cases} \tag{2.58}$$

2.6 Turbulence modeling

2.6.1 Turbulence of the liquid phase

The effective viscosity of the liquid phase is the sum of a molecular and a turbulent viscosity.

$$\nu_l^{eff} = \nu_l + \nu_l^t \tag{2.59}$$

The turbulent behavior of the liquid phase is in turn decomposed in two different contributions:

- an inherent **shear-induced turbulence** modeled as a classical single-phase $k-\epsilon$ expression:

$$\nu_l^{t,k-\epsilon} = C_\mu \frac{k_l^2}{\epsilon_l} \tag{2.60}$$

  where the typical value of the constant C_μ is 0.09

- a **bubble-induced turbulence** described by Sato et al. [27]:

$$\nu_l^{t,b} = \frac{1}{2} C_{\mu b} D_S \alpha \|\vec{U}_g - \vec{U}_l\| \tag{2.61}$$

  where the value of the constant $C_{\mu b}$ is 1.2.

To summarize, the turbulent viscosity of the liquid phase is given by:

$$\nu_l^t = C_\mu \frac{k_l^2}{\epsilon_l} + \frac{1}{2} C_{\mu b} D_S \alpha \|\vec{U}_g - \vec{U}_l\| \tag{2.62}$$

2.6.2 Turbulence of the gas phase

The turbulence of the gas phase is assumed to be dependant on the liquid phase turbulence and a turbulence responce coefficient C_t, defined as the ratio of the root mean square values of dispersed phase velocity fluctuations to those of the continuous phase, given by:

$$C_t = \frac{U_g'}{U_l'} \tag{2.63}$$

This approach described by Rusche [26] presents the advantage not to introduce any extra differential equation. The effective viscosity of the gas phase is simply expressed as:

$$\nu_g^{eff} = \nu_g + C_t^2 \nu_l^t \tag{2.64}$$

Various authors have derived expressions to calculate C_t as a function of local parameters such as the void fraction, but in the present solvers, a constant value is set by the user. By default, the value used for the simulations is set to zero, and no turbulence is affected to the gas phase.

2.6.3 Near-wall treatment: wall functions

It is well known that even for turbulent flows, there is very close to the wall a viscous sub-layer where the flow is quasi-laminar. Further away from the wall, in the outer parts of the near-wall region, the flow is turbulent. Between these two zones, there is a transition called the buffer region. The velocity profile and the extent of the different regions is shown in Figure 2.3. It has to be pointed out that within the frame of this work, $\vec{x}$ and $\vec{y}$ denote respectively the wall normal direction and the axial direction.

Thus, if one wants to capture the behaviour of the velocity in CFD, the near-wall region needs a specific attention and must be treated carefully. One option is to refine the mesh in the vicinity of the wall so that the near-wall region is resolved. But for flows at high Reynolds number, this solution can be very computationally expensive. Therefore, another approach is to model the flow based on wall functions as described below.

The k-equation is solved in the whole domain, even for near-wall cells, which is not the case for the ϵ-equation. Instead ϵ is found close to the wall from:

$$\epsilon_l = \frac{C_\mu^{3/4} k_l^{3/2}}{\kappa x} \tag{2.65}$$

where κ is the von Karman constant equal to 0.42.

Once k is known, it is possible to derive the dimensionless distance to the wall x^+ from the relation:

$$x^+ = \frac{C_\mu^{1/4} k_l^{1/2}}{\nu_l} \cdot x \tag{2.66}$$

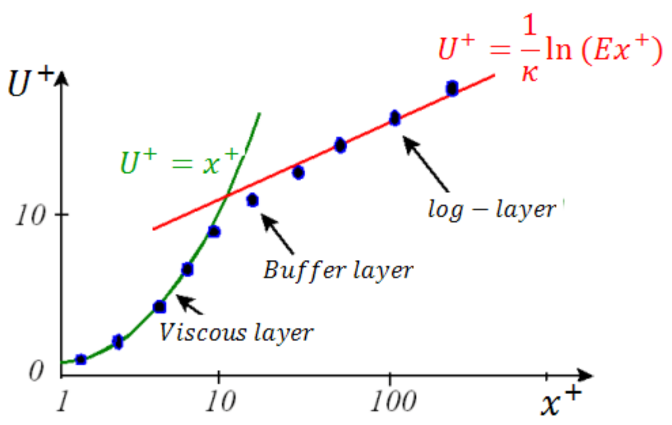

Figure 2.3: Different behaviors occuring in the near-wall region [29]

Depending on the value which is obtained, one will know in which layer the near-wall cells are and how to handle them. For $x^+ > 11.6$, it is assumed that the flow is in the turbulent region. Then the shear-induced turbulent viscosity of the liquid phase and the buoyant production term appearing in the k-ϵ equation are:

$$\nu_l^{t,k-\epsilon} = \nu_l \left(\frac{y\kappa}{\ln(Ex^+)} - 1 \right) \tag{2.67}$$

$$G_l^{t,k-\epsilon} = \frac{\nu_l^{t,k-\epsilon} \| \frac{\partial \vec{U}_l}{\partial x} \| C_\mu^{1/4} k_l^{1/2}}{\kappa x} \tag{2.68}$$

where E is a constant equal to 9.793.

On the opposite, if $x^+ < 11.6$, viscous effects dominate (viscous sub-layer) and it is assumed that there is no shear-induced turbulence:

$$\nu_l^{t,k-\epsilon} = G_l^{t,k-\epsilon} = 0 \tag{2.69}$$

Updating the fields for near-wall cells as detailed previously will implicitly apply the standard law of the wall for the velocity field of the liquid phase:

$$\begin{cases} U^+ = x^+ & \text{if } x^+ < 11.6 \\ U^+ = \frac{1}{\kappa}\ln(Ex^+) & \text{if } x^+ > 11.6 \end{cases} \tag{2.70}$$

2.7 Summary

The presented mathematical models contains 8 independent equations :

- 2 mass conservation equations for the liquid and gas phases
- 2 momentum conservation equations for the liquid and gas phases
- 1 energy conservation equation for the liquid phase
- 1 interfacial area concentration transport equation
- 2 additional partial derivative equations for the turbulence of the liquid phase

Using various closures laws, all the parameters appearing in these equations have been expressed as a function of 8 independent variables:

- the void fraction α
- the gas velocity $\vec{U}_g$

- the liquid velocity $\vec{U}_l$
- the specific enthalpy of the liquid phase i_l
- the interfacial are concentration a_i
- the turbulent kinetic energy of the liquid phase k_l
- the turbulent dissipation rate of the liquid phase ϵ_l
- the pressure p

The mathematical system of equations is properly defined and can be solved numerically.

CHAPTER 3 METHOD

In this chapter, the structure of the CFD software OpenFOAM 1.7.x is described and it is briefly explained how to use it. However, it is not intended to give a complete description of the software, this can be found in the User Guide [22] and the Programmer's Documentation [21]. Then this chapter focus on the code used in the solver, which will be analysed step-by-step. Lastly, the different validation cases, on which the solver will be applied, are presented.

3.1 Brief presentation of OpenFOAM

3.1.1 Overview

OpenFOAM is an object-oriented open-source CFD software package built on the C++ programing language. As a result, a basic knowledge of C++ is a pre-requisite and if needed the reader may refer to Stroustrup [28] on the C++ language. OpenFOAM is a complete tool using different libraries to solve complex fluid dynamics problems. The overall structure of OF is displayed in Figure 3.1. As it can be seen, the user has at his disposal many tools which are included in OF for pre-processing and solving, but OF does not include any specific post-processing software. The user must therefore install a post-processing software in order to analyse the data retrieved from OpenFOAM. ParaView ® is however the most commonly used too for this task and is adopted in this work.

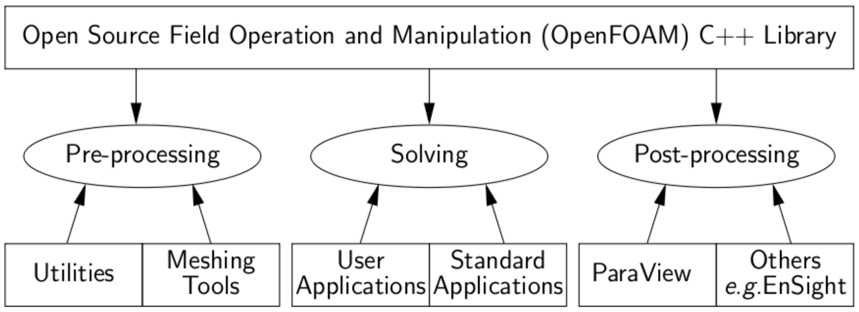

Figure 3.1: Overview of OpenFOAM structure [22]

Basically, there are two types of entities in OF. The first one is *applications*, which are executables, and can be either *solvers* or *utilities*. Solvers, as one can imagine, solve given continuum mechanics equations. Utilities are dedicated to data manipulation. Utilities are not essential in this study since they do not deal with the physics and will not be discussed futher in this report. The other type of entity is called *cases*. Both *solvers* and *cases* will be described more in depth in the following sections.

3.1.2 Solvers

Each solver represents a specific continuum mechanics problem to solve in which the governing equations are implemented. The source file of the solver must have the *.C*-extension and be placed, by convention, in a folder wearing the same name as the source file. This folder can also contain other files with the *.H*-extension which are "used" or referred to by the main source code. It is for example quite useful when the code is large, since it allows to separate the main code into several parts, which is of huge interest while debugging the code. Files with the *.H*-extension can also designate an already existing class used in the main source code. The last type of file included in the solver folder is another folder named *Make* which is essential for the compilation. *Make* is composed of two files: *files* and *options* which incorporate paths to the different files that must be compiled and the paths to the necessary libraries. For example, a solver called *newApp* could have the following structure:

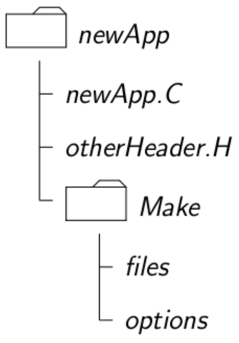

Figure 3.2: Directory structure for a solver [22]

As explained earlier, a solver contains the different equations to be solved. One of OpenFOAM's strengths is its equation representation, whose syntax is realy similar to the mathematical one. Considering for example the following transport equation for a field T:

$$\frac{\partial T}{\partial t} + \nabla \cdot \left(T\vec{U}\right) - \nabla \cdot \left(\nu \vec{\nabla} T\right) = 0 \tag{3.1}$$

would be written:

```
solve
(
   fvm::ddt(T)
 + fvm::div(phi, T)
 - fvm::laplacian(nu, T)
);
```

After the physical problem is properly put into equations in the solver, the latter must be compiled. This process is done very easily. In the Linux environment, one has only to open a terminal, go to the solver's directory and type in the command *wmake*. This will create the executable file that can be applied on cases.

3.1.3 Cases

If a solver contains the universal aspects of a physical problem, meaning the equations, a case on the contrary refers to the specificities of a problem (geometry, physical properties, computational schemes, etc). A case is a folder that is based on three main directories : *system*, *constant* and *0*.

The *system* directory usually contains three different files. The first file is *controlDict* and is used to set up for example the starting time of the simulation, the write interval, the end time, the time step as well as calculating the output data. The second file *fvSchemes* is for setting up the numerical discretisation schemes which are used for time and spatial derivatives. The third and last file is called *fvSolution*, it contains information associated with the solution procedure i.e. solvers to be used for the given equations, solution algorithm structure, relaxation and tolerances.

The *constant* directory is composed of one folder called *polyMesh* and several other files. The *polyMesh* folder includes the definition of the geometry. How to create a geometry will not be described in details here but the user may refer to [22]. Among the other files located in the *constant*, one finds for example the *transportProperties* file (which contains mostly physical properties of the two phases, pressure and constants), the *environmentalProperties* (including the wall heat flux and wall material properties), the *g* file (in which the gravity field is defined), the *interfacialProperties* file (for setting up the models chosen for the interfacial forces) and the *turbulenceProperties* (where the turbulence model and the associated constants are defined). Of course, the number of files and their nature depends on the case since the required information varies from one application to another, but the files described here are those normally used in the two-phase flow simulations considered here.

The *0* directory contains the initial and boundary conditions of all fields that have to be solved. There are few pre-defined initial or boundary conditions in OF that are quite often sufficient in common applications. They are described in the OpenFOAM User Guide [22].

Then, in order to run an specific solver (which must have been compiled before) on a case in the Linux environment, one has to open a terminal, go into the case folder and type the name of the compiled solver. This will run the code and additional directories containing the calculated fields will be created in the case directory. Their names are the time at which the results are written.

3.2 The solvers code explained

In this work, two different solvers have been implemented: one for adiabatic flows, *myTwoPhaseEulerFoamAdiabatic*, and one for heated flows, *myTwoPhaseEulerFoamBoiling*.

The solvers are improved versions of the *twoPhaseEulerFoam* solver which is already included in OF. Certain files have been removed because they were unnecessary for subcooled boiling, such as the kinetic theory model, but many others have been created or modified. **For instance**, the *phaseModel* class has been modified and many other interfacial forces have been implemented in the *interfacialModels* class (not only drag models as it was done so far). An energy equation written in terms of enthalpy has been introduced as well. The algorithm procedure is the so-called Pressure-Implicit with Splitting of Operatos (PISO) which is adapted for transient calculations. A graphical representation of the solution procedure used in *myTwoPhaseEulerFoamBoiling* can be seen in Figure 3.3.

These two solvers that have been created have a similar structure. *myTwoPhaseEulerFoamAdiabatic* differs from *myTwoPhaseEulerFoamBoiling* only through the headers *condensationModel.H*, *evaporationModel.H* and *HEqn.H* which are not present. The other headers are absolutely identical, except the fact that in *myTwoPhaseEulerFoamAdiabatic*, the phase change terms have been removed from the equations.

As can be noticed from Figure 3.3, all equations are set up in different header files with the *.H*-extension. The code of each of these files constituting *myTwoPhaseEulerFoamBoiling* will be described shortly within the following subsections. The line numbers appearing to the left side correspond to the number of the lines in the source code files, as they appear in the appendices.

3.2.1 condensationModel.H

First the magnitude of the relative velocity between the gas and the liquid phases is calculated. A field representing a very small bubble diameter is also created, which is added to the actual calculated bubble diameter. This feature aims to avoid a division by zero which would lead to numerical issues and make the program crash. The turbulent thermal conductivity is also defined and initiated, according to Eq.2.8.

```
01: Ur = Ua - Ub;
02: magUr = mag(Ur);
03:
04: dimensionedScalar plusDS
05:     (
06:         "plusDS",
07:         dimensionSet(0, 1, 0, 0, 0, 0, 0),
08:         scalar (1.0e-6)
09:     );
10:
11: volScalarField lambdatb = Cpb*rhob*nutb/PrandtlbT;
```

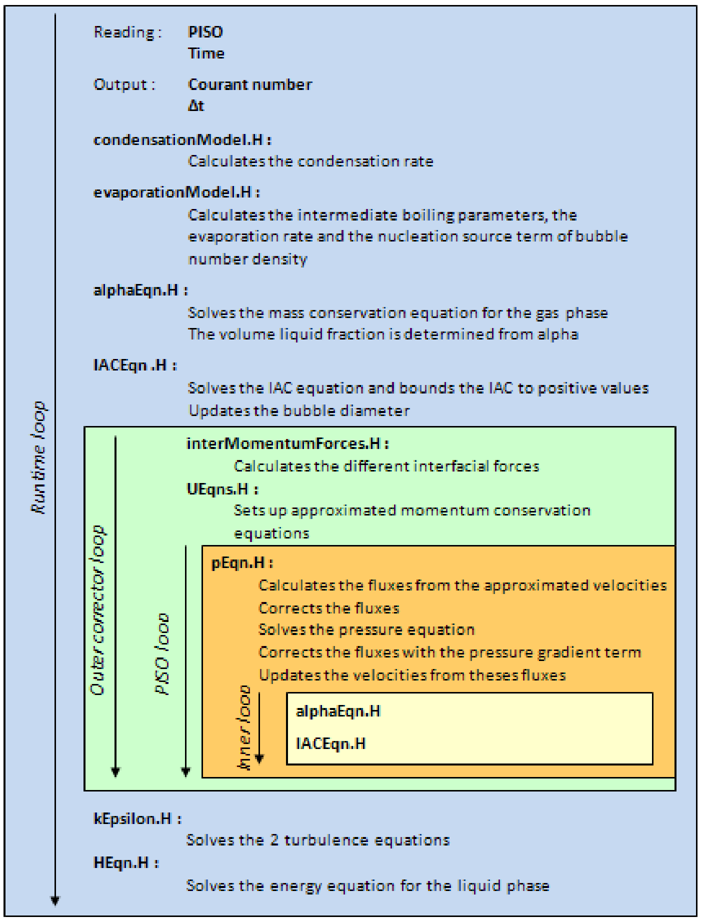
Runtime loop
Reading : PISO
Time
Output : Courant number
Δt
condensationModel.H :
Calculates the condensation rate
evaporationModel.H :
Calculates the intermediate boiling parameters, the evaporation rate and the nucleation source term of bubble number density
alphaEqn.H :
Solves the mass conservation equation for the gas phase
The volume liquid fraction is determined from alpha
IACEqn .H :
Solves the IAC equation and bounds the IAC to positive values
Updates the bubble diameter
Outer corrector loop
interMomentumForces.H :
Calculates the different interfacial forces
UEqns.H :
Sets up approximated momentum conservation equations
PISO loop
pEqn.H :
Calculates the fluxes from the approximated velocities
Corrects the fluxes
Solves the pressure equation
Corrects the fluxes with the pressure gradient term
Updates the velocities from theses fluxes
Inner loop
alphaEqn.H
IACEqn.H
kEpsilon.H :
Solves the 2 turbulence equations
HEqn.H :
Solves the energy equation for the liquid phase

Figure 3.3: Graphical representation of the solution procedure adopted for *myTwoPhaseEulerFoamBoiling* (inspired from Thiele [30])

Then, the heat transfer coefficient is calculated using Wolfert's correlation, according to Eq.2.47. Another correlation for condensation in the subcooled bulk, which is used in the NEPTUNE code instead of the Wolfert's correlation, has also been implemented but has not been used here. Then the condensation rate field is created from the heat transfer coefficient, cutting off potential unphysical negative values.

```
// Wolfert correlation: heat transfer coefficient for bubbles condensation in
// the subcooled bulk
volScalarField h = rhob*Cpb*sqrt(4.0/3.14159*magUr/(DS+plusDS)*lambdab
                    /(rhob*Cpb)*1.0/(1.0+lambdatb/lambdab));

volScalarField GammaLG
(
    IOobject
    (
            "GammaLG",
            runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    max(6.0*alpha/(DS+plusDS)*h*(Hbsat-Hb)/Cpb/hfg,
        0.0*alpha/(DS+plusDS)*h*(Hbsat-Hb)/Cpb/hfg)
);
```

3.2.2 evaporationModel.H

The *evaporationModel.H* leads to the determination of the wall heat flux partition and the evaporation rate can be calculated.

Even though all quantities are defined as *volScalarField* in the whole geometry, it is obvious that evaporation only occurs in near-wall cells. Therefore, a first step is to localize them and also the cell located at the same axial position in the bulk. For simplicity's sake, the bulk is assumed to be at the centerline. This is done in the following lines:

```
forAll(thePatchItselfWall,iFace)
{

    label iCell = thePatchItselfWall.faceCells()[iFace];
    vector iCellCentre = mesh.cellCentres()[iCell];

    //Point located at the center line but on the same axial position
    vector bulkPoint(0,iCellCentre.y(),iCellCentre.z());
    label bulkCell = mesh.findCell(bulkPoint);
```

Once the near-wall cells are localized, it is useful to calculate their dimensions. It is done in the following lines:

```
    // Needed to take the cell dimensions in order to check that the bubble are
    // are not too large for the mesh at the wall
    const cell & cc = mesh.cells()[iCell];
    labelList pLabels(cc.labels(ff));
    pointField pLocal(pLabels.size(), vector::zero);

    forAll (pLabels, pointi)
    {
        pLocal[pointi] = pp[pLabels[pointi]];
    }

    scalar xDim = Foam::max(pLocal & vector(1,0,0)) - Foam::min(pLocal & vector(1,0,0));
    scalar yDim = Foam::max(pLocal & vector(0,1,0)) - Foam::min(pLocal & vector(0,1,0));
    scalar zDim = Foam::max(pLocal & vector(0,0,1)) - Foam::min(pLocal & vector(0,0,1));

```

After that, an initial guess must be made for the wall superheat. This is done by assuming that the total heat flux density will be dedicated to single phase convection. However, the real heat transfer coefficient for single phase convection is not know yet since it is calculated from the Kurul and Podowski partitioning. Therefore, the first step is to calculate the Fanning friction factor according to a given correlation, and from factor to derive the Stanton number according to another correlation. These correlations are those implemented in the Fortan code developed by KTH-NRT in CFX ®. The original Fortran code can be found in the appendix. Using the definition of the Stanton number, one can finally calculate the heat transfer coefficient. This procedure to calculate an initial guess of the wall superheat is implemented as:

```
    for (int i=0; i<10; i++)
    {
        Cf[iCell] = 1.0/(Foam::log(Reyno.value()*Cf[iCell])/0.435+5.05);
    }

    // Stanton number
    Ch[iCell] = Cf[iCell]*Cf[iCell]/(1.0-1.783*Cf[iCell]);

    // Single-phase heat transfer coefficient
    H1F[iCell] = Ch[iCell]*rhob.value()*Cpb.value()*mag(Ub[bulkCell]);

    // Guess wall superheat
    Tsup[iCell] = qw.value()/H1F[iCell] - Tsub[iCell];
```

Then, an Newton-type iterative procedure is applied on the near-wall cells. The wall superheat is initially guessed assuming that the total heat flux is dedicated to single-phase convection. Depending on its value, several paths can be followed in the code. If it is below a very small value, meaning that there is no superheat, heat transfer is purely convective and all boiling parameters are set to zero and the *evaporationModel.H* is exited:

```
if (Tsup[iCell] <= 0.05)
{
    NSD[iCell] = 0.0;
    freq[iCell] = 0.0;
    Hq[iCell] = 0.0;
    A1F[iCell] = 1.0;
    goto label1;
}
```

Otherwise, several options are possible depending on the subcooling, as explained in the theory part. One will either use the Unal correlation at high subcooling, the Tolubinsky correlation at very low subcooling or a linear interpolation at low subcooling. The parameters described in the Kurul and Podowski model are calculated in an included header file named *nucleateBoilingModel.H*. After that, the wall superheat is updated if the total wall superheat is not equal to the sum of the three contributions (convection, queching, evaporation) as shown below:

```
Hh[iCell] = A1F[iCell]*H1F[iCell] + Aq[iCell]*Hq[iCell];

qtot[iCell] = evaporationSfRate[iCell]*Hlv[iCell] + Hh[iCell]*(Tsub[iCell]+Tsup[iCell]);

if (Hh[iCell] == 0.0)
{
    TsupCorr[iCell] = -Tsub[iCell];
}
else
{
    TsupCorr[iCell] = Tsup[iCell] + relax*(qw.value()-qtot[iCell])/Hh[iCell];

    if (TsupCorr[iCell] <= -Tsub[iCell])
    {
        TsupCorr[iCell] = -Tsub[iCell];
    }
}

error = abs(qtot[iCell]-qw.value())/qw.value();

if (TsupCorr[iCell] <= 0.0)
{
    TsupCorr[iCell] = 0.8*Tsup[iCell];
}

if (error > 0.00001)
{
    Tsup[iCell] = TsupCorr[iCell];
    goto label3;
}
```

Furthermore, within the included *nucleateBoilingModel.H* file, the solver checks that the bubble diameter at detachment is not larger than any of the near-wall cells dimensions. Otherwise, error messages will be displayed in the terminal for the user and the solver will be exited interrupting the calculations, as seen below:

```
//Check that the mesh is not too refined close to the wall in comparison with
```

```
//bubbles size at departure
if ( (dm.value()>xDim) || (dm.value()>yDim) || (dm.value()>zDim)  )
{
    Info<< "ERROR : The mesh is too refined in comparison with the bubble size at departure"<<  nl
<< endl;
    Info<< "IMPOSSIBLE to put all the energy of the created bubble in the near-wall cell "<<  nl <<
endl;

    return 1;
}
```

3.2.3 alphaEqn.H

Once the phase change rates are known, one can solve conservation equations, starting with the continuity equation for the gas phase. For numerical reasons, the form implemented is slightly different from what is shown in Eq.2.1 but the equation remains the same. Here, the conservation equation was divided by the void fraction and a mixture flux *phic* as well as a relative flux are present in the equation. The continuous phase fraction is last deduced from the dispersed one by substracting the former to one.

```
    fvScalarMatrix alphaEqn
    (
        fvm::ddt(alpha)
      + fvm::div(phic, alpha, scheme)
      + fvm::div(-fvc::flux(-phir, beta, schemer), alpha, schemer)
      ==
      (GammaGL-GammaLG)/rhoa
    );

    alphaEqn.relax();
    alphaEqn.solve();

    alpha = max(1.0e-8,alpha);

    beta = scalar(1) - alpha;
```

This loop can be repeated a predefined number of correction iterations, set in the *fvSolution* directory of the case by the user.

3.2.4 IACEqn.H

The next step treats the interfacial area concentration (IAC) transport equation, which is essential in order to determine the bubble diameter and the resulting interfacial forces. It begins with retrieving the sources terms by coalescence and breakup, respectively *coalescenceSource* and *breakupSource*, within the included header file *breakupAndCoalescence.H*. Then a field is created with a minimum value for the IAC to bound it to positive values:

```
# include "breakupAndCoalescence.H"

surfaceScalarField phiMix = phi;
surfaceScalarField phiRel = phia - phib;

dimensionedScalar smallIAC
(
    "smallIAC",
    dimensionSet(0, -1, 0, 0, 0, 0, 0),
    scalar (1.0)
);
```

Afterwards, the IAC transport equation is set up and solved, and since the void fraction is also known, the bubble diameter can be derived:

```
fvScalarMatrix IACEqn(IAC, IAC.dimensions()*dimVol/dimTime);

{
        IACEqn =
        (
            (
                fvm::ddt(IAC)
              + fvm::div(phiMix, IAC, "div(phiMix,IAC)")
              + fvm::div(-fvc::flux(-phiRel, beta, "div(phiRel,IAC)"), IAC, "div(phiRel,IAC)")
            )
              ==
                2.0/3.0*IAC/(alpha+scalar(1.0e-8))/rhoa*(-GammaLG)
              + 36.0*3.14/3.0*pow(alpha/IAC,2.0)*(coalescenceSource+breakupSource)
              + nucleationSource

        );

        IACEqn.relax();
        IACEqn.solve();
}

IAC = max(IAC,smallIAC);

DS = 6.0*alpha/IAC;
```

3.2.5 interMomentumForces.H

Before setting up the momentum equations, the different interfacial forces are defined in a separate file called *interMomentumForces.H*. The models used for each type of interfacial forces are previously specified by the user in the *constant* folder of the case, and read within *createFields.H*.The implementation of these models is not shown here but can be found in the appendix. *interMomentumForces.H* is pretty clear and does not require much explanation.

Recall simply that the implementation of the Tomiyama wall lubrication is highly dependant on the geometry. In this code, for example, the normal vector to the wall is defined as:

```
 //definition of the normal vector for 2D mesh
volVectorField normal
(
    IOobject
    (
        "normal",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    vector(-1,0,0)
);
```

which is valid for a pipe whose axis is given by $\vec{y}$ so this part of the code should be modified for other geometries.

3.2.6 UEqns.H

The momentum equations are solved in their phase-intensive forms, meaning they have been divided by the phase volume fraction. There are other discrepancies with the form developed in the theory part in Eq.2.3 that are described in details and justified numerically in Rusche's thesis [26], pp 108-111. For example, the effective stress tensor is decomposed in two terms, one diffusive part and one correction part. Once again, even if the formulation differs, the physics of the equations remains intact. Looking at the momentum equation for the liquid phase:

```
{
    volTensorField gradUbT = fvc::grad(Ub)().T();

    nuEffb = nutb + nub + 1.2*DS/2.0*alpha*magUr;

    volTensorField Rcb
    (
        "Rcb",
        ((2.0/3.0)*I)*(k + nuEffb*tr(gradUbT)) - nuEffb*gradUbT
    );

    surfaceScalarField phiRb =
        -fvc::interpolate(nuEffb)*mesh.magSf()*fvc::snGrad(beta)
        /fvc::interpolate(beta);

    UbEqn =
    (
        //(scalar(1) + alpha/beta/rhob*AVirtualMass)*   // Virtual mass force neglected
        (
            fvm::ddt(Ub)
          + fvm::div(phib, Ub, "div(phib,Ub)")
          - fvm::Sp(fvc::div(phib), Ub)
        )

      - fvm::laplacian(nuEffb, Ub)
      + fvc::div(Rcb)

      + fvm::div(phiRb, Ub, "div(phib,Ub)")
      - fvm::Sp(fvc::div(phiRb), Ub)

      + (fvc::grad(beta)/(fvc::average(beta)) & Rcb)
     ==
    //  g                              // Buoyancy term transfered to p-equation
      - fvm::Sp(alpha/beta/rhob*K, Ub)
//    + alpha/beta/rhob*K*Ua              // Explicit drag transfered to p-equation
      - alpha/beta/rhob*liftForce
//      + alpha/beta/rhob*virtualMassForceb  // Virtual mass force neglected
      - alpha/beta/rhob*turbulentDispersionForce
     - alpha/beta/rhob*wallLubricationForce
      + GammaLG/(beta)/rhob*Ua
      - fvm::Sp(GammaGL/(beta)/rhob, Ub)
    );

    UbEqn.relax();
}
```

where *Rcb* and *phiRb* are well described by Rusche [26]. The velocity equations are set up but not solved yet. As announced earlier, the virtual force has not been activated. It is also important to notice that the gravity and pressure gradient terms are actually not included in the momentum equations. They are treated at a later stage while coupling velocity and pressure in a so-called pressure equation. Transfering these terms to the pressure equation stabilizes the procedure.

3.2.7 pEqn.H

As explained before, the velocity equations are set up but the velocity fields not calculated yet when the code enters the pressure equation. The first step consists of predicting velocity fields from the matrices defined in *UEqns.H*. These predictions are not very accurate yet since they do not take pressure gradient nor gravity into account. This is achieved in the following lines:

```
02:     surfaceScalarField alphaf = fvc::interpolate(alpha);
03:     surfaceScalarField betaf = scalar(1) - alphaf;
04:
05:     volScalarField rUaA = 1.0/UaEqn.A();
06:     volScalarField rUbA = 1.0/UbEqn.A();
07:
08:     phia == (fvc::interpolate(Ua) & mesh.Sf());
09:     phib == (fvc::interpolate(Ub) & mesh.Sf());
10:
11:     surfaceScalarField rUaAf = fvc::interpolate(rUaA);
12:     surfaceScalarField rUbAf = fvc::interpolate(rUbA);
13:
14:     Ua = rUaA*UaEqn.H();
15:     Ub = rUbA*UbEqn.H();
```

Lines 5 and 6 retrieve and invert the central coefficients of the matrices defining the momentum equations. Multiplying these terms by the operation source from the same matrices in lines 14 and 15 yields to approximated velocity fields.

Interpolating these velocities at the cell face would yield to approximate fluxes. Then, corrections are applied to the mass fluxes:

```
38:     phia = (fvc::interpolate(Ua) & mesh.Sf()) + fvc::ddtPhiCorr(rUaA, Ua, phia)
39:         + phiDraga;
40:
41:     phib = (fvc::interpolate(Ub) & mesh.Sf()) + fvc::ddtPhiCorr(rUbA, Ub, phib)
42:         + phiDragb;
43:
44:     phi = alphaf*phia + betaf*phib;
```

where *fvc::ddtPhiCorr* introduces a correction due to the difference between the flux due to the interpolated velocities at the cell face and the real flux due to velocity, while *phiDraga* and *phiDragb* bring corrections resulting from gravity. The latter are defined as:

```
38:     phia = (fvc::interpolate(Ua) & mesh.Sf()) + fvc::ddtPhiCorr(rUaA, Ua, phia)
39:         + phiDraga;
40:
41:     phib = (fvc::interpolate(Ub) & mesh.Sf()) + fvc::ddtPhiCorr(rUbA, Ub, phib)
42:         + phiDragb;
43:
44:     phi = alphaf*phia + betaf*phib;
```

Then, the pressure equation is set-up and solved. This equation corresponds to a recasted version of the continuity equation expressed at cell faces, as explained by Rusche [26], pp. 112-113. This is done within a for-loop and several iterations may be necessary, if the mesh is non-orthogonal. Finally, the flux is completed by adding the pressure term. The obtained flux contains all physical contributions that occur in reality. Then, the phase fluxes are updated, as well as the velocities.

```
46:     surfaceScalarField Dp(”(rho*(1|A(U)))”, alphaf*rUaAf/rhoa + betaf*rUbAf/rhob);
47:
48:     for(int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
49:     {
50:         fvScalarMatrix pEqn
51:         (
        fvm::laplacian(Dp, p) == fvc::div(phi)
    );

    pEqn.setReference(pRefCell, pRefValue);
    pEqn.solve();

    if (nonOrth == nNonOrthCorr)
    {
        surfaceScalarField SfGradp = pEqn.flux()/Dp;

        phia -= rUaAf*SfGradp/rhoa;
        phib -= rUbAf*SfGradp/rhob;
        phi = alphaf*phia + betaf*phib;

        p.relax();
        SfGradp = pEqn.flux()/Dp;

        Ua += fvc::reconstruct(phiDraga - rUaAf*SfGradp/rhoa);
        Ua.correctBoundaryConditions();

        Ub += fvc::reconstruct(phiDragb - rUbAf*SfGradp/rhob);
        Ub.correctBoundaryConditions();

        U = alpha*Ua + beta*Ub;
    }
}
```

3.2.8 HEqn.H

At this stage, the energy equation can be implemented according to Eq.2.5 in analogy with the liquid velocity equation. Once again, a phase intensive formulation has been chosen.

```
{
    volVectorField gradHb = fvc::grad(Hb);

    volScalarField AlphaEffb = nutb/PrandtlbT + lambdab/(rhob*Cpb);

    volVectorField Qcb
    (
        "Qcb",
        - AlphaEffb*gradHb
    );

    HbEqn =
    (
        (
            fvm::ddt(Hb)
          + fvm::div(phib, Hb, "div(phib,Hb)")
          - fvm::Sp(fvc::div(phib), Hb)
        )

      - fvm::laplacian(AlphaEffb, Hb)
      + (fvc::grad(beta)/(fvc::average(beta)) & Qcb)

      ==

        DpDt/rhob
      + GammaLG/(scalar(1.0)-alpha)/rhob*(Ha)
      - fvm::Sp(GammaGL/(beta)/rhob, Hb)
```

```
        + qw*SfPerVol/beta/rhob
      );

      HbEqn.relax();
      HbEqn.solve();
  }
```

where *AlphaEffb* and *Qcb* denote respectively the effective thermal diffusivity and the effective heat transfer within the liquid phase. Lines 60 to 63 are the material derivative of the enthalpy. The first term on the RHS of the enthalpy equation is the energy gain by condensation, the second term is the energy loss by evaporation and the last one is the heat received from the heated wall. Recall that thanks to the field *SfPerVol*, this last term is non-zero only for near-wall cells.

3.2.9 kEpsilon.H

Lastly, the turbulent parameters are calculated as explained in the theory part. A classical $k - \epsilon$ model in combination with wall functions are used for the turbulence of the liquid phase if the turbulence is switched on by the user in the *turbulenceProperties* of the case. It starts with accumulating the wall contributions to the turbulent energy dissipation ϵ and to the buoyant production term G in the included header *wallFunctions.H*. Then the ϵ-equation is set. Before solving it, the terms of the ϵ-equation matrix that correspond to near-wall cells are removed because, as explained in the theory part, the ϵ-equation is not used in this region. This is performed within *wallDissipation.H*. After this step, the ϵ-equation is solved.

```
#include "wallFunctions.H"

// Dissipation equation
fvScalarMatrix epsEqn
(
    fvm::ddt(beta, epsilon)
  + fvm::div(phib, epsilon)
  - fvm::laplacian
    (
        alphaEps*nuEffb, epsilon,
        "laplacian(DepsilonEff,epsilon)"
    )
  ==
     C1*beta*G*epsilon/k
   - fvm::Sp(C2*beta*epsilon/k, epsilon)
);

#include "wallDissipation.H"

epsEqn.relax();
epsEqn.solve();
```

Next, the k-equation is set and solved. Further on, the turbulent viscosity is calculating either directly or in *wallViscosity.H* for the near-wall cells.

```
// Turbulent kinetic energy equation
fvScalarMatrix kEqn
(
    fvm::ddt(beta, k)
  + fvm::div(phib, k)
  - fvm::laplacian
    (
        alphak*nuEffb, k,
        "laplacian(DkEff,k)"
    )
  ==
        beta*G
      - fvm::Sp(beta*epsilon/k, k)
    );
    kEqn.relax();
    kEqn.solve();

    k.max(dimensionedScalar("zero", k.dimensions(), 1.0e-8));

    //- Re-calculate turbulence viscosity
    nutb = Cmu*sqr(k)/epsilon ;

     #include "wallViscosity.H"
```

At the end, the effective viscosities of both phases are also updated.

```
nuEffb = nutb + nub + 1.2*DS/2.0*alpha*magUr;
nuEffa = rhob/rhoa*nuEffb;
```

3.3 Test cases

3.3.1 The DEDALE experiment

The DEDALE experiment was carried out at EDF in 1995. One of its aims was to build up an important database for the validation of the NEPTUNE code, especially regarding the models used for interfacial forces, coalescence and breakup source terms and turbulence. It consists of an adiabatic vertical pipe of upwards bubbly flows with an air/water system at atmospheric pressure (at the outlet) and a temperature of 30°C (at the inlet). The transport properties of air and water under these conditions are shown in Table 3.1. The methacrylate pipe is 6 m long and has an inner diameter of 0.0381 m. Air is injected at the periphery of the water flow in 80 identical holes with an inner diameter of 0.6 mm.

Table 3.1: Transport properties of water and air in the DEDALE experiment

Property	Value
p	1.0 bar
ρ_l	995.65 kg/m^3
ρ_g	1.165 kg/m^3
λ_l	0.616 W/m/K
λ_g	0.026 W/m/K
c_{pl}	4179.8 J/kg/K
c_{pg}	1006.5 J/kg/K
ν_l	$8.00 \cdot 10^{-7}$ m^2/s
ν_g	$1.60 \cdot 10^{-5}$ m^2/s
σ	0.071 N/m

Measurements are realised at three different axial locations: $y/D = 8$, 55 and 155. At $y/D = 8$, the flow conditions are assumed to be the inlet ones. Void fraction, interfacial area concentration, bubble size and mean axial velocity, as well as mean and root mean square values of the liquid axial velocity are measured locally along the diameter at the three axial locations. A thorough description of the measurement devices will not be done here but one can refer to Bestion and Péturaud [4].

Of course, several tests were simulated with various combinations of inlet mass flux of air and water and we have kept two of them to validate our code. These tests are DEDALE1101 and DEDALE1103 and the parameters measured at the inlet are presented in Table 3.2.

In OpenFOAM, an equivalent 2D-channel has been modeled, instead of a real 3D-pipe, in order to limit the computational needs and the time of the simulations. For the same reasons and using the axial symmetry of the channel, only one half of the channel is considered. The mesh is homogeneous and composed respectively of 20 cells and 200 cells in the radial and axial directions, what sets the first cell node at a distance of

Table 3.2: DEDALE1101 and DEDALE1103 measured inlet conditions

Property	**DEDALE1101**	**DEDALE1103**
Liquid superficial velocity J_l	0.877 m/s	0.877 m/s
Gas superficial velocity J_g	0.0588 m/s	0.1851 m/s
α	0.048	0.152
k_l	$4.23{\cdot}10^{-3}$ m^2/s^2	$1.78{\cdot}10^{-2}$ m^2/s^2
a_i	97 m^{-1}	269 m^{-1}

$x+ \sim 20$. A view of the mesh and its radial decomposition is shown in Figure 3.4.

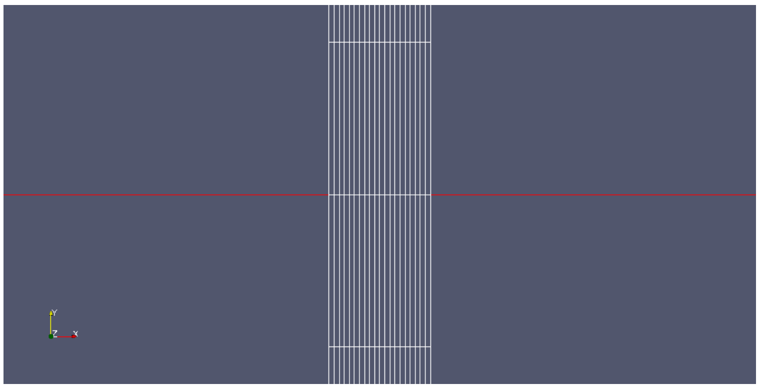

Figure 3.4: Zoom on the grid used for the DEDALE simulations

Initial and boundary conditions that have been applied are resumed in Table 3.3 and 3.4. The solver deals with real phase velocities and not superficial phase velocities which are the parameters given from the measurements. The former are thus calculated from the latter using a drift-flux model, whose distribution factor has been fixed so that the void fraction derived from the drift-flux model fits the void fraction measured at the inlet in the experiment. Another remark is that the boundary conditions at the inlet are taken uniform, what is of course not exact, since the inlet of the simulation refers in reality to the axial position $y/D = 8$ where the flow is developed. This choice for the simulations origins from the lack of data regarding the radial profile at $y/D = 8$ where only mean values are given. It must also be noted that the pressure p is set to 0. The real value of the pressure is used through an additional field *pReal* read from the *transportProperties* file, and it is added to the calculated pressure field p, which indeed only refers to the deviation from the reference value *pReal*.

Table 3.3: Initial and boundary conditions applied in OpenFOAM for DEDALE1101

Field	**Initial**	**Inlet**	**Outlet**	**Wall**	**Centerline**
U_l [m/s]	0.916	0.916	zeroGradient	0	symmetryPlane
U_g [m/s]	1.365	1.365	zeroGradient	0	symmetryPlane
α	0.048	0.048	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	97	97	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$4.23{\cdot}10^{-3}$	$4.23{\cdot}10^{-3}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$1.695{\cdot}10^{-2}$	$1.695{\cdot}10^{-2}$	zeroGradient	zeroGradient	symmetryPlane

3.3.2 The DEBORA experiment

The DEBORA experiment was carried out in 2000 by the CEA. This experiment differs from DEDALE since it aims to analyse upwards bubbly two-phase flows **with phase change** (both nucleation at the wall and condensation in the bulk). Its main goal was to provide a large data set covering for boiling in PWR up to DNB. However, it is both complex and costly to set up experimental facility at high pressure. Therefore, the choice has been made to use R12 which behaves at low pressure in a same manner as water under PWR conditions. The liquid-vapor density ratio in these experiments corresponds to that of water-steam at 95-150 bar.

Table 3.4: Initial and boundary conditions applied in OpenFOAM for DEDALE1103

Field	**Initial**	**Inlet**	**Outlet**	**Wall**	**Centerline**
U_l [m/s]	1.0157	1.0157	zeroGradient	0	symmetryPlane
U_g [m/s]	1.355	1.355	zeroGradient	0	symmetryPlane
α	0.152	0.152	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	269	269	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$1.78{\cdot}10^{-2}$	$1.78{\cdot}10^{-2}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$1.695{\cdot}10^{-2}$	$1.695{\cdot}10^{-2}$	zeroGradient	zeroGradient	symmetryPlane

DEBORA consists of a vertical pipe made of Inconel 600, whose physical properties can be seen in Table 3.5, with an inner diameter equal to 19.2 mm. Axially, it is divided into three parts: the adiabatic inlet section (1 m length), the heated section (3.5 m length) and the adiabatic outlet section (0.5 m). The intermediate section is heated with an electric heating system imposing a unifrom or non-uniform heat flux density. Subcooled R12 is fed at the bottom of the pipe, vapor bubbles are formed at the heated wall and migrate to the bulk where they condensate.

Table 3.5: Properties of Inconel 600 used as pipe material in the DEBORA experiment

Property	**Value**
ρ_w	8430 kg/m^3
λ_w	15.0 W/m/K
c_{pw}	444 J/kg/K

The measurements of radial distributions are performed at one axial location only, at the end of the heated section, but for 20 different radial positions. The radial profiles of void fraction and bubble diameters are measured by means of an optical probe. Besides, the liquid temperature radial profile is obtained by means of thermocouples.

Here as well, various experiments have been performed, combining different input parameters. The cases which are used for the validation of the present solver were all uniformly heated, and their experimental conditions are presented in Table 3.6 and the transport properties of R12 under these conditions are shown in Table 3.7.

Table 3.6: DEBORA selected cases and their experimental conditions

Case	**Pressure**	**Mass flow rate**	**Wall heat flux density**	**Inlet temperature**
	[bar]	**[kg/m^2/s]**	**[kW/m^2]**	**[°C]**
DEB5	26.15	1986	73.89	68.52
DEB6	26.15	1984.9	73.89	70.53
DEB10	14.59	2027.8	76.24	34.91
DEB13	26.17	2980.9	109.42	69.20

In OpenFOAM, the geometry created is again a simplified version: half of a 2D-pipe is created to model this experiment. Furthermore, one will only simulate the 3.5 m long heated part, neglecting the adiabatic inlet section. Once again, the grid is homogeneous with 20 cells in the radial direction and 250 cells in the axial direction, as it has been done by Lopez De Bertodano and Prabhudharwadkar [23]. Near-wall cells are thus located at a distance $x+ \sim 60$. Finally, the initial and boundary conditions used in OpenFOAM are summarised in Table 3.8, 3.9, 3.10 and 3.11. The same comment on the field p can be made: it represents indeed the deviation from a reference pressure. It must be observed that the initial and inlet values taken for U_g and a_i, while α is null, will not affect the final results after convergence of the code, but it might be necessary to chose them rather close to their final values for stability's sake.

Table 3.7: Transport properties of R12 in the selected DEBORA cases

Property	DEB5,DEB6	DEB10	DEB13
ρ_l [kg/m^3]	1017.1	1177.2	1016.8
ρ_g [kg/m^3]	172.06	84.91	172.24
λ_l [W/m/K]	$4.57 \cdot 10^{-2}$	$5.58 \cdot 10^{-2}$	$4.57 \cdot 10^{-2}$
λ_g [W/m/K]	$1.76 \cdot 10^{-2}$	$4.65 \cdot 10^{-2}$	$1.76 \cdot 10^{-2}$
c_{pl} [J/kg/K]	1419.8	1111.50	1420.7
c_{pg} [J/kg/K]	1290.8	861.71	1292.0
ν_l [m^2/s]	$8.80 \cdot 10^{-8}$	$1.12 \cdot 10^{-7}$	$8.80 \cdot 10^{-8}$
ν_g [m^2/s]	$9.13 \cdot 10^{-8}$	$1.55 \cdot 10^{-7}$	$9.13 \cdot 10^{-8}$
σ [N/m]	$1.77 \cdot 10^{-3}$	$4.65 \cdot 10^{-3}$	$1.77 \cdot 10^{-3}$
T_{sat} [°C]	86.64	58.15	86.64
$i_{l,sat}$ [J/kg/K]	292530.0	258450.0	292590.0
i_{lg} [J/kg/K]	86070.0	116100.0	86010.0

Table 3.8: Initial and boundary conditions used in OpenFOAM for DEB5

Field	Initial	Inlet	Outlet	Wall	Centerline
U_l [m/s]	1.744	1.744	zeroGradient	0	symmetryPlane
U_g [m/s]	1.9	1.9	zeroGradient	0	symmetryPlane
α	0	0	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	5000	5000	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$1.14 \cdot 10^{-2}$	$1.14 \cdot 10^{-2}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$1.49 \cdot 10^{-1}$	$1.49 \cdot 10^{-1}$	zeroGradient	zeroGradient	symmetryPlane
i_l [J/kg]	269740.0	269740.0	zeroGradient	zeroGradient	symmetryPlane

Table 3.9: Initial and boundary conditions used in OpenFOAM for DEB6

Field	Initial	Inlet	Outlet	Wall	Centerline
U_l [m/s]	1.76	1.76	zeroGradient	0	symmetryPlane
U_g [m/s]	1.9	1.9	zeroGradient	0	symmetryPlane
α	0	0	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	2000	2000	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$1.16 \cdot 10^{-2}$	$1.16 \cdot 10^{-2}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$1.53 \cdot 10^{-1}$	$1.53 \cdot 10^{-1}$	zeroGradient	zeroGradient	symmetryPlane
i_l [J/kg]	272060.0	272060.0	zeroGradient	zeroGradient	symmetryPlane

Table 3.10: Initial and boundary conditions used in OpenFOAM for DEB10

Field	Initial	Inlet	Outlet	Wall	Centerline
U_l [m/s]	1.586	1.586	zeroGradient	0	symmetryPlane
U_g [m/s]	1.8	1.8	zeroGradient	0	symmetryPlane
α	0	0	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	2000	2000	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$9.43 \cdot 10^{-3}$	$9.43 \cdot 10^{-3}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$1.12 \cdot 10^{-1}$	$1.12 \cdot 10^{-1}$	zeroGradient	zeroGradient	symmetryPlane
i_l [J/kg]	234030.0	234030.0	zeroGradient	zeroGradient	symmetryPlane

Table 3.11: Initial and boundary conditions used in OpenFOAM for DEB13

Field	**Initial**	**Inlet**	**Outlet**	**Wall**	**Centerline**
U_l [m/s]	2.626	2.626	zeroGradient	0	symmetryPlane
U_g [m/s]	2.9	2.9	zeroGradient	0	symmetryPlane
α	0	0	zeroGradient	zeroGradient	symmetryPlane
p	0	zeroGradient	0	zeroGradient	symmetryPlane
a_i [m^{-1}]	2000	2000	zeroGradient	zeroGradient	symmetryPlane
k [m^2/s^2]	$2.59{\cdot}10^{-2}$	$2.59{\cdot}10^{-2}$	zeroGradient	zeroGradient	symmetryPlane
ϵ [m^2/s^3]	$5.08{\cdot}10^{-1}$	$5.08{\cdot}10^{-1}$	zeroGradient	zeroGradient	symmetryPlane
i_l [J/kg]	270520.0	270520.0	zeroGradient	zeroGradient	symmetryPlane

CHAPTER 4

RESULTS

This chapter treats the results of the validation of the solvers developed. First the adiabatic solver *myTwoPhaseEulerFoamAdiabatic* is validated against the DEDALE cases. Then the results obtained using *myTwoPhaseEulerFoamBoiling* on the DEBORA cases are presented.

4.1 Results from *myTwoPhaseEulerFoamAdiabatic* on the DEDALE cases

DEDALE1101 and DEDALE1103 are the two tested configurations and they have been described in detail in the previous section. As it has been explained, the measurements in the first measuring section ($y/D = 8$) are then used as inlet boundary conditions for the simulations.

4.1.1 Validation on DEDALE1101

For DEDALE1101, simulations were initially run using the Hibiki and Ishii models for coalescence and break-up, which constitute the standard models for the adiabatic solver. However, for DEDALE1101, it has then been decided to launch new simulations using this time the Yao and Morel models for coalescence and break-up. The results calculated from OpenFOAM in the two other measuring sections ($y/D = 55$ and 155) are compared to the experimental data as well as the results obtained by Yao and Morel after implementation of their own coalescence and break-up models in the three-dimensional module of the CATHARE code (outcome of a joint effort between Areva NP, CEA, EDF and IRSN) in Figure 4.1. Unfortunately, there is no direct information about uncertainties for the local measurements. Nevertheless, using coherency tests on the mean liquid and gas volumetric fluxes, Yao and Morel [39] evaluated the uncertainties on the gas and liquid velocities by making the assumptions that the local quantities do not vary in the cross-section. These uncertainties are represented in Figure 4.1 by blue vertical error bars.

The experiments show that at the first elevation ($y/D = 55$), the void fraction forms a so called wall peak, where bubbles concentrate in a region close to the wall. This is the result of the equilibrium between the radial components of the lift force, the turbulent dispersion force and the wall lubrication force. At the second measuring cross-section ($y/D = 155$), the void profile is different: it is intermediate between wall peaking and core voiding. Actually, the coalescence effects increase along the pipe, which results in bigger bubbles, modifying this force equilibrium (acting notably on the lift force coefficient as shown in Figure 2.1) which changes the void fraction profile.

When the Hibiki and Ishii models are selected, the solver shows globally good agreement with the experimental data at $y/D = 55$. The void fraction profile is satisfying: the wall peaking phenomenon is captured, whereas Yao and Morel could not capture the negative gradient of the void fraction close to the wall, and the values are reasonably accurate both at the peak and in the center (where it is actually better than what Yao and Morel got). Nevertheless, Yao and Morel's results are better to calculate the position of the wall peak: the wall peak obtained using the present solver is slightly too far from the wall. Concerning the bubble diameter, the results obtained are very good in the bulk of the liquid but the increase of the bubble diameter

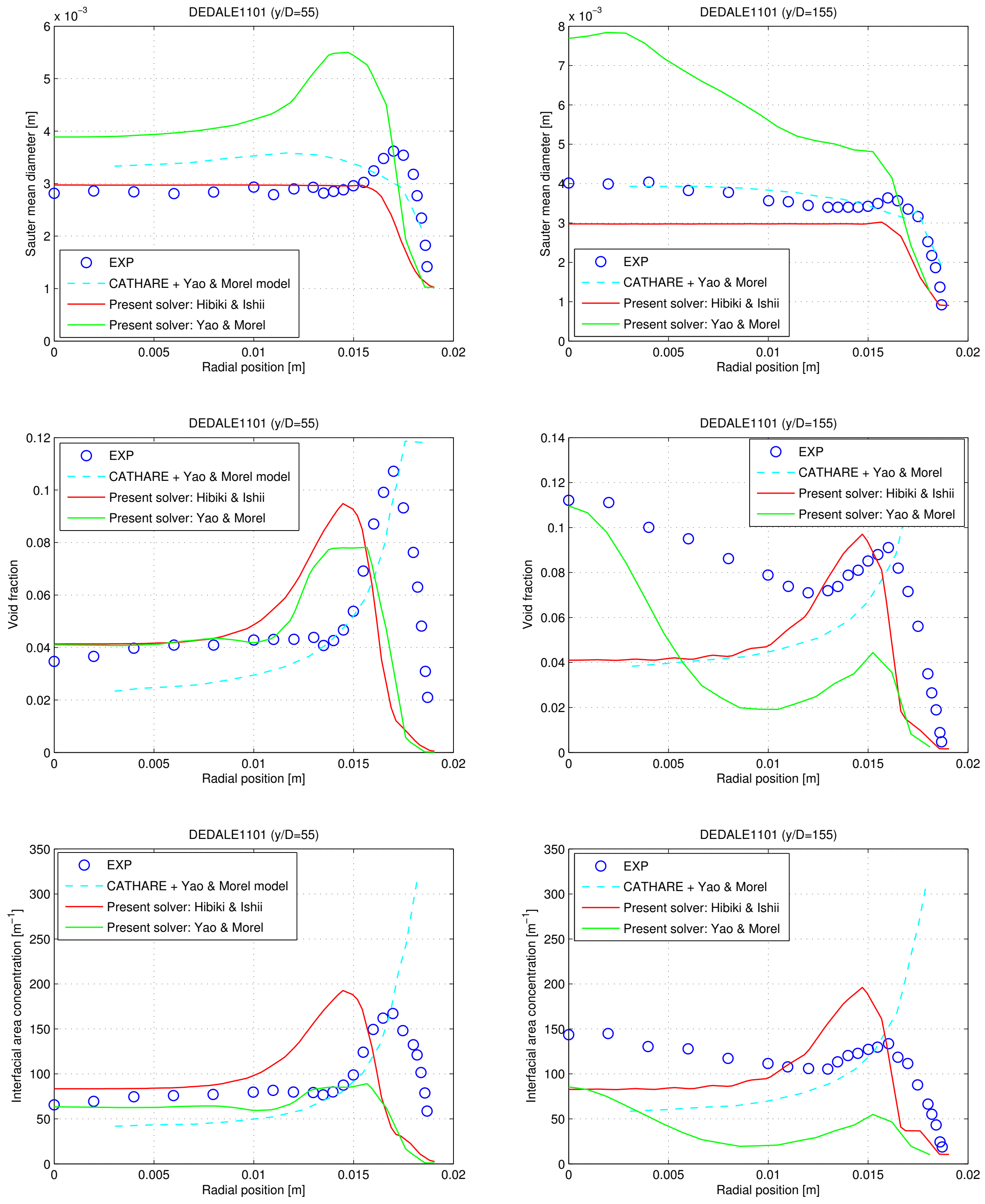

Figure 4.1: Comparison for DEDALE1101 between the results obtained in OpenFOAM with *myTwoPhaseEulerFoamAdiabatic*, the experimental data and the results obtained by Yao and Morel with CATHARE

around the wall peak region is not detected. The predictions for the gas velocity are excellent. The biggest discrepancy between the present simulations and the experiments concern the liquid velocity, which is underestimated in the whole cross-section. This might seem surprising because one could think that the mass flow used in the simulations is then not equal to the one really applied in the experiments. However, two different aspects must be pointed out. First, the predictions obtained by Yao and Morel with CATHARE are underestimated in the whole cross-section too, which proves that this problem in not inherent to my simulations but probably due to inconsistencies between the measured values and the input parameters that are supposedly used for this experiment. Then, while evaluating the uncertainties on the measurements of phase velocities, Yao and Morel assumed that the radial profiles of local parameters are uniform, which is

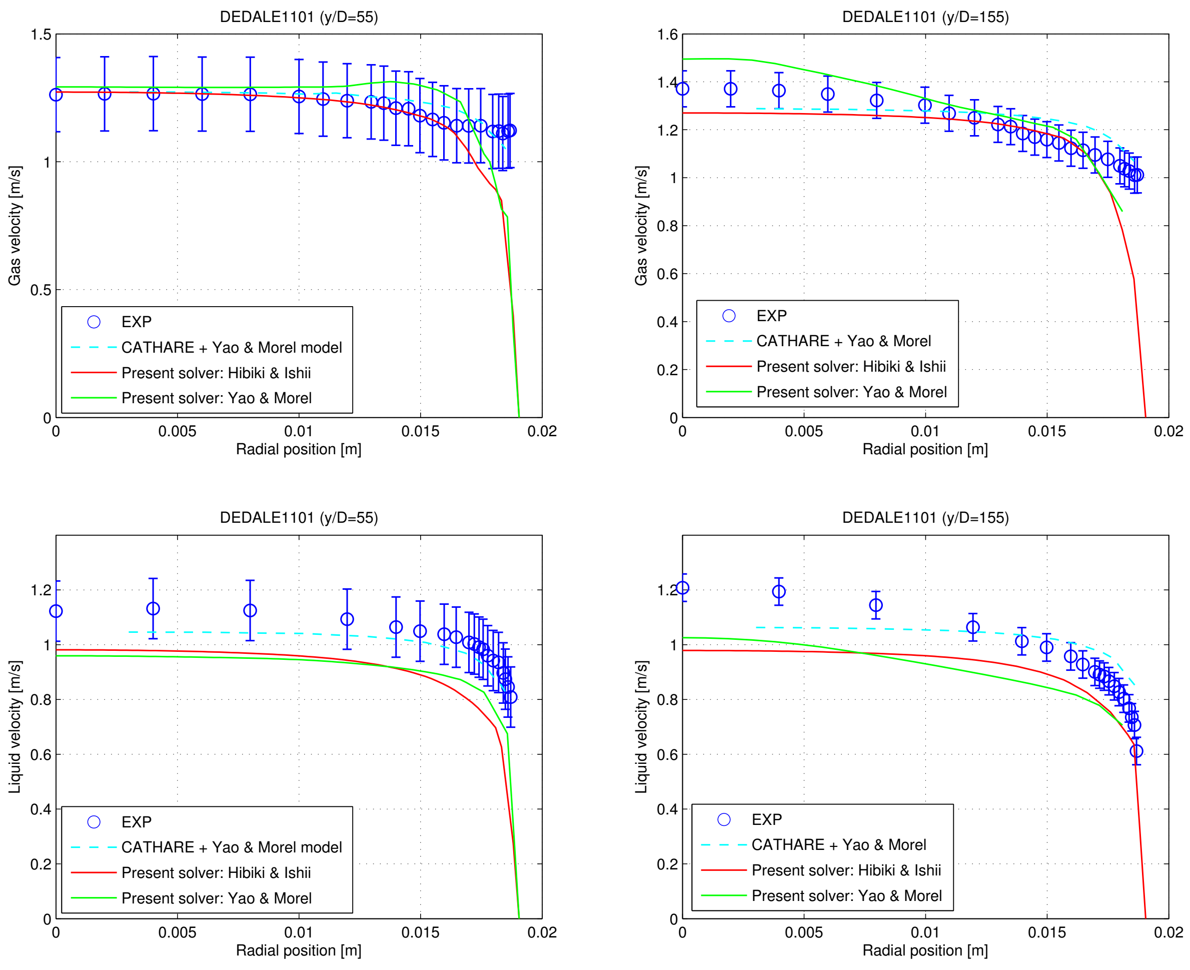

Figure 4.1: (*continued*)

a very crude approximation. Therefore, the real uncertainties should be significantly higher and the radial profiles of the liquid velocity obtained from *myTwoPhaseEulerFoamAdiabatic* could indeed be in agreement with that coming from the measurements. But the comparisons performed at $y/D = 155$ reveal that the Hibiki and Ishii models fail to reproduce the axial evolution of the radial distribution. This can also be seen on from Figure 4.2.

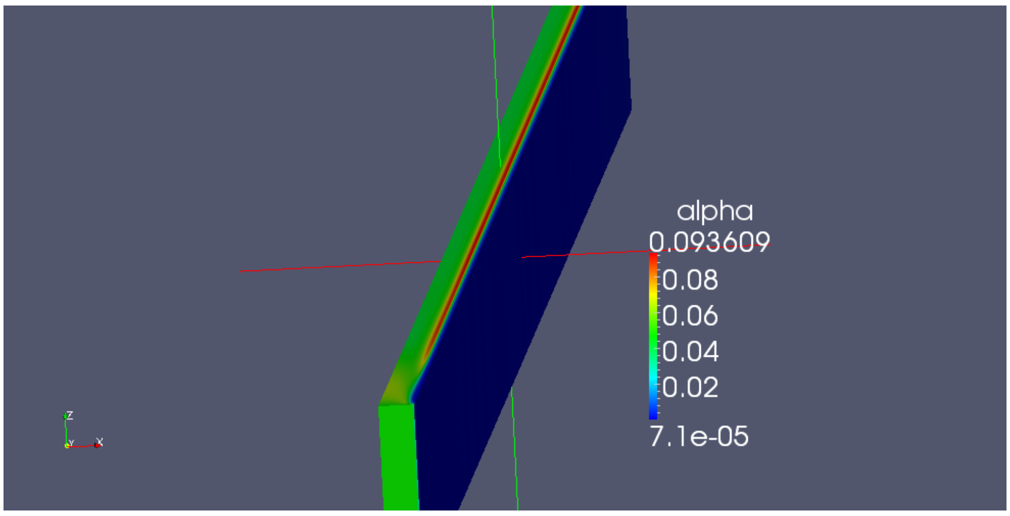

Figure 4.2: 3D view of DEDALE1101 when the Hibiki and Ishii models are selected

If instead of the Hibiki and Ishii models, the Yao and Morel models for coalescence and break-up phenomena are used in the present solver, the wall peaking phenomenon at $y/D = 55$ is still qualitatively captured but the numerical results are worse. However at $y/D = 155$, the solver succeeds in capturing qualitatively the axial evolution of the radial profiles and the progressive core voiding phenomenon as observed in Figure 4.3, which is not possible with CATHARE even if the same models for coalescence and break-up are used. The fact that the results obtained with OpenFOAM depends on the models which are selected can be justified. On the one hand, the Hibiki and Ishii models involve constants which have been fitted on air/water adiabatic experiments to capture the wall peak at intermediate height. On the other hand, all the terms presented in the Yao and Morel models have been derived analytically.

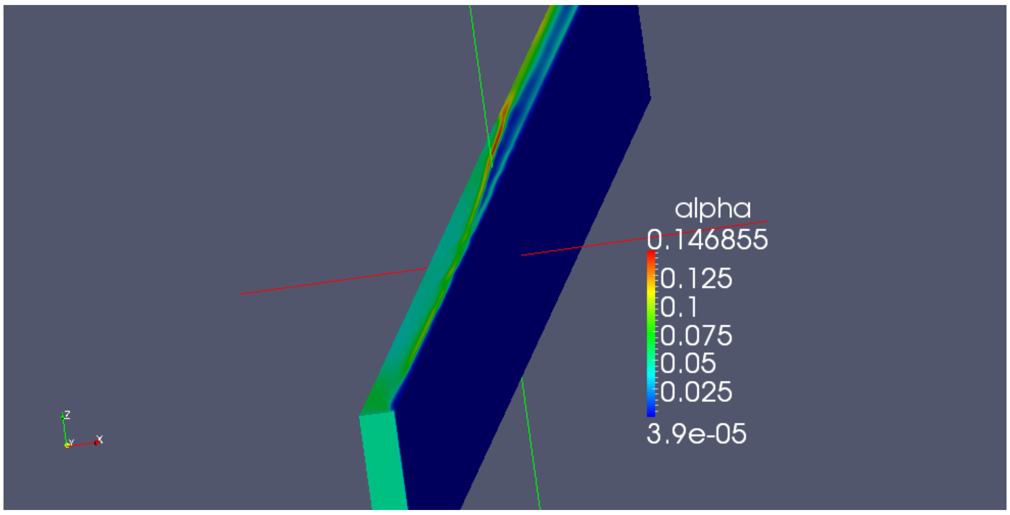

Figure 4.3: 3D view of DEDALE1101 when the Yao and Morel models are selected

4.1.2 Validation on DEDALE1103

For this test, a transition in the flow regime has been observed during the experiment. At $y/D = 55$, the flow is bubbly, but at $y/D = 155$, the flow is governed by the slug regime. Therefore, it would not be coherent to use the Yao and Morel models with the present solver because this association captures the axial evolution, as shown in the previous sub-section and would try to resolve the flow in the slug region, what is off the frame of this work, and out of the validity range of this solver. Since the Hibiki and Ishii models do not capture the axial evolution of the radial profiles, they can be chosen in order to compare the results at $y/D = 55$ with the experimental results as well as what Yao and Morel obtained with CATHARE, what is show in Figure 4.4.

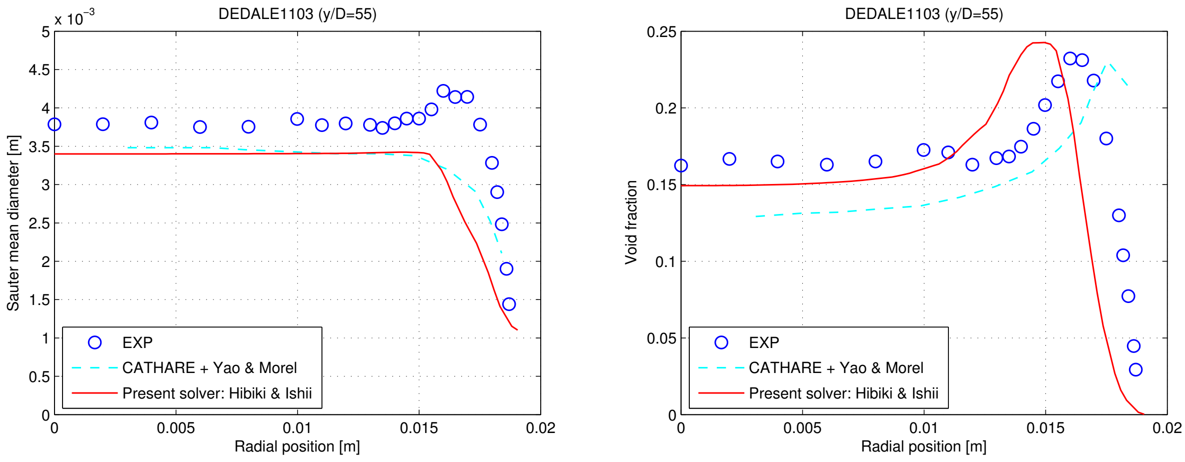

Figure 4.4: Comparison for DEDALE1103 between the results obtained in OpenFOAM with *myTwoPhaseEuler-FoamAdiabatic*, the experimental data and the results obtained by Yao and Morel with CATHARE

Once again, the solver reflects rather well the reality and its results are close to that obtained by Yao and Morel with CATHARE. The wall peak of the void fraction profile is capture here again and the values in the

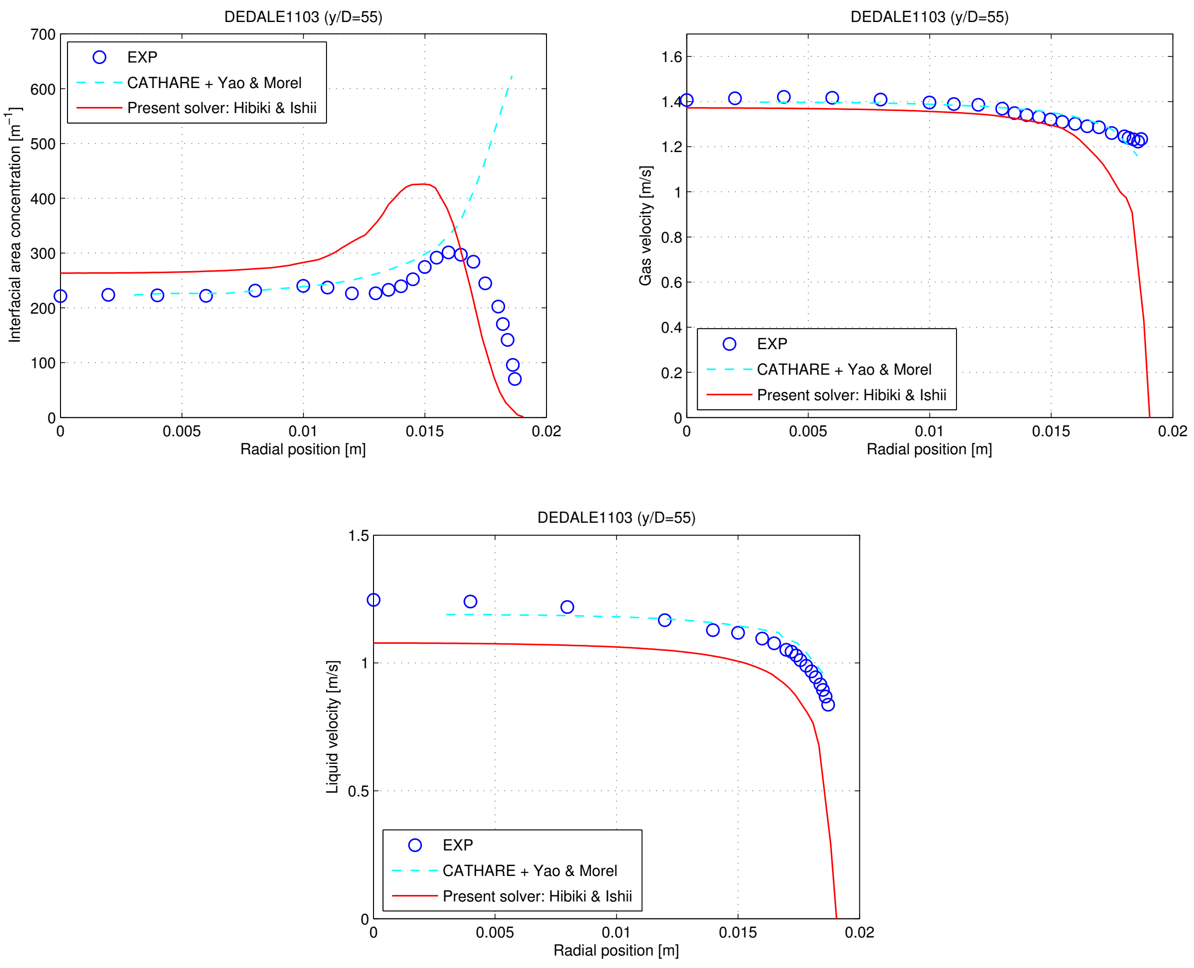

Figure 4.4: (*continued*)

bulk of the coolant and at the peak are even better than that of CATHARE, but it is slightly too far from the wall once again. The bubble diameter profile is well reproduced, except its increase in the wall peak region. The gas velocity is also well predicted.The liquid velocity is underestimated, as it was already the case for DEDALE1101, but the lack of data regarding the uncertaintities make any conclusion complicated. One could nevertheless expect higher uncertaintities on the liquid velocities for DEDALE1103 than for DEDALE1101 since the mass flow rate is higher.

4.2 Results from *myTwoPhaseEulerFoamBoiling* on the DEBORA cases

4.2.1 Validation on the selected DEBORA cases

Calculations of the four selected DEBORA cases indicated in Table 3.6 have been made with *myTwoPhaseEulerFoamBoiling*, using either the Hibiki and Ishi, or the Yao and Morel model. In Figures 4.5-4.9 the results are compared to the experimental data and those obtained by Yao and Morel with CATHARE. Each figure presents four graphs corresponding to the radial profiles of a local parameter at the outlet of the heated section for DEB5, DEB6, DEB10 and DEB13. The void fraction and temperature profiles are presented in Figures 4.5 and 4.7 respectively whereas the geometric local parameters, meaning the bubble diameter and the interfacial area concentration are presented in Figures 4.8 and 4.9 respectively.

Figure 4.5 shows that the void fraction predictions are globally not good. The void fraction is strongly underestimated by our model. Indeed, void is almost only present for a few layers of cells close to the wall and the core of the duct in our simulations is almost entirely occupied by pure liquid. However, the experimental results show evidence of significant amount of void (few percents) even in the bulk of the liquid.

It seems that there is not enough vapor generated at the wall. Another remark that can be done is that using the Hibiki and Ishii model gives higher void fraction at the wall than the Yao and Morel model.

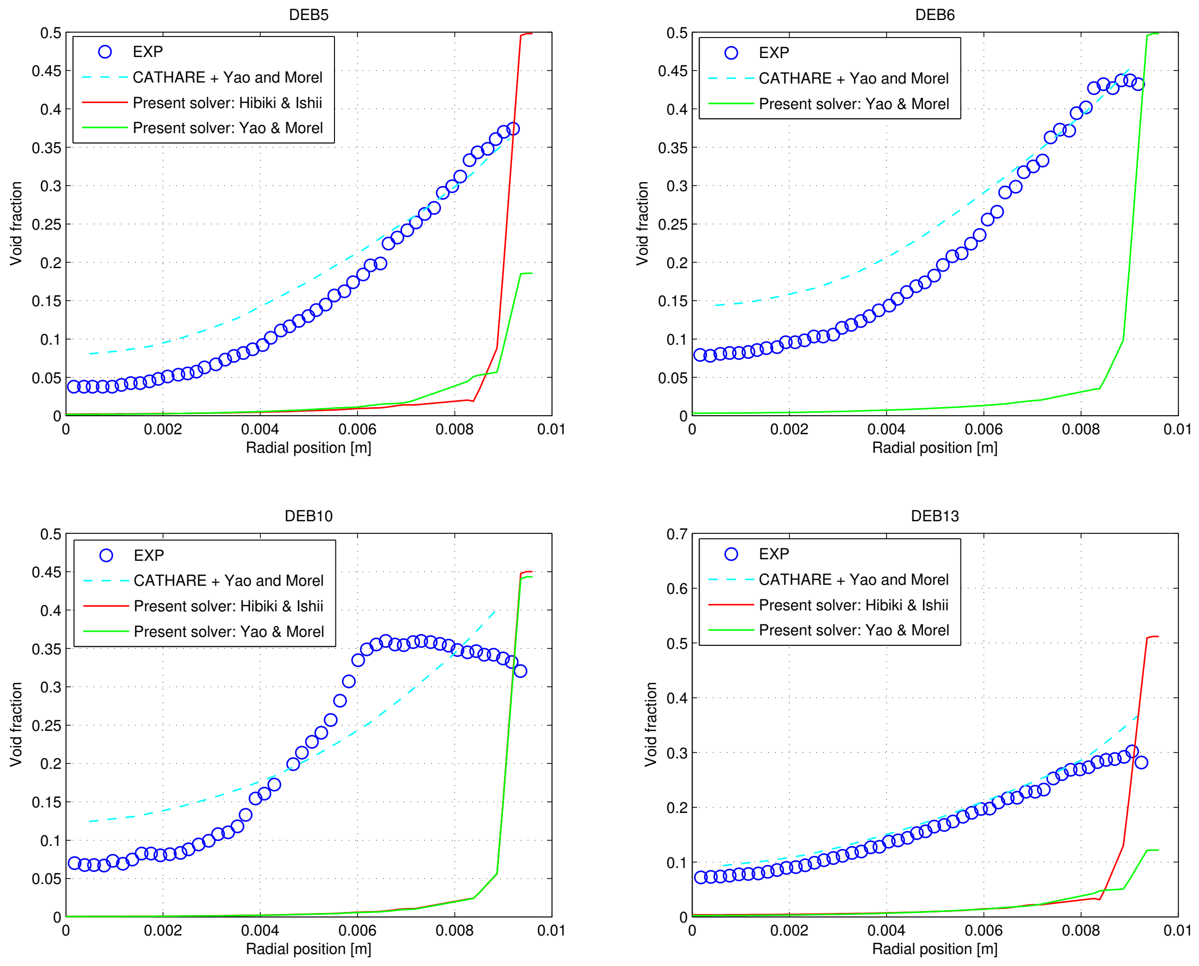

Figure 4.5: Comparison of the void fraction profiles at the end of the heated section between the results obtained in OpenFOAM with *myTwoPhaseEulerFoamBoiling*, the experimental data and the results obtained by Yao and Morel with CATHARE for the selected DEBORA cases

This underestimated void fraction has to be analysed in light of the liquid temperature profiles. The trend of the temperature profiles is consistent: the mean liquid subcooling is much lower at the end of the test section than at the inlet, and it also decreases along the radius of the pipe, as shown in Figure 4.6. Nevertheless Figure 4.7 reveals strong discrepancies between the measured value and the simulations for the liquid subcooling: the liquid is not as warm as it should be, probably implying that the problem in the present solver lies in the treatment of the wall heat flux.

Figure 4.8 shows the limit of our present solver too. The bubble diameter is highly understimated in the core of the pipe, but the value at the wall are in good agreement with the experimental data. Since the bubble departure diameter is the key parameter of the Kurul and Podowski boiling sub-model, the evaporation intermediate parameters are reasonable and therefore not the origin of the problem. If the liquid subcooling was lower, more bubbles would be generated and migrate towards the centerline of the pipe, increasing the void fraction in the bulk of the coolant and the bubble diameter too, according to Eq.2.48.

4.2.2 Sensitivity tests: influence of the grid resolution

When developing a solver, a programmer wants to implement a numerical procedure which is as grid independent as possible. It would not be physically acceptable to have a solver whose final results are very sensitive to the grid adopted for the simulations. For the *myTwoPhaseEulerFoamBoiling* solver in combination with the Yao and Morel model, several grid dependency tests have been performed on the DEB5 case. The same simulation has be run using 10, 14, 16 or 20 radial cells and the predictions on the local parameters are

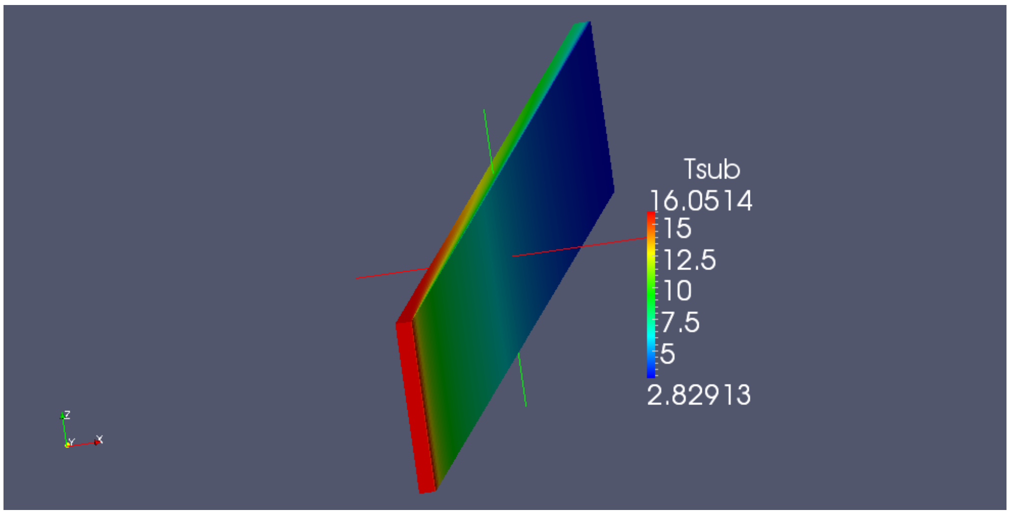

Figure 4.6: 3D view of the liquid subcooling for the DEB5 case when the Yao and Morel model is used

compared in Figure 4.10.

Figure 4.10 shows essentially that a small variation of the radial refinement affects considerably the void fraction in the near wall region, influencing the liquid temperature. The higher the void fraction at the wall, the higher the subsequent condensation, the higher the liquid temperature. Once again, this defect in the code is due to the treatment of the wall heat flux and evaporation rate and it will be explained here. At the wall, it is possible to know the wall heat flux density $q_w^{''}$ (which is an input parameter) and the evaporation mass flux per unit wall area. These two terms are given per unit wall area, but they have to be taken into account for the conservation equations of near-wall cells. The difficulty is: how to convert the known terms per unit wall area into volumetric terms used in the conservations equations? The strategy employed here has been to multiply by $A_w^{'''}$ representing the contact area between the wall and the cell per unit volume, as shown in Eqs. 2.5 and 2.45. This term is null for non wall-adjacent cells, and for wall-adjacent cells, it is equal to:

$$A_w^{'''} = \frac{y_{cell} z_{cell}}{x_{cell} y_{cell} z_{cell}} = \frac{1}{x_{cell}} \tag{4.1}$$

where $(x_{cell}, y_{cell}, z_{cell})$ are the cell dimensions, as described in Figure 4.11. This shows that the source terms due to the evaporation or the wall heat flux appearing in the conservation equations are highly sensitive to the mesh refinement, and so are the solutions to these equations.

4.2.3 Sensitivity tests: influence of the inlet gas velocity and interfacial area concentration

At the inlet, water is subcooled and therefore a proper definition of certain gas parameters is not possible. If the void fraction α is obviously null, how to define at the inlet the interfacial area concentration, the bubble diameter and the gas velocity when there are no gas bubbles?

Since $\vec{U}_g$ and a_i are two fields calculated through partial derivative equations, it has been decided to set them uniformly at the inlet, giving them realistic values read from the experimental results obtained once the steady state is reached. However, a modification of these inlet values, as long as they are realistic, should not alter the results at steady state.

In order to verify that the results of the solver *myTwoPhaseEulerFoamBoiling* are not affected by the inlet gas velocity nor the inlet interfacial void fraction, a series of dependency tests have been carried out on the DEB5 cases, modifying from one simulation to the other only one of these parameters and keeping all the others identical. The influence of the inlet gas velocity can be seen in Figure 4.12 and that of the inlet interfacial area concentration is depicted in Figure 4.13. Modifying these parameters do not affect the final results of the solver, as expected from a purely physical point of view. This is a good sign of the solver's robustness.

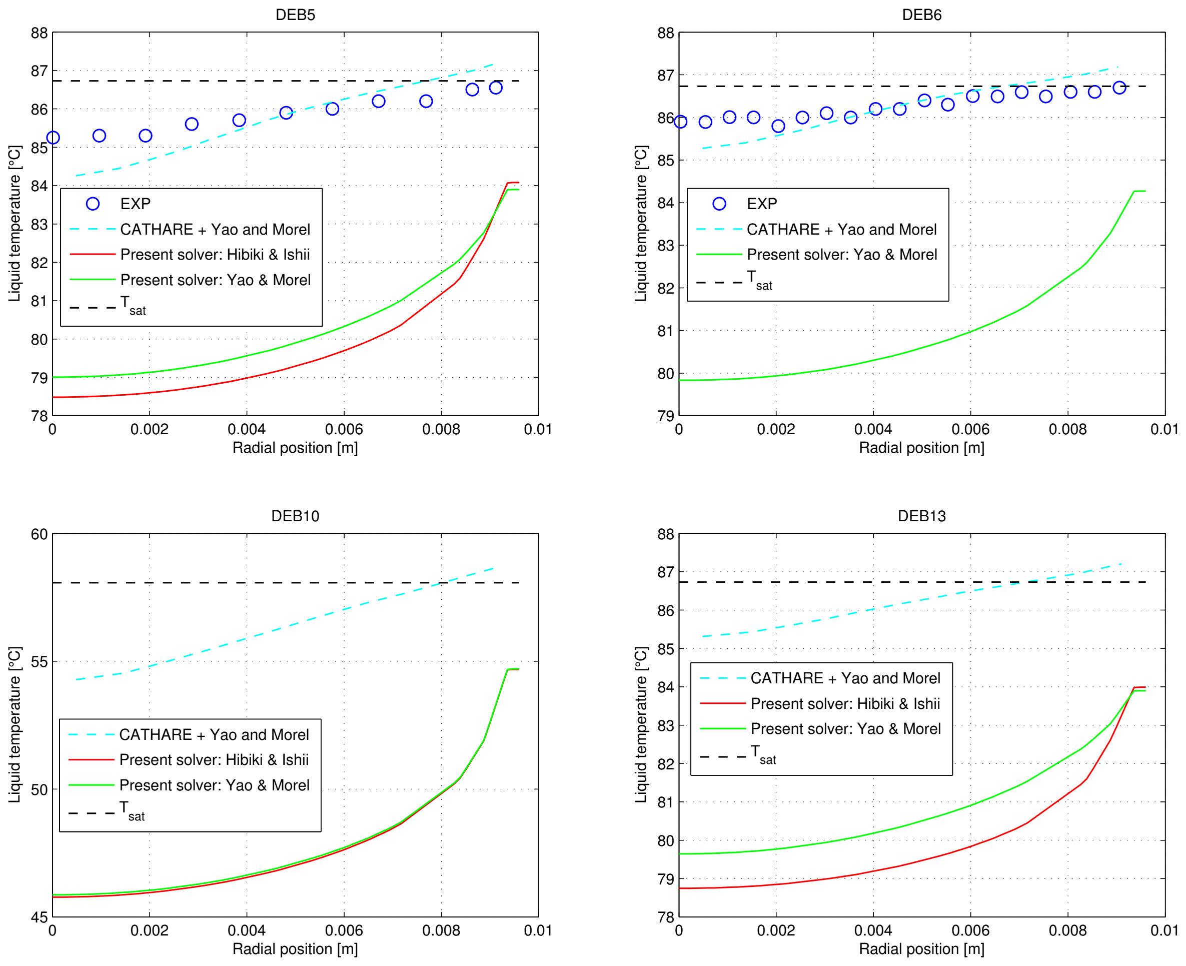

Figure 4.7: Comparison of the liquid temperature profiles at the end of the heated section between the results obtained in OpenFOAM with *myTwoPhaseEulerFoamBoiling*, the experimental data and the results obtained by Yao and Morel with CATHARE for the selected DEBORA cases

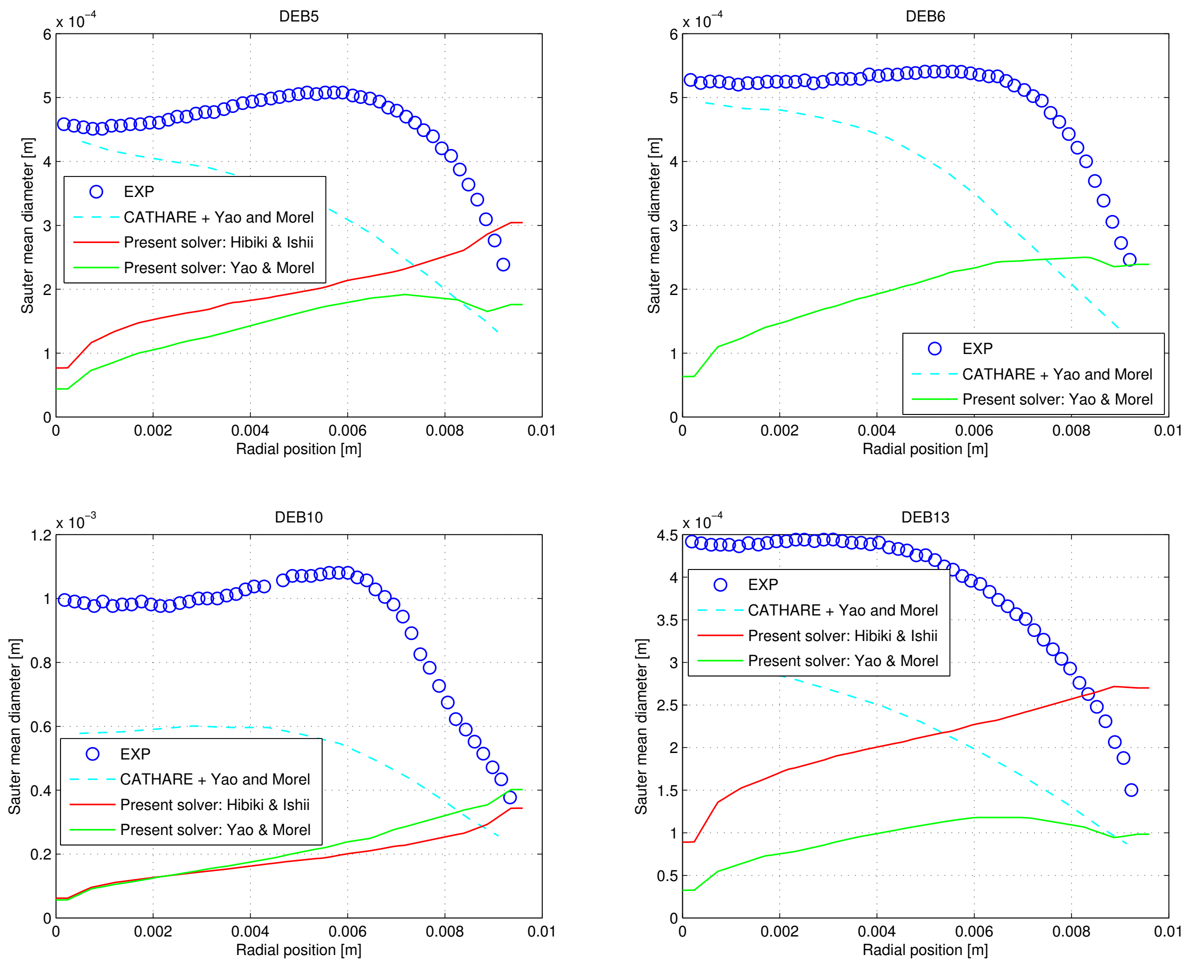

Figure 4.8: Comparison of the bubble diameter profiles at the end of the heated section between the results obtained in OpenFOAM with *myTwoPhaseEulerFoamBoiling*, the experimental data and the results obtained by Yao and Morel with CATHARE for the selected DEBORA cases

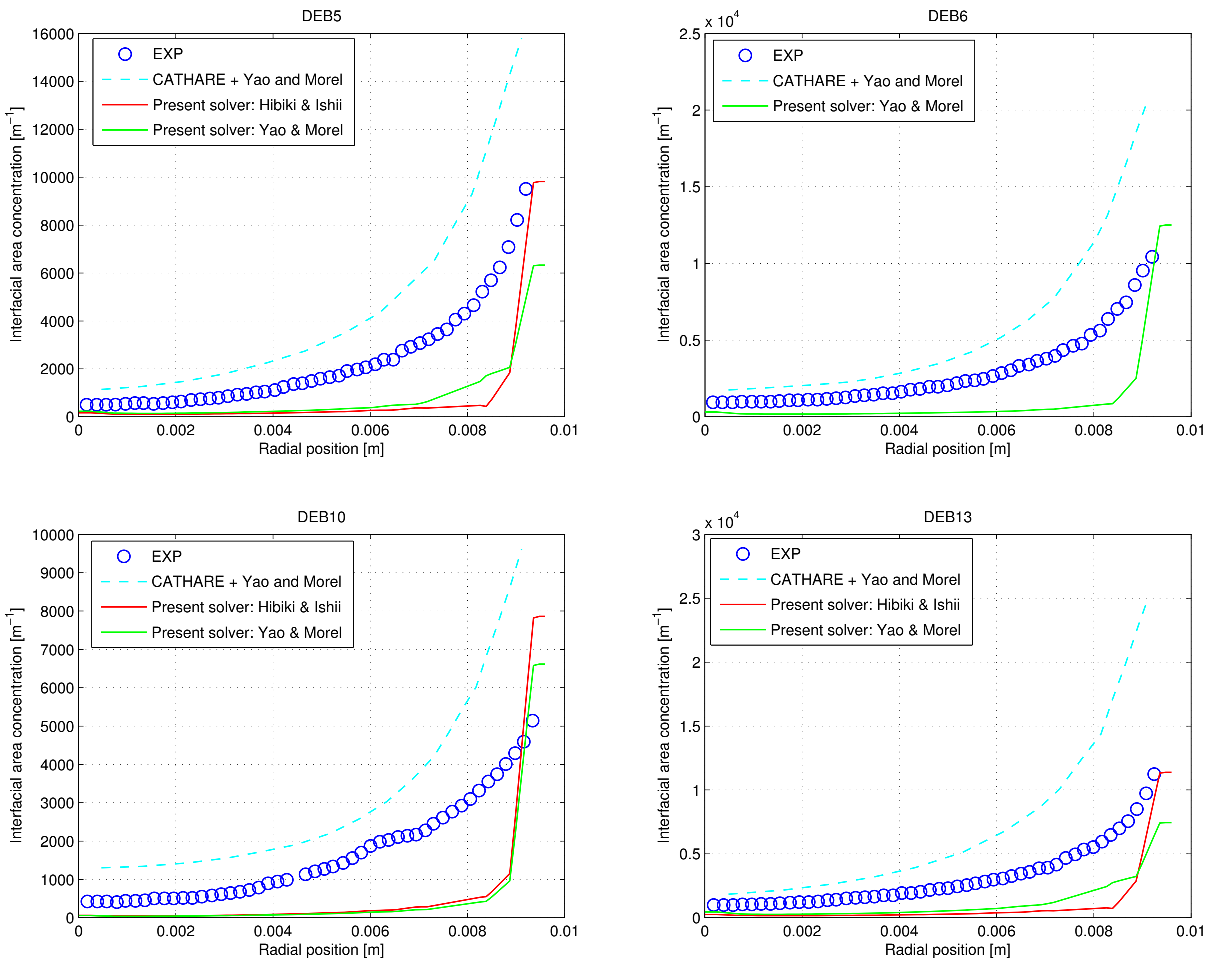

Figure 4.9: Comparison of the interfacial area concentration profiles at the end of the heated section between the results obtained in OpenFOAM with *myTwoPhaseEulerFoamBoiling*, the experimental data and the results obtained by Yao and Morel with CATHARE for the selected DEBORA cases

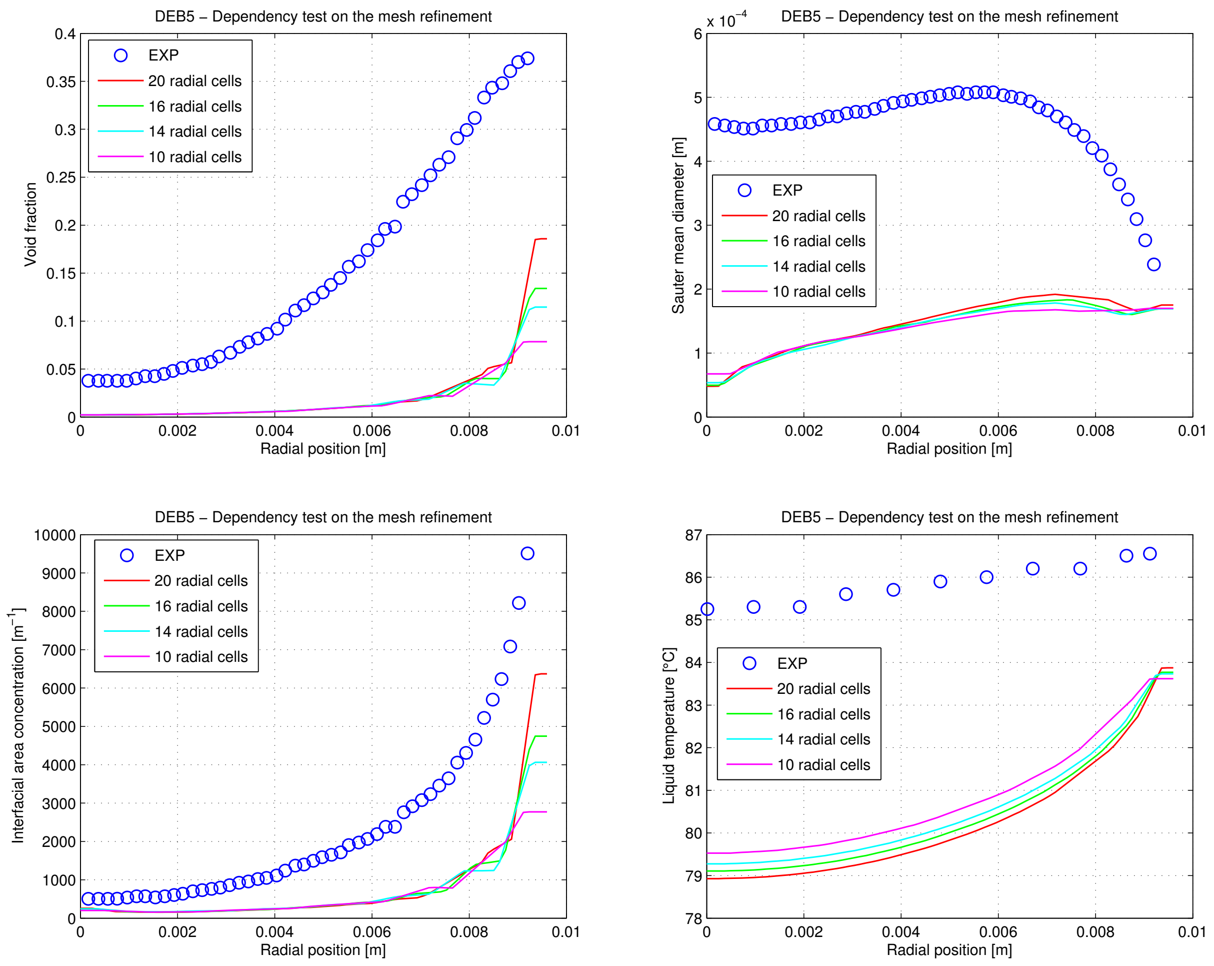

Figure 4.10: Comparison of the results obtained from *myTwoPhaseEulerFoamBoiling* at the end of the heated section for DEB5 with variying number of radial cells for the grid

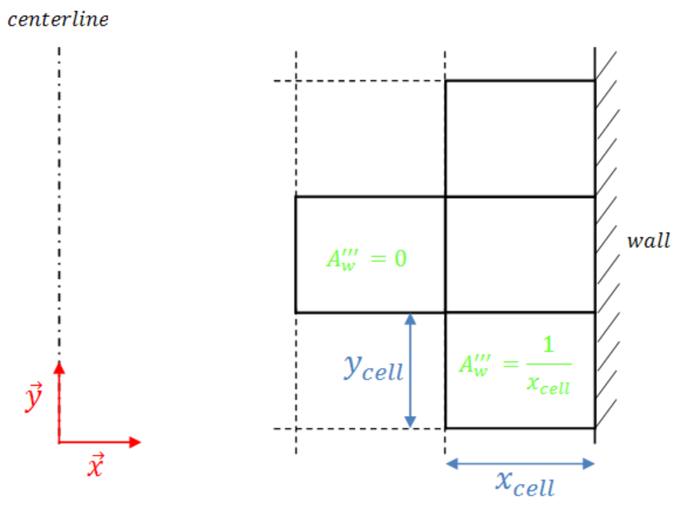

Figure 4.11: Map of the field A_w''' introducing a grid-dependency in the solvers

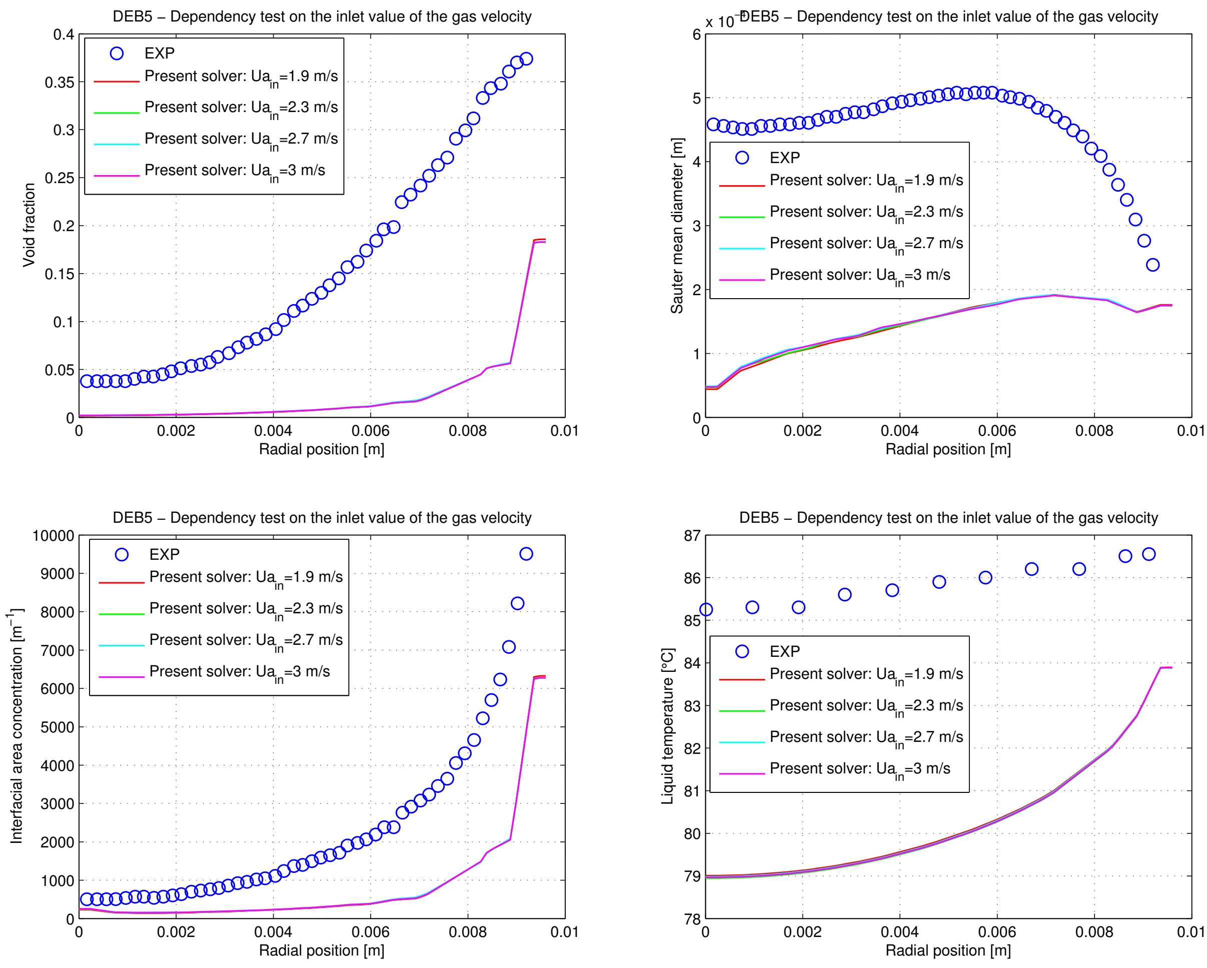

Figure 4.12: Comparison of the results obtained from *myTwoPhaseEulerFoamBoiling* at the end of the heated section for DEB5 with varying inlet gas velocity

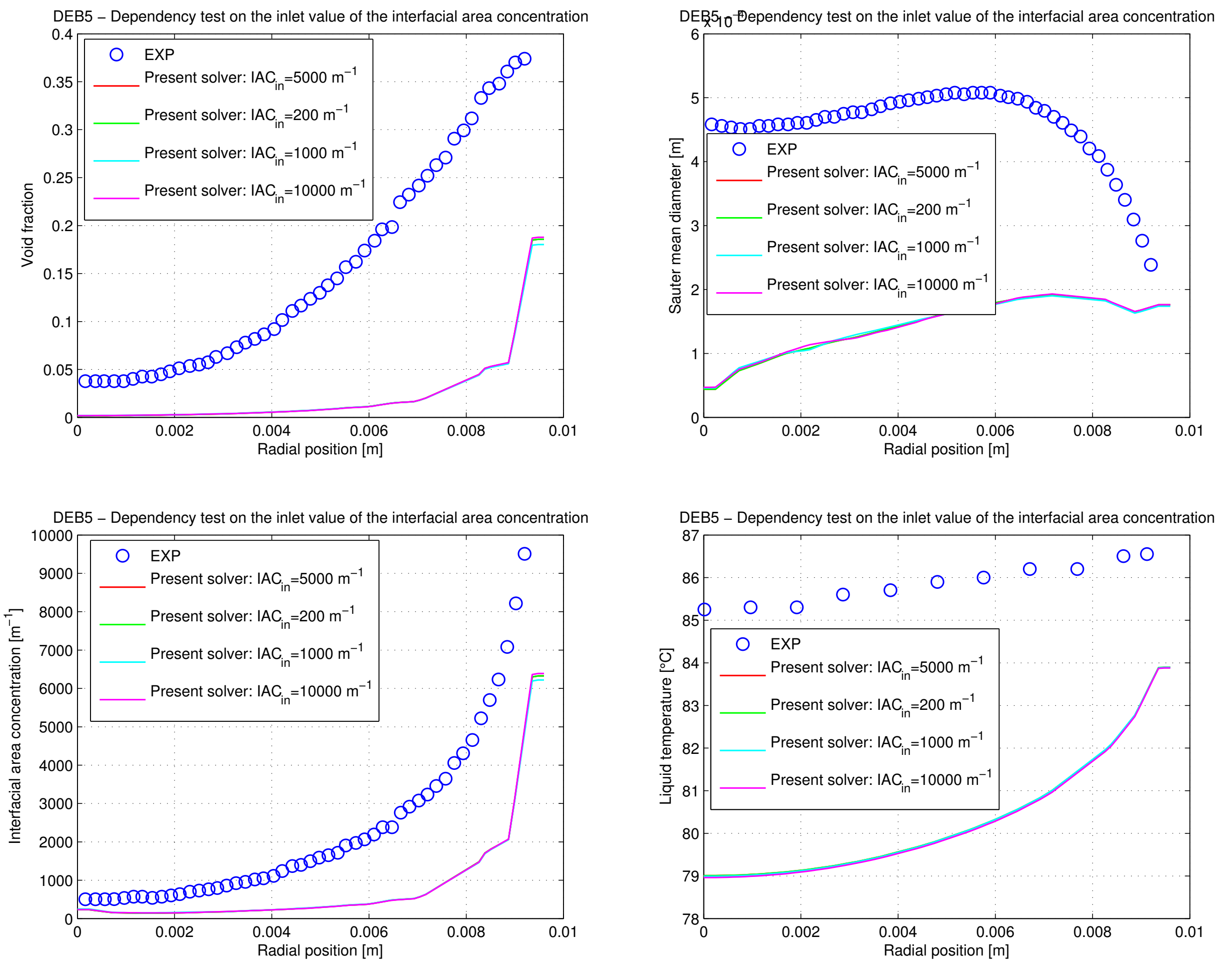

Figure 4.13: Comparison of the results obtained from *myTwoPhaseEulerFoamBoiling* at the end of the heated section for DEB5 with varying inlet interfacial area concentration

CHAPTER 5 DISCUSSION

5.1 On the *myTwoPhaseEulerFoamAdiabatic* solver

Even if the results given by this solver are satisfactory, it should be validated on a much wider range of experimental data with various flow conditions and using various geometries such as annulus, which would first require a new modeling of the wall lubrication force since the Tomiyama model only applies to pipe. In addition, one should also make sure that the results remain consistant when the solver is applied to three-dimensional simulations.

The models used for the interfacial forces could be improved. The wall lubrication force model according to Tomiyama has two main inconveniencies: it is only applicable to pipes and it depends on their diameter. An option would be to implement the Frank model [11] in order to have a formulation with a wider range of applicability which would in the same time only depend on the distance to the wall. The virtual mass force has been implemented in this work but not activated in the momentum equation because it turned out to unstabilise the solver. This issue should be investigated too. However, it is acceptable not to consider the virtual mass force since it reflects the influence of the accelerating fluid around a bubble and therefore vanishes when the steady-state is reached.

In order to better reproduce the intermediate void fraction profile, as the one observed at the higher measuring section in DEDALE1101 ($y/D = 155$), one shoud probably adopt another approach for the interfacial area concentration. The present model is based on the assumption that there is only one possible size for bubbles at a given location in the pipe. Further research has been conducted on so called Multiple-Size-Group (MUSIG) approaches that allow to have different bubble sizes and therefore having as many transport equations as considered bubble sizes. With only two groups, Ishii and Kim [15] obtained encouraging results compared to the present approach.

Other possible improvements concern the turbulence modeling. Instead of combining the Sato model with a classic single phase k-ϵ turbulence model, one could follow the method described by Morel [25] and implement a modified k-ϵ model taking into account the turbulence caused by the bubbles on the liquid phase through additional source terms.

5.2 On the *myTwoPhaseEulerFoamBoiling* solver

As explained earlier, a great challenge and an issue which still remain to be resolved in this solver is how to properly model the heat flux density received by the liquid from the wall and implement it in a grid-independent manner. Imposing a uniform heat flux density may seem like a very simple problem in CFD because in most of the CFD software packages, the user just has to select the corresponding feature without having to think about the way it is solved numerically.

A way to overcome this problem related to the treatment of the wall heat flux density could be to implement a dimensionless temperature wall function. Instead of adding a source term accounting for the wall heat in the energy equation for near-wall cells, one may calculate the temperature in the node of the near-wall cells through a dimensionless temperature wall function, which would depend on the wall heat flux density. A possible formulation is given by Kader [18]. This would at least solve the very high sensitivity of the solver to the grid resolution.

A related issue is how the boundary condition for the liquid enthalpy at the wall should be applied properly. The choice of a *zeroGradient* used in the simulations is questionable. In OpenFOAM, it might be common to impose a uniform enthalpy gradient at the wall, according to the Fourier law, implying that conduction is the main heat transfer mechanism between the wall and the liquid. But this in not valid in our case since the heat transfer phenomenon is governed by other mechanisms, as can be seen from the heat flux partitioning. However, tests have been performed imposing a non-zero uniform gradient at the wall and the results were almost identical.

CHAPTER 6

CONCLUSIONS

According to many experts, nuclear power will play an important role in tomorrow's energy supply. This opinion is essentially motivated by global warming and the depletion of fossil resources. However, this technology still faces sometimes the public opinion's reluctancy based mostly on safety issues and storage of nuclear waste. A few safety issues in the nuclear industry are related to subcooled nucleate boiling. This phenomenon designates the first boiling regime: liquid evaporates at a heated wall, generating bubbles which then migrate to the subcooled liquid and condensate, heating the latter up. This regime normally occurs in the lower part of BWR fuel bundles and could also occur at the highest part of a PWR core. For a heat flux equal to the Critical Heat Flux (CHF), one leaves the nucleate boing regime. This transition is called Departure From Nucleate boiling and marks a very sudden increase in the fuel clad temperature that may lead to structural damage and consequently to infringement to the defence-in-depth principle. Another reason why one would like to be able to predict this phenomenon more accurately is that the void generated in a BWR core will affect the reactivity of the reactor, and in a PWR it leads to a so-called axial offset anomaly due to the deposition of lithium borate. Therefore, accurate models of subcooled nucleate boiling are of major interest. So far, the codes used for this purpose are old system level codes and there is a crucial need of improved three-dimensional models in CFD.

In this study, subcooled nucleate boiling in vertical pipes has been modeled within the OpenFOAM framework. OpenFOAM is a relatively new CFD software package, quite different from the most common commercial codes. Its user-friendliness is limited since it does not contain any graphical interface. Besides, the number of existing models that comes with it is limited. However, this code has a huge potential. The implementation of equations is transparent and the source code is rather easily understandable, and as it open source, the code is modifiable, allowing the user to implement customised models. Therefore, this code can continuously be improved by the users community and does not rely on any release by a company.

The modelling of subcooled nucleate boiling has been carried out in two steps. In the first step, a solver named *myTwoPhaseEulerFoamAdiabatic* has been implemented to simulate adiabatic bubbly flow using a Eulerian-Eulerian approach. In this solver, the mass and momentum conservation equations are solved for both phases and an interfacial area transport equation has been implemented. Additionally, the turbulence has been neglected in the gas phase, and modeled for the liquid phase using a classic k-ϵ model in combination with standard wall functions. A bubble induced turbulent viscosity has been implemented according to the proposed model by Sato. Special care has been taken for the interfacial forces and the following models have been chosen: the Ishii-Zuber drag, the Tomiyama lift, the Tomiyama wall lubrication and the Gosman turbulent dispersion forces. The Zuber virtual mass force has been implemented but it was not activated while solving the momentum equations in the present studies. Furthermore, two different models, namely the Hibiki and Ishii model as well as the Yao and Morel model, have been implemented for the coalescence and breakup source terms appearing in the interfacial area concentration transport equation. Either one or the other can be selected by the user.

Then, a solver called *myTwoPhaseEulerFoamBoiling* has been developed based on the previous one. In order to take into account nucleation at the heated pipe wall and condensation in the subcooled liquid, while assuming that the gas remains at the saturated conditions, only one energy conservation equation has been included - the one for the liquid phase. Furthermore phase change terms have been introduced in the mass- and momentum conservation equations, as well as in the interfacial area concentration transport equation.

The turbulence modeling has not been modified.

The solver *myTwoPhaseEulerFoamAdiabatic* has been tested and validated again two data sets extracted from the DEDALE experimental database. The results are satisfactory since the void fraction wall peak observed experimentally is captured and the predicted values both at the peak and in the core of the pipe are reasonably accurate. The calculated position of the peak could however be improved. It is currently slightly too far from the wall but this might be related to the profiles applied at the inlet. In the simulations, they have been chosen uniform because there is a lack of information concerning the radial profiles at the inlet and only cross-section average values could be found in the literature. However in reality, the flow will be partially developed. The results also show that the Hibiki and Ishii model for the coalescence and breakup terms appearing in the interfacial area equation leads to more accurate results than the Yao and Morel model at intermediate channel heights but it cannot capture the axial evolution of the radial profiles. It is believed that this is related to the fact that the numerical constants used by Yao and Morel were derived analitycally, whereas those used by Hibiki and Ishii were fitted on experimental data.

With regards to the comparison between the results obtained from the *myTwoPhaseEulerFoamBoiling* solver and those from the experiment, one must admit that some further improvements have to be done. The trends of the distributions of many parameters are physically reasonable, as for example, the temperature increases axially and decreases radially away fromm the wall, but the results are too inaccurate. The boiling sub-model is believed to be properly implemented because the key parameter, the bubble detachment diameter, is predicted with a relatively good precision. It has also been tested (but not shown in this report) to remove the condensation rate but the void fraction predictions were still far too low. Therefore, it is quite plausible that the reason for these discrepancies lies in the energy equation. This hypothesis is strengthened by looking at the void fraction and liquid temperature predictions, which are highly underestimated. Since the rise in kinetic energy of the fluid between the inlet and the outlet is not so important, this implies that part of the energy flux from the wall is lost. This issue regarding the proper treatment of the wall heat flux density for near-wall cells is the origin of the grid sensitivity which has been noticed. However, the work which has been done on *myTwoPhaseEulerFoamBoiling* remains valuable. The solvers code is stable, the solution procedure is well defined and its architecture properly constructed with different classes for different physical entities such as interfacial forces or coalescence and breakup mechanisms. Besides, its robustness has been proven by investigating its insensitivity to the inlet gas velocity and the inlet interfacial area concentration, which do not have any physical significance when water is subcooled.

There are several aspects that need to be addressed in future work on this topic. In order to improve the *myTwoPhaseEulerFoamAdiabatic* solver, one could first focus on the interfacial forces. Using the Frank [11] wall lubrication model would make this force valid for other geometries than pipes and make this force geometry-independent. Then considering a Multi-Size-Group (MUSIG), as was done by Ishii and Kim [15] for the modeling of the bubble transport phenomena instead of the one-group interfacial area concentration equation would perhaps capture the axial evolution of the parameters radial profiles and give better predictions of the bubbly-to-slug transition. Finally, modified k-ϵ equations dedicated to two-phase flows could be implemented, as for instance the one described by Morel [25].

Regarding *myTwoPhaseEulerFoamBoiling*, whose results are not as satisfactory, the challenge is most certainly how to properly implement the heat flux applied to the near-wall cells in a grid-independent manner. The approach described by Kader [18] based on using a temperature wall function for near wall cells could be a solution. Therefore the heat flux should probably be removed from the enthalpy equation and this temperature wall function used for the calculation of the liquid enthalpy in the near-wall cells instead. Besides one should investigate the boundary condition to apply to the enthalpy at the wall.

BIBLIOGRAPHY

[1] H. Anglart. *Thermal-Hydraulics in Nuclear Systems* (KTH, Stockholm, 2008).

[2] Inc. ANSYS. *ANSYS CFX-Solver Theory Guide*, 12.1 edn., November 2009.

[3] S.P. Antal et al. *Analysis of phase distribution in fully developed laminar bubbly two-phase flow. International Journal of Multiphase Flow*, 7:pp. 635–652, 1991.

[4] D. Bestion and P. Peturaud. *Deliverable D2.2.1: Review of the existing data basis for the validation of models for CHF.* 6th EURATOM framework programme, NURESIM-SP2-TH, 2006.

[5] J. Boree et al. *Ecoulements diphasiques eau-vapeur avec changement de phase.* Tech. rep., Rapport intermediaire IMFT- Interface, 1995.

[6] J. Boussinesq. *Théorie de l'écoulement tourbillonnant et tumultueux des liquides dans les lits rectilignes à grandes sections* (Gauthier-Villars, Paris, 1897).

[7] W.C. Ceumern-Lindenstjerna. *Bubble departure diameter and release frequencies during nucleate pool boiling of water and aqueous nacl solutions. Heat Transfer in boiling*, 1977.

[8] J.M. Delhaye. *Some issues related to the modelling of interfacial areas in gas–liquid flows, part ii: modelling the source terms for dispersed flows. C. R. Acad. Sci. Paris*, t. 329(Série II b):pp. p. 473–486, 2001.

[9] M. DelValle and D.B.R Kenning. *Subcooled flow boiling at high heat flux. International Journal of Heat and Mass Transfer*, 28:pp. 1907–1920, 1985.

[10] D.A. Drew and R.T. Lahey. *The virtual mass and lift force on a sphere in rotating and straining inviscid flow. International Journal of Multiphase Flow*, 13:pp. 113–121, 1987.

[11] Th. Frank. *Advances in computational fluid dynamics (cfd) of 3-dimensional gas–liquid multiphase flows.* In *NAFEMS Seminar "Simulation of Complex Flows (CFD)"*, pp. 1–18 (Wiesbaden, Germany, 2005).

[12] A.D. Gosman et al. *Multidimensional modeling of turbulent two-phase flow in stirred vessels. AIChE Journal*, 38:pp. 1946–1956, 1992.

[13] T. Hibiki and M. Ishii. *Development of one-group interfacial area transport equation in bubbly flow systems. International Journal of Heat and Mass Transfer*, 45:pp. 2351–2372, 2002.

[14] M. Ishii and S. Kim. *Micro four-sensor probe measurement of interfacial area transport for bubbly flow in round pipes. Nuclear Engineering and Design*, 205:pp. 2711–2726, 2001.

[15] M. Ishii and S. Kim. *Development of one-group and two- group interfacial area transport equation. Nuclear Science and Engineering*, 3:pp. 257–273, 2004.

[16] W.H. Jens and P.A. Lottes. *Analysis of heat transfer burnout, pressure drop, and density data for high-pressure water.* Tech. rep., Argonne National Laboratories, 1951.

[17] M. Jischa and H.B. Rieke. *About the prediction of turbulent prandtl and schmidt numbers from modeled transport equations. International Journal of Heat and Mass Transfer*, 22(11):pp. 1547–1555, 1979. ISSN 0017-9310. doi:DOI:10.1016/0017-9310(79)90134-0.

[18] B.A. Kader. *Temperature and concentration profiles in fully turbulent boundary layers. International Journal of Heat and Mass Transfer*, 24:pp. 1541–1544, 1981.

[19] N. Kurul and M.Z. Podowski. *On the modeling of multidimensional effects in boiling channels. Proceedings of the 27th National Heat Transfer Conference*, June 1991.

[20] M. Lemmert and J.M Chawla. *Influence of flow velocity on surface boiling heat transfer coefficient. Heat Transfer in Boiling*, pp. 237–247, 1977.

[21] OpenCFD Limited. *OpenFOAM Programmer's C++ Documentation.* http://foam.sourceforge.net/doc/Doxygen/html/, 2011.

[22] OpenCFD Limited. *OpenFOAM User Guide.* http://www.openfoam.com/docs/user/, 2011.

[23] M. Lopez De Bertodano and D. Prabhudharwadkar. *Computational fluid dynamics*, pp. 420–444. CFD Two Fluid Model for Adiabatic and Boiling Bubbly Flows in Ducts (InTech, 2010).

[24] E. Manon. *Contribution à lanalyse et à la modélisation des écoulements bouillants sous-saturés dans les conditions des Réacteurs à Eau sous Pression en présence d'ébullition.* Ph.D. thesis, Ecole Centrale Paris, 2000.

[25] C. Morel. *An order of magnitude analysis of the two-phase k–e model. International Journal of Fluid Mechanics Research*, 22:pp. 21–44, 1995.

[26] H. Rusche. *Computational fluid dynamics of dispersed two-phase flows at high phase fractions.* Ph.D. thesis, Imperial College, London, 2002.

[27] Y. Sato et al. *Momentum and heat transfer in two-phase bubbly flow. International Journal of Multiphase Flow*, 7:pp. 167–178, 1981.

[28] B. Stroustrup. *The C++ programming language* (Addison-Wesley, 1991), second edition edn. ISBN 0-201-53992-6.

[29] X. Pedruelo Tapia. *Modeling of wind flow over complex terrain using OpenFOAM.* Master's thesis, University of Gävle, Sweden, 2009.

[30] R. Thiele. *Direct Contact Condensation with OpenFOAM.* Master's thesis, KTH, Stockholm, 2010.

[31] J.R.S. Thom et al. *Boiling in subcooled water during flow up heated tubes or annuli.* In *Symposium on Boiling Heat Transfer in Steam Generating Units and Heat Exchangers* (Institute of Mechanical Engineers, London, 1965).

[32] V.I. Tolubinsky and D.M. Kostanchuk. *Vapor bubble growth rate and heat transfer intensity at subcooled water boiling.* In *4th International Heat Transfer Conference* (1970).

[33] A. Tomiyama. *Struggle with computational bubble dynamics. Third International Conference on Multiphase Flow*, 1998.

[34] T.Salnikova. *TWO-PHASE CFD ANALYSES IN FUEL ASSEMBLY SUB-CHANNELS OF PRESSURIZED WATER REACTORS UNDER SWIRL CONDITIONS.* Ph.D. thesis, Technische Universität Dresden, 2008.

[35] H.C. Unal. *Maximum bubble diameter, maximum bubble-growth rate during the sub- cooled nucleate flow boiling of water up to 17.7 mn/m2. International Journal of Heat and Mass Transfer*, 19:pp. 643–649, 1976.

[36] R.M. Wellek et al. *Shapes of liquid drops moving in liquid media. AIChE Journal*, 12:pp. 854–860, 1966.

[37] K. Wolfert. *Non-equilibrium mass transfer between liquid and vapor phases during depressurization processes in transient two-phase flow.* In *Proc. 2nd CSNI Specialists meeting*, vol. 2, pp. 1377–1387 (Paris, 1978).

[38] Q. Wu et al. *One-group interfacial area transport in vertical bubble flow. International Journal of Heat and Mass Transfer*, 41:pp. 1103–1112, 1998.

[39] W. Yao and C. Morel. *Volumetric interfacial area prediction in upward bubbly two-phase flow. International Journal of Heat and Mass Transfer*, 47:pp. 307–328, 2004.

[40] N. Zuber. *On the dispersed two-phase flow in the laminar flow regime. Chemical Engineering Science*, 19(11):pp. 897–917, 1964. ISSN 0009-2509. doi:DOI:10.1016/0009-2509(64)85067-3.

APPENDIX A INSTALLATION GUIDELINES

Here are a few guidelines which will help you to install the new solvers *myTwoPhaseEulerFoamAdiabatic* and *myTwoPhaseEulerFoamBoiling* and use them on the provided cases. These guidelines are given for *myTwoPhaseEulerFoamAdiabatic* but the procedure is identical for *myTwoPhaseEulerFoamBoiling*

1) Copying the cases

- Enter the folder *run* included on the attached DVD
- Copy the folder *myTwoPhaseEulerFoamAdiabatic* (which contains the different cases on which the solver can be tested) and paste it in the run directory of your local workspace (in my case for instance: /home/michta/OpenFOAM/michta-1.7.x/run). If this run directory does not exist yet, create it.

2) Copying the solver

- Enter the folder *solvers* included on the attached DVD
- Copy the folder *myTwoPhaseEulerFoamAdiabatic* (which contains the solver) and paste it in the solvers directory of your local workspace (in my case for instance: /home/michta/OpenFOAM/michta-1.7.x/applications/solvers). If this solvers directory does not exist yet, create it.

3) Libraries linking and compiling

- Open a terminal that you will use from now on
- Go into the *myTwoPhaseEulerFoamAdiabatic* solver folder of your local workspace (in my case : /home/michta/OpenFOAM/michta-1.7.x/applications/myTwoPhaseEulerFoamAdiabatic) and perform successively the following steps:
  - Go into the *phaseModel* folder and compile it using the command *wmake libso* (it is essential to compile this submodel first!)
  - Go into the *breakupAndCoalescenceModels* folder and compile it using the command *wmake libso*
  - Go into the *interfacialModels* folder and compile it using the command *wmake libso*
- Being located in your *myTwoPhaseEulerFoamAdiabatic* solver folder, compile it by using the command *wmake*.

4) Running a case

- Still using the terminal, locate the folder of the case you want to run
- Then simply type the name of the solver you want to use for this case in the terminal. Here we will just use the command *myTwoPhaseEulerFoamAdiabatic*.

5) Postprocessing

- In order to postprocess your simulation, using ParaView, you must first be located (still using the terminal) in the folder of the case you want to postprocess
- Then simply use the command *paraFoam*.

APPENDIX B SOURCE CODE: MAIN PROGRAM

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2010 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    myTwoPhaseEulerFoamBoiling

Author
    Edouard Michta
    michta@kth.se

Description
    Solver for modeling of subcooled nucleate boiling

    Solver based on twoPhaseEulerFoam, accounting for phase change due to
    nucleation at a heated wall and subsequent condensation in the bulk liquid

    Equations solved:
    - mass and momentum conservation equation for both phases
    - energy equation for the liquid phase
    - one-group interfacial area concentration transport equation
    - if ""turbulence"" is switched on: RANS classic k-epsilon model + standard
    wall functions

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "nearWallDist.H"
#include "wallFvPatch.H"
#include "Switch.H"

#include "IFstream.H"
#include "OFstream.H"

#include "phaseModel.H"
#include "dragModel.H"
#include "liftModel.H"
#include "virtualMassModel.H"
#include "turbulentDispersionModel.H"
#include "wallLubricationModel.H"
#include "breakupModel.H"
#include "coalescenceModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createMesh.H"
    #include "readGravitationalAcceleration.H"
    #include "readEnvironmentalProperties.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"
    #include "readTimeControls.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTwoPhaseEulerFoamControls.H"
        #include "CourantNos.H"
        #include "setDeltaT.H"

        runTime++;
        Info<< "Time = " << runTime.timeName() << nl << endl;

        #include "condensationModel.H"
        #include "evaporationModel.H"
        #include "alphaEqn.H"
        #include "IACEqn.H"

// --- Outer-corrector loop
        for (int oCorr=0; oCorr<nCorr; oCorr++)
        {

            #include "interMomentumForces.H"
```

```
              #include "UEqns.H"

            // --- PISO loop
            for (int corr=0; corr<nCorr; corr++)
            {
                  #include "pEqn.H"
                if (correctAlpha && corr<nCorr-1)
                {
                      #include "alphaEqn.H"
                      #include "IACEqn.H"
                }
            }

              #include "DDtU.H"
              #include "kEpsilon.H"
              #include "HEqns.H"
        }

          #include "write.H"

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}


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

APPENDIX C

SOURCE CODE: ISHII-ZUBER DRAG

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "Ishii.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(Ishii, 0);

    addToRunTimeSelectionTable
    (
        dragModel,
        Ishii,
        dictionary
    );
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::Ishii::Ishii
(
    const dictionary& interfaceDict,
    const volScalarField& alpha,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    dragModel(interfaceDict, alpha, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::Ishii::~Ishii()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::Ishii::AD
(
    const volScalarField& Ur,
    const volScalarField& DS

) const
{

    dimensionedScalar plusDS
    (
        "plusDS",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
        scalar (1.0e-6)
    );

    scalar alphaMax = 0.52;

    dimensionedScalar muStar = (phasea_.rho()*phasea_.nu()+0.4*phaseb_.rho()
        *phaseb_.nu())/(phasea_.rho()*phasea_.nu()+phaseb_.rho()*phaseb_.nu());
    volScalarField nuMix = phaseb_.nu()*pow((scalar(1.0)-alpha_/alphaMax),
                            -2.5*alphaMax*muStar);
    volScalarField ReMix = max(Ur*DS/nuMix, scalar(1.0e-3));
    volScalarField Cd = 24.0*(scalar(1) + 0.15*pow(ReMix, 0.687))/(ReMix+0.001);

    forAll(ReMix, celli)
    {
        if(ReMix[celli] > 1000.0)
        {
            Cd[celli] = 0.44;
        }
    }

    return 0.75*Cd*phaseb_.rho()*Ur/(DS+plusDS);
}

```

```

// ********************************************************************************* //

```

APPENDIX D

SOURCE CODE: TOMIYAMA LIFT

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "liftTomiyama.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(liftTomiyama, 0);

    addToRunTimeSelectionTable
    (
        liftModel,
        liftTomiyama,
        dictionary
    );
```

```
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::liftTomiyama::liftTomiyama
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    liftModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::liftTomiyama::~liftTomiyama()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::liftTomiyama::AL
(
    const volScalarField& Ur,
    const volScalarField& alpha,
    const volScalarField& DS

) const
{

//-------- gravity vector --------

      dimensionedScalar g
      (
          "g",
          dimensionSet(0, 1, -2, 0, 0, 0, 0),
      scalar (9.81)
      );

//--------------------------
     volScalarField Red = (Ur*DS)/phaseb_.nu();
     volScalarField Eo = mag(g)*(phaseb_.rho()-phasea_.rho())*pow(DS,2.0)/phaseb_.sig();
     volScalarField dH = DS*pow((scalar(1.0)+scalar(0.163)*pow(Eo,0.757)),0.333);
     volScalarField EodH = mag(g)*(phaseb_.rho()-phasea_.rho())*pow(dH,2.0)/phaseb_.sig();
     volScalarField fEodH = scalar(0.00105)*pow(EodH,3.0)-scalar(0.0159)*pow(EodH,2.0)
                           -scalar(0.0204)*EodH+scalar(0.474);
     volScalarField Cl = fEodH;
//--------------------------
    forAll(EodH, celli)
    {
        if (EodH[celli] >= 10.0)
        {
            Cl[celli] = -0.27;
        }

        else if (EodH[celli] < 4.0)
        {
            Cl[celli] = min(scalar(0.288)*tanh((scalar(0.121)*Red[celli])),fEodH[celli]);
        }
    }

return  Cl*phaseb_.rho();

}

// ************************************************************************* //

```

APPENDIX E SOURCE CODE: TOMIYAMA WALL LUBRICATION

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "wallLubricationTomiyama.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(wallLubricationTomiyama, 0);

    addToRunTimeSelectionTable
    (
        wallLubricationModel,
        wallLubricationTomiyama,
        dictionary
    );
```

```
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::wallLubricationTomiyama::wallLubricationTomiyama
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    wallLubricationModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::wallLubricationTomiyama::~wallLubricationTomiyama()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::wallLubricationTomiyama::AWL
(
    const volScalarField& alpha,
    const volScalarField& DS,
    const volScalarField& Ur

) const
{

    dimensionedScalar g
    (
        "g",
        dimensionSet(0, 1, -2, 0, 0, 0, 0),
        scalar (9.81)
    );

    volScalarField Eo = mag(g)*(phaseb_.rho()-phasea_.rho())*pow(DS,2.0)/phaseb_.sig();
    volScalarField CwlEo = exp(-0.933*Eo+0.179);

    forAll(Eo, celli)
    {
        if (Eo[celli] < 1.0)
        {
            CwlEo[celli] = 0.47;
        }
        else if (Eo[celli] >= 1.0 && Eo[celli] <= 5.0)
        {
            CwlEo[celli] = exp(-0.933*Eo[celli]+0.179);
        }
        else if (Eo[celli] > 5.0 && Eo[celli] <= 33.0)
        {
            CwlEo[celli] = 0.00599*Eo[celli]-0.0187;
        }
        else if (Eo[celli] > 33.0)
        {
            CwlEo[celli] = 0.179;
        }

    }

    return 0.5*CwlEo*phaseb_.rho()*pow(Ur,2.0)*DS;
            //*pos(alpha-scalar(0.0001));
}


// ************************************************************************* //

```

APPENDIX F SOURCE CODE: GOSMAN TURBULENT DISPERSION

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "Gosman.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(Gosman, 0);

    addToRunTimeSelectionTable
    (
        turbulentDispersionModel,
        Gosman,
        dictionary
    );
```

```
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::Gosman::Gosman
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    turbulentDispersionModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::Gosman::~Gosman()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::Gosman::ATD
(
    const volScalarField& Ur,
    const volScalarField& nutb,
    const volScalarField& DS,
    const volScalarField& alpha

) const
{

    scalar sigmat = 0.9;

    dimensionedScalar plusDS
    (
        ”plusDS”,
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
        scalar (1.0e-6)
    );

    scalar alphaMax = 0.52;

//If calculated from Ishii drag

    dimensionedScalar muStar = (phasea_.rho()*phasea_.nu()+0.4*phaseb_.rho()*phaseb_.nu())
                        /(phasea_.rho()*phasea_.nu()+phaseb_.rho()*phaseb_.nu());
    volScalarField nuMix = phaseb_.nu()*pow((scalar(1.0)-alpha/alphaMax),
                                 -2.5*alphaMax*muStar);
    volScalarField ReMix = max(Ur*DS/nuMix, scalar(1.0e-3));
    volScalarField Cd = 24.0*(scalar(1) + 0.15*pow(ReMix, 0.687))/(ReMix+0.001);

    forAll(ReMix, celli)
    {
        if(ReMix[celli] > 1000.0)
        {
            Cd[celli] = 0.44;
        }
    }


return  scalar(0.75)*Cd/(DS+plusDS)*phaseb_.rho()*nutb/sigmat*Ur;

}



// ************************************************************************* //

```

APPENDIX G SOURCE CODE: ZUBER VIRTUAL MASS

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "Zuber.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(Zuber, 0);

    addToRunTimeSelectionTable
    (
        virtualMassModel,
        Zuber,
        dictionary
    );
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::Zuber::Zuber
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    virtualMassModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::Zuber::~Zuber()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::Zuber::AVM
(
    const dimensionedScalar& Cvm,
    const volScalarField& alpha

) const
{

return Cvm*(scalar(1.0)+2*alpha)/(scalar(1.0)-alpha)*phaseb_.rho();

}



// ************************************************************************* //

```

APPENDIX H SOURCE CODE: HIBIKI-ISHII COALESCENCE MODEL

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "HibikiIshiiC.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(HibikiIshiiC, 0);

    addToRunTimeSelectionTable
    (
        coalescenceModel,
        HibikiIshiiC,
        dictionary
    );
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::HibikiIshiiC::HibikiIshiiC
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    coalescenceModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::HibikiIshiiC::~HibikiIshiiC()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::HibikiIshiiC::C
(
    const volScalarField& alpha,
    const volScalarField& IAC,
    const volScalarField& epsilon,
    const volScalarField& Ur

) const
{

    scalar GammaC = 0.005;
    scalar Kc = 1.29;
    scalar alphaMax = 0.52;

    volScalarField dBuff = 6.0*alpha/IAC;

    dimensionedScalar plusDS
    (
        "plusDS",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
        scalar (1.0e-6)
    );

    volScalarField sourceRCa = - GammaC*pow(alpha,2.0)*pow(epsilon,1.0/3.0);
    volScalarField sourceRCb = pow(dBuff+plusDS,11.0/3.0)*(alphaMax-alpha);
    volScalarField sourceRCc = exp(-Kc*pow(phaseb_.rho(),1.0/2.0)
        *pow(dBuff+plusDS,5.0/6.0)*pow(epsilon,1.0/3.0)/pow(phaseb_.sig(),1.0/2.0));
    volScalarField sourceRC = sourceRCa/sourceRCb*sourceRCc;

    return sourceRC*scalar(1.0);
}


// ************************************************************************* //

```

APPENDIX I

SOURCE CODE: HIBIKI-ISHII BREAKUP MODEL

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "HibikiIshii.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(HibikiIshii, 0);

    addToRunTimeSelectionTable
    (
        breakupModel,
        HibikiIshii,
        dictionary
    );
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::HibikiIshii::HibikiIshii
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    breakupModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::HibikiIshii::~HibikiIshii()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::HibikiIshii::B
(
    const volScalarField& alpha,
    const volScalarField& IAC,
    const volScalarField& epsilon,
    const volScalarField& Ur

) const
{

    scalar GammaB = 0.005;
    scalar Kb = 1.37;
    scalar alphaMax = 0.52;

    volScalarField dBuff = 6.0*alpha/IAC;

    dimensionedScalar plusDS
    (
        "plusDS",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
        scalar (1.0e-6)
    );

    volScalarField sourceTIa = GammaB*alpha*(1.0-alpha)*pow(epsilon,1.0/3.0);
    volScalarField sourceTIb = pow(dBuff+plusDS,11.0/3.0)*(alphaMax-alpha);
    volScalarField sourceTIc = exp(-Kb*phaseb_.sig()/phaseb_.rho()
                                 /pow(dBuff+plusDS,5.0/3.0)/pow(epsilon,2.0/3.0));
    volScalarField sourceTI = sourceTIa/sourceTIb*sourceTIc;


    return sourceTI*scalar(1.0);
}


// ************************************************************************* //

```

APPENDIX J

SOURCE CODE: YAO-MOREL COALESCENCE MODEL

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "YaoMorelC.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(YaoMorelC, 0);

    addToRunTimeSelectionTable
    (
        coalescenceModel,
        YaoMorelC,
        dictionary
    );
```

```
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::YaoMorelC::YaoMorelC
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    coalescenceModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::YaoMorelC::~YaoMorelC()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::YaoMorelC::C
(
    const volScalarField& alpha,
    const volScalarField& IAC,
    const volScalarField& epsilon,
    const volScalarField& Ur

) const
{

    scalar Kc1 = 2.86;
    scalar Kc2 = 1.922;
    scalar Kc3 = 1.017;
    scalar alphaMax = 0.52;

    dimensionedScalar plusDS
    (
        "plusDS",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
    scalar (0.000001)
    );

    volScalarField dBuff = 6.0*alpha/IAC;

    volScalarField We = 2.0*phaseb_.rho()*pow(epsilon*(dBuff+plusDS),2.0/3.0)
                    *(dBuff+plusDS)/phaseb_.sig();
    scalar WeCr = 1.24;
    volScalarField gAlpha = (pow(alphaMax,1.0/3.0)-pow(alpha,1.0/3.0))
                         /pow(alphaMax,1.0/3.0);

    volScalarField sourceRCa = -Kc1*pow(epsilon,1.0/3.0)*pow(alpha,2.0);
    volScalarField sourceRCb = pow((dBuff+plusDS),(11.0/3.0))
                            *(gAlpha + Kc2*alpha*sqrt(We/WeCr));
    volScalarField sourceRCc = exp(-Kc3*sqrt(We/WeCr));
    volScalarField sourceRC = sourceRCa/sourceRCb*sourceRCc;

    return sourceRC*scalar(1.0);
}


// ************************************************************************* //

```

APPENDIX K SOURCE CODE: YAO-MOREL COALESCENCE MODEL

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*---------------------------------------------------------------------------*/

#include "YaoMorel.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(YaoMorel, 0);

    addToRunTimeSelectionTable
    (
        breakupModel,
        YaoMorel,
        dictionary
    );
```

```
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::YaoMorel::YaoMorel
(
    const dictionary& interfaceDict,
    const phaseModel& phasea,
    const phaseModel& phaseb
)
:
    breakupModel(interfaceDict, phasea, phaseb)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::YaoMorel::~YaoMorel()
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::YaoMorel::B
(
    const volScalarField& alpha,
    const volScalarField& IAC,
    const volScalarField& epsilon,
    const volScalarField& Ur

) const
{

    scalar Kb1 = 1.6;
    scalar Kb2 = 0.42;
    scalar alphaMax = 0.52;

    dimensionedScalar plusDS
    (
        "plusDS",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
        scalar (0.000001)
    );

    volScalarField dBuff = 6.0*alpha/IAC;

    volScalarField We = 2.0*phaseb_.rho()*pow(epsilon*(dBuff+plusDS),2.0/3.0)
                        *(dBuff+plusDS)/phaseb_.sig();
    scalar WeCr = 1.24;

    volScalarField gAlpha = (pow(alphaMax,1.0/3.0)-pow(alpha,1.0/3.0))
                            /pow(alphaMax,1.0/3.0);

    volScalarField sourceTIa = Kb1*pow(epsilon,1.0/3.0)*alpha*(1.0-alpha);
    volScalarField sourceTIb = pow((dBuff+plusDS),(11.0/3.0))*(1.0+Kb2*(1.0-alpha)
                                *sqrt(We/WeCr));
    volScalarField sourceTIc = exp(-WeCr/We);
    volScalarField sourceTI = sourceTIa/sourceTIb*sourceTIc;

    return sourceTI*scalar(1.0);
}


// ************************************************************************* //

```

APPENDIX L

SOURCE CODE: FORTRAN CODE IMPLEMENTED BY KTH-NRT IN CFX

```
      PROGRAM SBCOOL
      DIMENSION TSUPV(50)
      DIMENSION TSUBV(50), Q1FV(50), QQV(50), QEV(50)
      DIMENSION FREQV(50), ASDV(50), DMV(50), ERATEV(50)
      DIMENSION A1FV(50), A2FV(50), H1FV(50), HQV(50), H2FV(50)
C
      COMMON /HEAT1/RHOV,HLV,CONL,ALFA,ACONS,B,DMCON,H1F,QW
      COMMON /HEAT2/A1,A2,DM,DMI,FREQ,ASD,HQ,H2F,A2F,A1F,ERATE
      COMMON /ASDD/ASD1,ASD2,FLAGD
      LOGICAL FLAGD
C
C      DRIVER PROGRAM FOR THE SUBCOOLED BOILING MODEL
C
      Q2PR = 570.E3      ! Wall heat flux
      TSUB = 20.         ! Liquid subcooling
      VEL = 900./787.61  ! Liquid velocity
C
C      Below the subcooled boiling model is called
C      for various values of subcoolings
C
      TSUB = -1.0
      DO I=1,50
       TSUBV(I) = TSUB + 1.0
       TSUB = TSUBV(I)
       CALL SBCMOD (TSUP,VEL,TSUB,Q1F,QQ,QE,Q2PR)
C
C       Parameters returned by subcooled boiling
C       model (through list of parameters and common
C       blocks)
C
       TSUPV(I) = TSUP    ! wall superheat
       Q1FV(I) = Q1F      ! convective heat
       QQV(I)  = QQ       ! quenching heat
       QEV(I)  = QE       ! evaporation heat
       FREQV(I) = FREQ    ! frequency of detachment
       ASDV(I) = ASD      ! nucleation site density
       DMV(I) = DM        ! diameter of bubble
       ERATEV(I) = ERATE ! evaporation rate
       A1FV(I) = A1F      ! area fraction
       A2FV(I) = A2F      ! area fraction
```

```
        H1FV(I) = H1F
        HQV(I) = HQ
        H2FV(I) = H2F
       ENDDO
C
C
       open(1134,file='original.dat')
       DO I=1,50
        WRITE(1134,'(1X,14(E12.5,1X))') TSUBV(I), TSUPV(I),
      : Q1FV(I), QQV(I), QEV(I), FREQV(I), ASDV(I), DMV(I),
      : ERATEV(I), A1FV(I), A2FV(I), H1FV(I), HQV(I),
      : H2FV(I)
       ENDDO
       close(1134)
C
       END
C   +++++++++++++++++++++++++++++++++++++++++++++++++++++
       SUBROUTINE  SBCMOD (TSUP,VEL,TSUB,Q1F,QQ,QE,Q2PR)
C   +++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C  Arguments
C  ---------
C    Type          Name          ! Mode : Definition
C
       REAL         TSUP          ! Water subcooling
       REAL         VEL          ! Velocity of water
       REAL         TSUB          ! Water subcooling
       REAL         Q1F          ! Heat to water
       REAL         QQ          ! Quench heat
       REAL         QE          ! Heat to evaporation
       REAL         Q2PR          ! heat flux on heater surface
C
C  Include files
C  -------------
C
C  Variables passed to subroutine via include files
C  ------------------------------------------------
C
C    PRESSU   - system pressure
C    RHOSAT   - density at saturation line
C    IPHLIQ   - cuntinuous liquid phase index
C    IPHBUB   - bubble phase index
C    CPSAT    - specific heat at saturation
C    CONSAT   - thermal conductivity at saturation
C    VISLAM   - molecular dynamic viscosity at saturation
C    CONWAL   - wall conductivity
C    RHOWAL   - wall material density
C    CPWALL   - wall heat capacity
C
C  COMMON blocks
C  -------------
C
       COMMON /HEAT1/RHOV,HLV,CONL,ALFA,ACONS,B,DMCON,H1F,QW
       COMMON /HEAT2/A1,A2,DM,DMI,FREQ,ASD,HQ,H2F,A2F,A1F,ERATE
       COMMON /ASDD/ASD1,ASD2,FLAGD
       LOGICAL FLAGD
C
C  Variables passed to subroutine via common blocks
C  ------------------------------------------------
C
```

```
C
C  Locals
C  ------
C     Type           Name           ! Definition
C
C  Set initial values for locals
C  -----------------------------
C
C  Save locals
C  -----------
C SBCMOD-------------------------------------END-OF-HEADING
C
C      INITIALIZE
C
      A1    = 0.E0
      A2    = 0.E0
      DM    = 0.E0
      DMI   = 0.E0
      FREQ  = 0.E0
      ASD   = 0.E0
      HQ    = 0.E0
      H2F   = 0.E0
      A2F   = 0.E0
      A1F   = 0.E0
      ERATE= 0.E0
      TW    = 0.E0
      Q1F   = 0.E0
      QQ    = 0.E0
      QE    = 0.E0
      QW    = Q2PR
C
C ==================================================================
C                              PROPERTIES AT P=45 BAR
      PRES   = 45.e5
      RHOL   = 787.61
      RHOV   = 22.70
      CPL    = 4949.18
      CONL   = 0.6088
      VISK   = 1.305127e-7
      CONW   = 19.0
      RHOW   = 7865.0
      CPW    = 400.0
      SIGMA  = 0.02429
      HFG    = 1675854.0
C                   CALCULATED PROPERTIES
      VISD   =  VISK*RHOL
      PRL    =  VISD*CPL/CONL
      ALFA   =  CONL/RHOL/CPL
C
      HLV    =  HFG + CPL*TSUB
C
C ==================================================================
C
      GAMA   = SQRT(CONW*RHOW*CPW/CONL/RHOL/CPL)
      A1     = SQRT((RHOL-RHOV)*9.81E0/SIGMA)/(HFG*VISD)
      A2     = 0.5E0*SQRT(ALFA/3.14159E0)*0.013E0*PRL**1.7E0*GAMA
C
      BCONS = 1.E0/2.E0/(1.E0-RHOV/RHOL)
      DMCON = 2.42E-5*PRES**0.709E0
C
```

```
C ============================================================
C
C        delta = 0.0001925
C
      REYNO  = VEL*3.2586E-4/VISK
C
      CF   = 0.062E0
      DO I = 1,10
      CF   = 1.E0/( ALOG(REYNO*CF)/0.435E0 + 5.05E0 )
      END DO
C
C      CF = sqrt (actual Cf/2)
C      Ch = s/(1+12.8(Pr^0.68 - 1) sqrt(s))
C
      H1F    = CF*CF/(1.E0- 1.783E0*CF)
      H1F    = H1F*RHOL*CPL*VEL
C
C ================================================================
C                             GUESS TSUP
      TSUP   = QW/H1F-TSUB
C                             IS IT ALREADY IN SINGLE PHASE ?
      FLAGD = .FALSE.
      IF (TSUP.LE.0.05E0) THEN
         DM    = 0.E0
         ASD   = 0.E0
         FREQ = 0.E0
         HQ    = 0.E0
         A1F   = 1.E0
         GO TO 130
      END IF
C
      IF (TSUB.GT.2.E0) THEN
            B = TSUB*BCONS
      ELSE
            TMAX   = QW/H1F-0.001E0
            IF (TMAX.GT.3) TMAX = 3.E0
            IF (TMAX.LT.2) THEN
               FLAGD = .TRUE.
               WRITE (*,*) ' SWITCHED TO TOLUBINSKY DIAM.'
               GO TO 25
            END IF
C
            B2     = 2.E0*BCONS
            B3     = TMAX*BCONS
C
            AA2    = (A1*(QW - H1F*2.E0))**0.33333E0*A2
            AA3    = (A1*(QW - H1F*TMAX))**0.33333E0*A2
            IF (VEL.LT.0.61E0) FI=1.E0
            IF (VEL.GE.0.61E0) FI=(VEL/0.61E0)**0.47E0
            DM2     = DMCON*AA2/SQRT(B2*FI)
            DM3     = DMCON*AA3/SQRT(B3*FI)
            DMI     = DM2 + (DM3-DM2)*(TSUB-2.E0)/(TMAX-2.E0)
C
      END IF
   25 CONTINUE
C --------------------- START ITERATION
C
      ITLIM=100
      RELAX =0.5E0
  110 ITER  =0
  120 ITER  = ITER+1
      IF (ITER.GT.ITLIM) THEN
        ITLIM=ITLIM+50
        RELAX=0.6E0*RELAX
        IF (RELAX.LT.0.005E0) THEN
               WRITE (*,*) 'TSUB = ',TSUB,'   VEL = ',VEL
               WRITE (*,*) 'COULD NOT FIND WALL TEMP.'
               TSUP = -TSUB
               RETURN
        END IF
        GO TO 120
      END IF
C
      CALL SBHEAT(TSUP,TSUB,VEL)
      HH    = A1F*H1F + A2F*HQ
      QTOT  = ERATE*HLV + HH*(TSUB+TSUP)
      IF (HH.EQ.0.E0) THEN
          TSUP1 = -TSUB
      ELSE
          TSUP1 = TSUP + RELAX*(QW-QTOT)/HH
          IF (TSUP1.LE.-TSUB) TSUP1=-TSUB
      END IF
C
      ERR   = ABS((QTOT-QW)/QW)
      IF (TSUP1.LE.0.E0) TSUP1 = 0.8E0*TSUP
      TSUP  = TSUP1
      IF (ERR.GT.1.E-5) GO TO 120
C
  130 CONTINUE
C
        Q1F = A1F*H1F*(TSUP+TSUB)
        QQ  = A2F*HQ*(TSUP+TSUB)
        QE  = ERATE*HLV
C
      RETURN
      END
C      ++++++++++++++++++++++++++++++++++++++
      SUBROUTINE  SBHEAT (TSUP, TSUB, VEL)
C      ++++++++++++++++++++++++++++++++++++++
C
C  Arguments
C  ---------
C     Type          Name          ! Mode : Definition
C
      REAL          TSUP          ! Wall superheat (Twall-Tsat)
      REAL          TSUB          ! Water subcooling (Tsat-Tliq)
      REAL          VEL           ! Water velocity
C
C  Include files
C  -------------
C
C  COMMON blocks
C  -------------
C
      COMMON /HEAT1/RHOV,HLV,CONL,ALFA,ACONS,B,DMCON,H1F,QW
      COMMON /HEAT2/A1,A2,DM,DMI,FREQ,ASD,HQ,H2F,A2F,A1F,ERATE
      COMMON /ASDD/ASD1,ASD2,FLAGD
      LOGICAL FLAGD
C
C  Variables passed to subroutine via common blocks
```

```
C  ------------------------
C
C SBHEAT---------------------------END-OF-HEADING
C
      ASD = 0.0
C
      IF (TSUB.GT.2.E0) THEN
          A      = (A1*(QW - H1F*TSUB))**0.33333E0*A2
          IF (VEL.LT.0.61E0) FI=1.E0
          IF (VEL.GE.0.61E0) FI=(VEL/0.61E0)**0.47E0
          DM     = DMCON*A/SQRT(B*FI)
      ELSE
        IF (FLAGD) THEN
          DM     = 0.0014E0*EXP(-TSUB/45.E0)
        ELSE
          DM     = DMI
        END IF
      END IF
      FREQ  = 3.546E0/SQRT(DM)
      IF (TSUP.LE.0.0001E0) TSUP=0.0001E0
      ASD  = (210.E0*TSUP)**1.805E0
C       write(*,*) 'DM,FREQ,ASD=',DM,FREQ,ASD
C
C
      ATOT  = 1.E0
      A2F   = 3.1415E0*DM*DM*ASD
      A1F   = ATOT - A2F
C
      IF (A1F.LE.1.E-4) THEN
         A2F = ATOT
         A1F = 1.E-4
         ASD = 1.E0/3.1415E0/DM/DM
      END IF
C
      ERATE = 3.14159E0*DM*DM*DM/6.E0*ASD*FREQ*RHOV
      EMAX  = QW/HLV
      IF (ERATE.GT.EMAX) ERATE = EMAX
C
      HQ    = 2.E0*CONL/SQRT(3.14159E0*ALFA/FREQ)
C
      RETURN
      END

```