ENEA Ricerca Sistema Elettrico	Sigla di identificazione	Distrib.	Pag.	di
	NNFISS – LP3 - 016	L	1	139

Titolo

DEVELOPMENT AND VALIDATION OF FEM-LCORE CODE FOR THE THERMAL HYDRAULICS OF OPEN CORES

Descrittori

Tipologia del documento: Rapporto Tecnico

Collocazione contrattuale: Accordo di programma ENEA-MSE: tema di ricerca "Nuovo nucleare da fissione"

Argomenti trattati: Fluidodinamica, termoidraulica del nocciolo, reattori nucleari veloci, generation IV reactors.

Sommario

In this report we discuss the development of the FEM-LCORE code for the study of three-dimensional thermal hydraulics of liquid metal reactors. The code solves the standard Navier-Stokes equations with turbulence models in the reactor upper and lower plena and a special model in the core where all the sub-channel assembly details are summarized in parametric coefficients. Advances in turbulence modeling are reported. In particular, the kω model and a new four-parameter turbulence model are discussed. Moreover, the implementation of parallel computing features in the code with the use of PETSc libraries is illustrated. With this new version of the code we investigate some assembly blockage events that can occur in the core in incidental situations and compare different behaviors for closed and open core models.

Note:

This report has been realized in the frame of ENEA-CIRTEN (UniBo) collaboration.
REPORT LP3-E1b – PAR 2008-2009
Authors:
V.I. Mikhin, M. Polidori, K. Voukelatou – ENEA
G.Bornia, M.Finelli, S. Manservisi – University of Bologna (DIENCA)

REV.	DESCRIZIONE	DATA		REDAZIONE	CONVALIDA	APPROVAZIONE
2			NOME			
			FIRMA			
1			NOME			
			FIRMA			
0	EMISSIONE	01/09/2011	NOME	K. Voukelatou	P. Meloni	P. Meloni
			FIRMA			

DEVELOPMENT AND VALIDATION OF FEM-LCORE CODE FOR THE THERMAL HYDRAULICS OF OPEN CORES

G. Bornia, M. Finelli, S. Manservisi
NUCLEAR ENGINEERING LABORATORY
OF MONTECUCCOLINO
DIENCA - UNIVERSITY OF BOLOGNA
Via dei Colli 16, 40136 Bologna, Italy
sandro.manservisi@unibo.it

V. Mikhin, M. Polidori, K. Voukelatou
ENEA - BOLOGNA
Via Martiri di Montesole 4, 40129 Bologna, Italy
massimiliano.polidori@enea.it

September 1 2011

Abstract. In this report we discuss the development of the FEM-LCORE code for the study of three-dimensional thermal hydraulics of liquid metal reactors. The code solves the standard Navier-Stokes equations with turbulence models in the reactor upper and lower plena and a special model in the core where all the sub-channel assembly details are summarized in parametric coefficients. Advances in turbulence modeling are reported. In particular, the κ-ω model and a new four-parameter turbulence model are discussed. Moreover, the implementation of parallel computing features in the code with the use of PETSc libraries is illustrated. With this new version of the code we investigate some assembly blockage events that can occur in the core in incidental situations and compare different behaviors for closed and open core models.

Contents

Introduction

The FEM-LCORE code is a 3D finite element code for the simulation of the thermo-hydraulic behavior of liquid metal nuclear reactors. This report illustrates the advances introduced in the code in order to improve the modeling of the behavior of liquid metals and to increase the computational performance. The code solves the three-dimensional Navier-Stokes, energy and

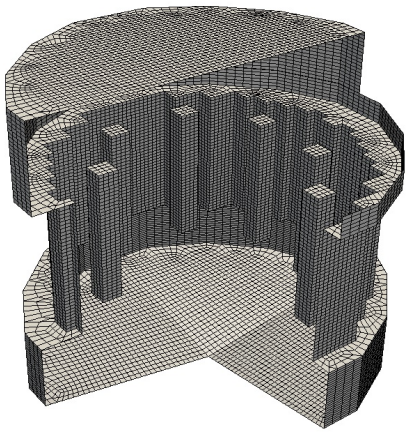

Figure 1: Boundary of the computational three-dimensional reactor model

turbulence equations by means of the finite element method. Some turbulence models are implemented and briefly discussed. A two-level approach is considered for the thermo-hydraulic modeling of the core region, in order to describe the phenomena occurring at fine and coarse grid scales. A porous medium approach is adopted for the description of the assembly geometric details.

In Chapter 1 we illustrate the set of equations adopted for the thermo-hydraulic reactor modeling together with their finite element approximation. Then we overview the structure of the code and show the implementation of the various classes.

Chapter 2 is devoted to the description of different turbulence models that have been implemented or considered for the implementation in the code, together with their comparison with experimental data. In particular, various forms of the κ-ω model are considered and a discussion of the appropriate boundary conditions is also given. A comparison of some results obtained with FEM-LCORE and with other commercial software for the κ-ω model

is discussed. Then, a new turbulence four parametric κ - ϵ-κ_θ - ϵ_θ model is proposed, whose implementation will be described in a future report. The goal of this model is to reproduce in an accurate manner the behavior of low Prandtl number fluids such as liquid metals without the use of wall functions.

In Chapter 3 we describe the advances obtained in the parallelization of the code. The use of PETSc libraries for the implementation of parallel matrices and vectors is illustrated and some tests for the study of the computational performances are examined.

In Chapter 4 we show some examples of simulation of a flow blockage event occurring in the core. A flow blockage can be extremely dangerous as the convection heat transfer strongly diminishes and higher temperatures are expected, closer to the melting point of the core structural and nuclear materials. We consider the cases of both open and closed assemblies and show the results for some different configurations. We compare the results in terms of velocity profiles and peak temperatures.

Chapter 1

Thermo-hydraulic reactor model and FEM-LCORE implementation

In this chapter we recall the basic thermo-hydraulic reactor model and its finite element discretization as proposed in [3]. A discussion of more advanced and new features can be found in Chapters 2 and 3 where turbulence models and parallelization are introduced.

This version of the code presents new features due to parallelization and modularization of the program. The code is split into some independent programs: the mesh data manager `gencase`, the core power data handler `datagen` and the solver program `femlcore`. The `gencase` executable reads a coarse basic mesh, generates the multigrid structure and also divides the domain in a desired number of sub-regions that can be used for parallel computation. With the appropriate mesh, the code can solve not only different reactor models but also assembly and subassembly problems. In order to use different libraries for basic algebraic operations with sparse matrices, an interface defined by object-oriented classes for matrices and vectors is introduced. In the next section we overview the basic structure of the code and its modules.

1.1 Plenum region model

1.1.1 Navier-Stokes and energy equation

The FEM-LCORE is a three-dimensional thermo-hydraulic code that allows to compute the velocity, pressure and temperature distributions in differ-

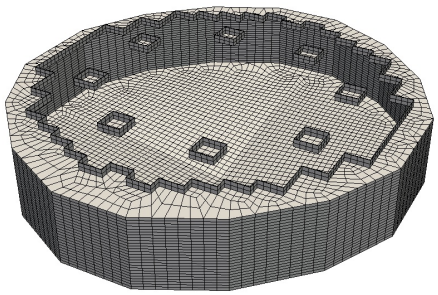

Figure 1.1: Boundary of the computational three-dimensional lower plenum reactor model

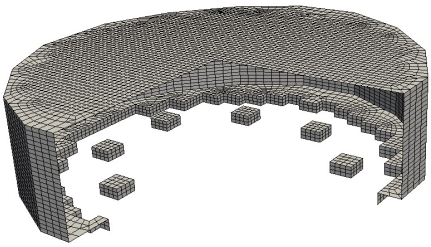

Figure 1.2: Boundary of the computational three-dimensional upper plenum reactor model

ent regions of the reactor. In the regions below and above the core, the coolant flows in an open three-dimensional domain and the coolant state can be determined by using standard three-dimensional CFD models. On the other hand, inside the core the geometry is complex and therefore the three-dimensional flow should be approximated and modeled. In this section, we summarize the equations implemented in the code that are adopted in the modeling of the lower and upper plenum. In these regions the code solves the three-dimensional mass, momentum and energy equations coupled with turbulence models. In particular we may use turbulence models such as κ - ϵ, κ - ω and LES.

Let $(\mathbf{v}, p, T)$ be the state of this system defined by the velocity, pressure and temperature fields and Ω be the domain with boundary Γ. We assume that the evolution of the system is described by the solution of the following equations

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\, \mathbf{v}) &= 0\,, & (1.1)\\ \frac{\partial \rho\, \mathbf{v}}{\partial t} + \nabla \cdot (\rho \overline{\mathbf{v}\mathbf{v}}) &= -\nabla p + \nabla \cdot \bar{\tau} + \rho \mathbf{g}\,, & (1.2)\\ \frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{v} e) &= \Phi + \nabla \cdot \mathbf{q} + \dot{Q}\,. & (1.3) \end{aligned}$$

The fluid can be considered incompressible, while the density is assumed only to be slightly variable as a function of temperature, with a given law $\rho = \rho(T)$. The tensor $\bar{\tau}$ is defined by

$$\bar{\tau} = 2\mu \bar{D}(\mathbf{v})\,, \quad D_{ij}(\mathbf{v}) = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \tag{1.4}$$

with $i, j = x, y, z$. In a similar way the tensor $\overline{\mathbf{v}\mathbf{v}}$ is defined as $\overline{\mathbf{v}\mathbf{v}}_{ij} = v_i\, v_j$. The heat flux, $\mathbf{q}$, is given by Fourier's law

$$\mathbf{q} = -\lambda \nabla T \equiv -C_p \frac{\mu}{Pr} \nabla T\,. \tag{1.5}$$

The Reynolds Re, the laminar Prandtl Pr and the Péclet Pe numbers are defined by

$$Re = \frac{\rho u D}{\mu} \quad Pr = \frac{C_p \mu}{\lambda}, \quad Pe = Re\, Pr\,. \tag{1.6}$$

In order to solve these equations it is also necessary to specify some state law. We assume

$$\rho = a + \gamma T, \quad e \equiv C_v T + \frac{\mathbf{v}^2}{2}\,, \tag{1.7}$$

where a,γ and C_v are constant. The quantity $\mathbf{g}$ denotes the gravity acceleration vector, C_v is the volume specific heat and λ the heat conductivity. The quantity $\dot{Q}$ is the volume heat source and Φ the dissipative heat term. The equations are completed with these data and appropriate boundary conditions.

For high Reynolds numbers the numerical solution of the (1.1-1.3) cannot be computed efficiently and therefore we need an approximate model. Mathematically, one may think of separating the velocity vector into a resolved-scale field and a subgrid-scale field. The resolved part of the field represents the average flow, while the subgrid part of the velocity represents the "small

scales", whose effect on the resolved field is included through the subgrid-scale model. This decomposition results in

$$\mathbf{v} = \bar{\mathbf{v}} + \bar{\mathbf{v}}' \,, \tag{1.8}$$

where $\bar{\mathbf{v}}$ is the resolved-scale field and $\bar{\mathbf{v}}'$ is the subgrid-scale field.

The filtered equations are developed from the incompressible Navier-Stokes equations of motion. By substituting $\mathbf{v} = \bar{\mathbf{v}} + \bar{\mathbf{v}}'$ and $p = \bar{p} + p'$ in the decomposition and then filtering the resulting equation we write the equations of motion for the average fields $\bar{\mathbf{v}}$ and $\bar{p}$ as

$$\frac{\partial \rho \bar{\mathbf{v}}}{\partial t} + \nabla \cdot (\rho \overline{\bar{\mathbf{v}}\bar{\mathbf{v}}}) = -\nabla \bar{p} + \nabla \cdot \bar{\tau} + \rho \bar{\mathbf{g}} \,. \tag{1.9}$$

We have assumed that the filtering operation and the differentiation operation commute, which is not generally the case but it may be thought that the errors associated with this assumption are usually small. The extra term $\partial \tau_{ij}/\partial x_j$ arises from the non-linear advection terms and hence

$$\tau_{ij} = \bar{v}_i \bar{v}_j - \overline{v_i v_j} \,. \tag{1.10}$$

Similar equations can be derived for the subgrid-scale field. Subgrid-scale turbulence models usually employ the Boussinesq hypothesis, and seek to calculate the deviatoric part of stress using

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\mu_t \bar{S}_{ij} \,, \tag{1.11}$$

where $\bar{S}_{ij}$ is the rate-of-strain tensor for the resolved scale and μ_t is the subgrid-scale turbulent viscosity. Substituting into the filtered Navier-Stokes equations, we then have

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \overline{\mathbf{v}\mathbf{v}}) = -\nabla p + \nabla \cdot [(\nu + \nu_t)\nabla \mathbf{v}] + \rho \mathbf{g} \,, \tag{1.12}$$

where we have used the incompressibility constraint to simplify the equation and the pressure is now modified to include the trace term $\tau_{kk}\delta_{ij}/3$. In the rest of this report we drop the average notation $\bar{\mathbf{v}}$ and $\bar{p}$ to use the standard notation $\mathbf{v}$ and p. With this notation these approximation models result in the same equations as (1.1-1.2) for the average fields $(\bar{u}, \bar{p}, \bar{T})$ and a modified viscous stress tensor as

$$\bar{\tau} = 2(\mu + \rho\nu_t)\bar{D}(\bar{u}) \,, \tag{1.13}$$

with a modified heat flux $\mathbf{q}$ as

$$\mathbf{q} = -C_p\left(\frac{\mu}{Pr} + \frac{\rho\nu_t}{Pr_t}\right)\nabla T \,. \tag{1.14}$$

The functions ν_t and Pr_t are called turbulent viscosity and turbulent Prandtl numbers and must be computed by solving other transport equations, referred to as *turbulence models*. We will postpone the discussion of various turbulence models in Chapter 2. For many fluids Pr_t is assumed to be constant.

1.1.2 FEM-LCORE implementation

In the implementation of the code we focus only on liquid lead as coolant. Below we report the state equations that are implemented in the code for liquid lead coolant [7, 10]. These state equations can be modified easily in the configuration files of the code. The lead density is assumed to be a function of temperature as

$$\rho = (11367 - 1.1944 \times T)\,\frac{Kg}{m^3} \tag{1.15}$$

for lead in the range $600K < T < 1700K$. The following correlation has been used for the viscosity μ

$$\mu = 4.55 \times 10^{-04}\, e^{(1069/T)}\, Pa \cdot s\,, \tag{1.16}$$

for lead in the range $600K < T < 1500K$. For the mean coefficient of thermal expansion (AISI 316L) we assume

$$\alpha_v = 14.77 \times 10^{-6} + 12.20^{-9}(T - 273.16) + 12.18^{-12}(T - 273.16)^2\,\frac{m^3}{K}\,. \tag{1.17}$$

The lead thermal conductivity κ is

$$\kappa = 15.8 + 108 \times 10^{-4}\,(T - 600.4)\,\frac{W}{m \cdot K}\,. \tag{1.18}$$

The constant pressure specific heat capacity for lead is assumed not to depend on temperature, with a value of

$$C_p = 147.3\,\frac{J}{Kg \cdot K}\,. \tag{1.19}$$

The equations (1.1-1.3) with the introduction of the turbulent viscosity are implemented in the code. We use the finite element method and therefore the variational form of the Navier-Stokes system. In the rest of the paper we denote the spaces of all possible solutions for pressure, velocity and temperature with $P(\Omega)$, $\mathbf{V}(\Omega)$ and $H(\Omega)$, respectively. The pressure space $P(\Omega)$, the velocity space $\mathbf{V}(\Omega)$ and the energy space $H(\Omega)$ are in general

infinite-dimensional spaces. If the spaces $P(\Omega)$, $\mathbf{V}(\Omega)$ and $H(\Omega)$ are finite-dimensional then the solution $(\mathbf{v}, p, T)$ will be denoted by $(\mathbf{v}_h, p_h, T_h)$ and the corresponding spaces by $P_h(\Omega)$, $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$. In order to solve the pressure, velocity and energy fields we use the finite-dimensional space of piecewise-linear polynomials for $P_h(\Omega)$ and the finite space of piecewise-quadratic polynomials for $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$. In this report the equations are always discretized by Lagrangian finite element families with parameter h.

The finite element Navier-Stokes system becomes

$$\int_\Omega \beta \frac{\partial \rho}{\partial t}\, \psi_h d\mathbf{x} + \int_\Omega \nabla \cdot (\rho\, \mathbf{v}_h)\, \psi_h d\mathbf{x} = 0 \quad \forall \psi_h \in P_h(\Omega)\,, \tag{1.20}$$

$$\begin{aligned} \int_\Omega \frac{\partial\, \rho\, \mathbf{v}_h}{\partial t} \cdot \phi_h \, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h}) \cdot \phi_h \, d\mathbf{x} = \qquad & \forall \phi_h \in \mathbf{V}_h(\Omega) \\ \int_\Omega p_h \nabla \cdot \phi_h \, d\mathbf{x} - \int_\Omega \bar{\tau}_h : \overline{\nabla \phi_h} \, d\mathbf{x} + \int_\Omega \rho \mathbf{g} \cdot \phi_h \, d\mathbf{x} - & \\ \int_\Gamma (p_h \vec{n} - \bar{\tau}_h \cdot \vec{n}) \cdot \phi_{\mathbf{h}} \, d\mathbf{s}\,, & \end{aligned} \tag{1.21}$$

$$\begin{aligned} \int_\Omega \frac{\partial \rho\, C_p\, T_h}{\partial t} \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot (\rho \mathbf{v}_h C_p\, T_h)\, \varphi_h \, d\mathbf{x} = \qquad & \forall \varphi_h \in H_h(\Omega) \\ \int_\Omega \Phi_h\, \varphi_h \, d\mathbf{x} - \int_\Omega \mathbf{q}_h \cdot \nabla \varphi_h \, d\mathbf{x} + \int_\Omega \dot{Q}_h\, \varphi_h \, d\mathbf{x} + \int_\Gamma k\ \nabla T_h \cdot \vec{n}\, \varphi_h \, d\mathbf{x}\,. & \end{aligned} \tag{1.22}$$

The constant β is set to zero for incompressible fluid. Since the solution spaces are finite-dimensional we can consider the basis functions $\{\psi_h(i)\}_i$, $\{\phi_h(i)\}_i$ and $\{\varphi_h(i)\}_i$ for $P_h(\Omega)$, $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$, respectively. Therefore the finite element problem (1.20-1.22) yields a system of equations which has one equation for each FEM basis element. For a Newtonian fluid the viscous stress is given by

$$\tau_{hij} = 2(\mu + \rho\, \nu_t)\, S_{hij}\,, \tag{1.23}$$

where the viscous strain-rate tensor is defined by

$$S_{hij} \equiv \frac{1}{2}\left(\frac{\partial v_{hi}}{\partial x_j} + \frac{\partial v_{hj}}{\partial x_i}\right) - \frac{1}{3}\frac{\partial v_{hk}}{\partial x_k}\delta_{ij}\,. \tag{1.24}$$

The heat flux becomes

$$\mathbf{q}_h = \equiv -C_p(\frac{\mu}{Pr} + \frac{\rho \nu_t}{Pr_t})\nabla T_h\,. \tag{1.25}$$

The Navier-Stokes solver is implemented in the class `MGSolverNS` which consists of the header file `MGSolverNS.h` and class file `MGSolverNS3D.C`. The parameters can be set in the file

`/config/class/MGSNSconf.h .`

The physical properties can be define inside this file. All the class configuration files are under a unique directory `/config/class/`. The energy solver is implemented in the class `MGSolverT` which is defined in the files `MGSolverT3D.C` and `MGSolverT.h`. The parameters can be set in the file

`/config/class/MGSTconf.h .`

In this version of the code each equation class has one one source file where both two-dimensional and three-dimensional equations are written in a dimension independent manner.

1.2 Reactor core region model

1.2.1 Two-level finite element model

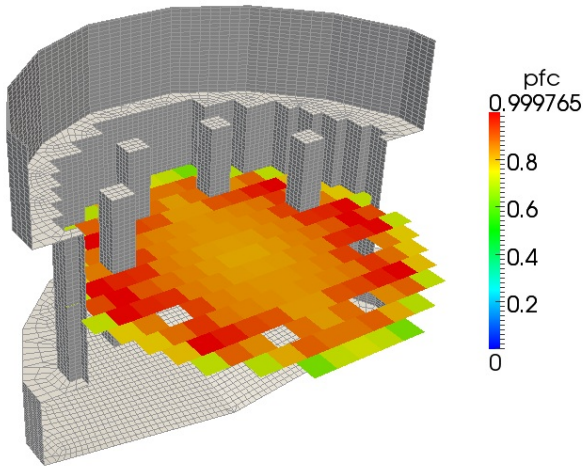

Figure 1.3: Boundary of the computational three-dimensional core reactor model and core power distribution

In the core region the geometry is so complex and detailed that a direct simulation with the purpose of computing the velocity, pressure and temperature distributions is not possible and an approximation is in order. This

approximation is presented in [4] and in this section we briefly recall the equations.

Let us consider a two level solution scheme where all the equations are formulated both at a fine and at a coarse level of space discretization. At the fine level the fluid motion is exactly resolved by the pressure, velocity and temperature solution fields. We denote by $\{\psi_h(i)\}_i$, $\{\phi_h(i)\}_i$ and $\{\varphi_h(i)\}_i$ the basis functions for $P_h(\Omega)$, $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$. Different from the fine level is the coarse level which takes into account only large geometrical structures and solves only for average fields. The equations for these average fields (coarse level) should take into account the fine level by using information from the finer grid through level transfer operators. The definition of these transfer operators is still an open problem. We use the *hat* label for all the quantities at the coarse level. In particular we denote the solution at the coarse level by $(\hat{p}_h, \widehat{\mathbf{v}}_h, \hat{T}_h)$ and the solution spaces by $\widehat{P}_h(\Omega)$, $\widehat{\mathbf{V}}_h(\Omega)$ and $\widehat{H}_h(\Omega)$ respectively.

Let $(p_h, \mathbf{v}_h) \in P_h(\Omega) \times \mathbf{V}_h(\Omega)$ be the solution of the problem at the fine level obtained by solving the Navier-Stokes system written as

$$\int_\Omega \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}})\right) \psi_h(k)\, d\mathbf{x} = 0\,. \tag{1.26}$$

Now let $(\hat{p}_h, \widehat{\mathbf{v}}_h)$ be the solution at the coarse level (fuel assembly level) that satisfies the Navier-Stokes equation with test functions $\widehat{\phi}_h$. These test functions are large enough to describe only the assembly details and satisfy the boundary conditions at the coarse level. Therefore if we substitute the coarse solution $(\hat{p}_h, \widehat{\mathbf{v}}_h)$ the (1.1) becomes [4]

$$\int_\Omega \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}}_h)\right) \widehat{\psi}_h(k)\, d\mathbf{x} = \int_\Omega R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h)\, d\mathbf{x}\,, \tag{1.27}$$

with the total mass fine-coarse transfer operator R^c_{ef} defined by

$$R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \int_\Omega \nabla \cdot \rho(\widehat{\mathbf{v}}_h - \mathbf{v})\, \widehat{\psi}_h(k)\, d\mathbf{x}\,. \tag{1.28}$$

In a similar way let $(p_h, \mathbf{v}_h) \in P_h(\Omega) \times \mathbf{V}_h(\Omega)$ be the solution of the problem at the fine level obtained by solving the momentum equation with the test basis $\{\phi_h(i)\}_i$ in $\mathbf{V}_h(\Omega)$. Now consider the solution $(\hat{p}_h, \widehat{\mathbf{v}}_h)$ at the coarse level (fuel assembly level). It is clear that $(\hat{p}_h, \widehat{\mathbf{v}}_h)$ is different from $(p_h, \mathbf{v}_h)$ and should satisfy the Navier-Stokes equation with test functions $\widehat{\phi}_h$ large enough to describe only the assembly details and satisfy the boundary conditions at the coarse level. Therefore if we substitute the coarse solution

$(\hat{p}_h, \widehat{\mathbf{v}}_h)$ in the momentum equation we have [4]

$$\int_\Omega \frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \int_\Omega \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega \bar{\bar{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)}\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \int_\Omega R^m_{cf}(p_h, \mathbf{v}_h, \widehat{p}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x}\,, \tag{1.29}$$

where the fine-coarse transfer operator $R^m_{cf}(p_h, \mathbf{v}_h, \widehat{p}_h, \widehat{\mathbf{v}}_h)$ is defined by

$$\int_\Omega R^m_{cf}(p_h, \mathbf{v}_h, \widehat{p}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \int_\Omega P^m_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega S^m_{cf}(p_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega K^m_{cf}(\mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x}\,. \tag{1.30}$$

The momentum fine-coarse transfer operator $P_{cf}(p_h - \widehat{p}_h, \mathbf{v}_h - \widehat{\mathbf{v}}_h)$ defines the difference between the rate of virtual work in the fine and in the coarse scale [4] and the turbulent transfer operator $T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h)$ by

$$T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = -\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h} + \nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h} - \nabla \cdot \rho\, \overline{(\widehat{\mathbf{v}}_h - \mathbf{v}_h)(\widehat{\mathbf{v}}_h - \mathbf{v}_h)}\,. \tag{1.31}$$

The turbulent transfer operator $T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h)$ gives the turbulent contribution from the fine to the coarse level. The operator $S^m_{cf}(\widehat{p}_h)$ is defined by

$$\int_\Omega S^m_{cf}(p_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = -\int_\Gamma p_h\, \vec{n} \cdot \widehat{\phi}_h(k)\, d\mathbf{s} \tag{1.32}$$

and the operator $K^m_{cf}(\mathbf{v}_h)$

$$\int_\Omega K^m_{cf}(\mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \int_\Gamma (\bar{\tau}_h \cdot \vec{n}) \cdot \widehat{\phi}_h(k)\, d\mathbf{s}\,. \tag{1.33}$$

The operator $S^m_{cf}(p_h)$ denotes a non symmetric pressure correction from the sub-grid to the pressure distributions of the assembly fuel elements. If the sub-level pressure distribution is symmetric then this term is exactly zero. The operator $K^m_{cf}(v_h)$ determines the friction energy that is dissipated at the fine level inside the assembly. The operator $T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h)$ defines the turbulent energy transfer from the fine to the coarse level. The equation on the coarse level is similar to the equation on the fine level with the exception of the transfer operators.

We can apply the same procedure for the energy equation [4]. Let $(T_h, \mathbf{v}_h) \in H_h(\Omega) \times \mathbf{V}_h(\Omega)$ be the solution of the problem at the fine level and $\widehat{T}_h$ be the coarse level solution. The energy equation becomes [4]

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega \nabla \cdot (\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega \Phi_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega k\, \nabla \widehat{T}_h \cdot \nabla \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega Q_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} = \int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,, \tag{1.34}$$

where the global fine-coarse transfer energy operator R^e_{cf} is defined by

$$\int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x} = \int_\Omega S^e_{cf}(T_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,. \tag{1.35}$$

where

$$P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.36}$$

is the energy fine-coarse transfer operator [4] and

$$T^e_{cf}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla\cdot(\rho\, C_p\, \widehat{\mathbf{v}}_h\, \widehat{T}_h) - \nabla\cdot(\rho\, C_p\, \mathbf{v}_h\, T_h) - \nabla\cdot(\rho\, C_p\, (\widehat{\mathbf{v}}_h - \mathbf{v}_h)\, (\widehat{T}_h - T_h))\,. \tag{1.37}$$

is the energy fine-coarse transfer turbulent operator. The operator $S^e_{cf}(T_h)$ is defined by

$$\int_\Omega S^e_{cf}(T_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} = \int_\Gamma k\, (\nabla T_h \cdot \vec{n})\, \widehat{\varphi}_h(k)\, d\mathbf{x}\,. \tag{1.38}$$

In order to complete the equations (1.27),(1.29) and (1.34) we must compute or model the reactor transfer operators. It is very difficult to define these fully three-dimensional operators in transient situations. Probably these should be computed numerically but in working conditions and near steady state configuration these may be modeled with strong assumptions. The assumptions are as follows.

- P^c_{ef}. In the reactor model we assume incompressibility on both the coarse and the fine level and therefore

$$P^c_{ef} = 0\,. \tag{1.39}$$

The assumption is exact since

$$P^c_{ef}(\widehat{\mathbf{v}}_h - \mathbf{v}_h) = \nabla \cdot \rho(\widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.40}$$

is different from zero only if mass is generated at the fine level. The total mass transfer operator P^c_{ef} may be different from zero if there is a phase change.

- T^m_{cf}. It is usual to compute the term $T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h)$ by using the Reynolds hypothesis, namely
$$T_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = \nabla \cdot \bar{\widehat{\tau}}^{\tau}_h \tag{1.41}$$
where the turbulent tensor $\bar{\widehat{\tau}}^{\tau}_h$ is defined as
$$\bar{\widehat{\tau}}^{\tau}_h = 2\mu_\tau D(\widehat{\mathbf{v}}_h) \tag{1.42}$$
with μ_τ the turbulent viscosity.

- P^m_{cf}. The operator $P_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)$ defines the momentum transfer from fine to coarse level due to the sub-grid fluctuations and boundary conditions. This can be defined by
$$\begin{aligned} &P_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) = \\ &\zeta(\mathbf{x})\Big(\frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k) + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \\ &\qquad \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \bar{\widehat{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)} - \rho \mathbf{g} \cdot \widehat{\phi}_h(k) - \nabla \cdot \bar{\widehat{\tau}}^{eff}\Big) \end{aligned} \tag{1.43}$$
where $\zeta(\mathbf{x})$ is the fraction of fuel and structural material in the total volume. The tensor $\bar{\widehat{\tau}}^{eff}$ is defined as
$$\bar{\widehat{\tau}}^{eff}_h = 2\mu^{eff} D(\widehat{\mathbf{v}}_h)\,. \tag{1.44}$$
The values of μ^{eff} depends on the assembly geometry and can be determined only with direct simulation of the channel or sub-channel configuration or by experiment.

- S^m_{cf}. The operator $S^m_{cf}(p_h)$ indicates a non symmetric pressure correction from the subgrid to the pressure distributions of the assembly fuel elements. If the sub-level pressure distribution is symmetric then this term is exactly zero. Therefore we may assume in working conditions
$$S^m_{cf}(p_h) = 0\,. \tag{1.45}$$

- $K^m_{cf}(v_h)$. The operator $K^m_{cf}(v_h)$ determines the friction energy that is dissipated at the fine level inside the assembly. We assume that the assembly is composed by a certain number of channels and that the pressure drop in this channel can be computed with classical engineering formulas. In working conditions for forced motion in equivalent channels we may set

$$K^m_{cf}(\mathbf{v}_h) = \zeta(\mathbf{x}) \frac{\rho 2 \widehat{\mathbf{v}}_h |\widehat{\mathbf{v}}_h|}{D_{eq}} f_\lambda \,, \tag{1.46}$$

  where D_{eq} is the equivalent diameter of the channel and f_λ the friction coefficient.

- T^e_{cf}. It is usual to compute the term $T^e_{cf}(T_h, \widehat{T}_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h)$, following Reynolds analogy for the turbulent Prandtl number Pr_t, as

$$T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla \cdot (\frac{\mu_t}{Pr_t} \nabla \widehat{T}_h) \,, \tag{1.47}$$

  with μ_t the turbulent viscosity previously defined.

- P^e_{cf}. The operator $P^e_{cf}(\mathbf{v}_h - \widehat{\mathbf{v}}_h)$ defines the energy exchange from fine to coarse level due to the sub-grid fluctuation and boundary conditions. This can be defined as

$$\begin{aligned} &P^e_{cf}(T_h - \widehat{T}_h) = \\ &\zeta(\mathbf{x}) \Big(\frac{\partial \rho C_p \widehat{T}_h}{\partial t} + \nabla \cdot (\rho C_p \widehat{\mathbf{v}}_h \widehat{T}_h) - \Phi_h - Q_h - \nabla \cdot (k^{eff} \nabla \widehat{T}_h) \Big) \end{aligned} \tag{1.48}$$

  where $\zeta(\mathbf{x})$ is the fraction of fuel and structural material in the volume. The values of k^{eff} depend on the assembly geometry and can be determined with direct simulations of the channel or sub-channel configurations or by experiment.

- S^e_{cf}. The operator $S^e_{cf}(p_h)$ is the heat source that is generated through the fuel pin surfaces. For the heat production in the core we may assume

$$S^e_{cf}(p_h) = W(x, y, z) \,, \tag{1.49}$$

  where W_0 is assumed to be a known function of space which is defined by the power distribution factor (see Section 1.3.3).

1.2.2 FEM-LCORE implementation

We can assume the density as a weakly dependent function of temperature and almost independent of pressure. We assume

$$\rho(T,P) = \rho_0(T)\exp(\beta p) \tag{1.50}$$

with $\beta \approx 0$ and define $\rho_{in} = \rho_0(T_{in})$. For the reactor model, with vertical forced motion in working conditions, the state variables $(\widehat{\mathbf{v}}, \widehat{p}, \widehat{T})$ are the solution of the following finite element system

$$\int_\Omega \beta \frac{\partial p_h}{\partial t}\,\widehat{\psi}_h(k)\,d\mathbf{x} + \int_\Omega (\nabla\cdot\rho\,\widehat{\mathbf{v}}_h)\,\widehat{\psi}_h(k)\,d\mathbf{x} = 0\,. \tag{1.51}$$

$$\begin{aligned}\int_\Omega \frac{\partial\,\rho\,\widehat{\mathbf{v}}_h}{\partial t}\cdot\widehat{\phi}_h(k)\,d\mathbf{x} + \int_\Omega (\nabla\cdot\rho\,\overline{\widehat{\mathbf{v}}_h\widehat{\mathbf{v}}_h})\cdot\widehat{\phi}_h(k)\,d\mathbf{x} - \\ \int_\Omega \widehat{p}_h\,\nabla\cdot\widehat{\phi}_h(k)\,d\mathbf{x} + \int_\Omega (\bar{\bar{\tau}}_h + \bar{\bar{\tau}}_h^{\tau} + \bar{\bar{\tau}}_h^{eff}) : \nabla\widehat{\phi}_h(k)\,d\mathbf{x} + \\ \int_\Omega \frac{2\rho\,\widehat{\mathbf{v}}_h|\widehat{\mathbf{v}}_h|}{D_{eq}}\lambda\cdot\widehat{\phi}_h(k)\,d\mathbf{x} - \int_\Omega \rho\mathbf{g}\cdot\widehat{\phi}_h(k)\,d\mathbf{x} = 0\end{aligned} \tag{1.52}$$

$$\begin{aligned}\int_\Omega \frac{\partial\,\rho\,C_p\,\widehat{T}_h}{\partial t}\,\widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\,C_p\widehat{\mathbf{v}}_h\,\widehat{T}_h)\,\widehat{\varphi}_h(k)\,d\mathbf{x} - \\ \int_\Omega \Phi_h\,\widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega (k + k^{eff} + \frac{\mu_t}{Pr_t})\,\nabla\widehat{T}_h\cdot\nabla\widehat{\varphi}_h(k)\,d\mathbf{x} - \\ \int_\Omega Q_h\,\widehat{\varphi}_h(k)\,d\mathbf{x} - \int_\Omega W_0(x,y,z)\,r(\mathbf{x})\,d\mathbf{x} = 0\,.\end{aligned} \tag{1.53}$$

for all $\widehat{\psi}_h(k)$, $\widehat{\phi}_h(k)$ and $\widehat{\varphi}_h(k)$ basis functions. $r(\mathbf{x}) = 1/(1-\zeta(\mathbf{x}))$ is the coolant occupation ratio. The equations (1.51-1.53) are implemented in the code and must be completed with the appropriate boundary conditions.

1.3 The FEM-LCORE code structure

1.3.1 The version 3.0 of the reactor code

The FEM-LCORE code is a CFD code developed at the Laboratory of Montecuccolino at the University of Bologna with the purpose of modelling the thermo-hydraulic behavior of a core of a lead cooled nuclear reactor. Given the complexity of this problem, the main rules in the code development are

the modularity of data structures, the optimization of memory requirements and execution time. To meet these requirements the code is written in *C++* and a number of advanced external tools, such as MPI, PETSc, LibMesh and HDF5 libraries are used.

The `FEM-LCORE` code is nowadays an application of a broader code, called `FEMuS`, developed at the Laboratory of Montecuccolino. The code `FEMuS` was born as a sub-application of the `LibMesh` libraries. Currently `FEMuS` can be installed as a standalone package, in which `FEM-LCORE` is one of its applications. In this version the `femlcore` program of `FEM-LCORE` has no more calls to the LibMesh library and links only with external libraries such as `Laspack` and `PETSc`. Only the `gencase` application for treating mesh input data needs the support of functions from the LibMesh library.

In this User Guide we introduce a general and brief guide to some aspects which are relevant to this application. This application consists of the basic three-dimensional modules for solving the Navier-Stokes, energy and turbulence model equations. A special treatment of the Navier-Stokes equations is considered in the core region where some additional terms are present. The execution of `FEM-LCORE` consists of three steps:

- Pre-processing: mesh and data generation.
- Running: solution of the discretized Finite Element approximation.
- Post-processing: visualization of the results.

This user guide refers to the version 3.0 of the code. The versions 1.0 and 2.0 were discussed in the documents [4] and [3]. The version 1.0 and 2.0 are very different but a mesh reactor input file generated with GAMBIT mesh generator for version 1.0 and 2.0 runs on this version 3.0 and vice versa.

The parallel version of the code must be installed with the LibMesh library since the partitioning of the mesh must be regenerated by the `gencase` program for each parallel configuration (i.e., for each number of processors used). The `gencase` program is used to generate the multigrid mesh starting from the basic mesh generated by a CAD mesh generator. If the LibMesh library is not installed, the multigrid mesh files along with the files for restriction and prolongation operators cannot be generated and they should be provided to the user separately. After every execution of the `gencase` program the mesh files can be found in the `input` directory. Unless one wants to change the geometry or the number of processors, these files do not change. If one wishes to change other quantities such as physical properties, the `gencase` program need not be run a second time.

In order to install the complete version of the `FEM-LCORE` program one must first install the LibMesh and PETSC libraries. The instructions for the PETSc installation can be found at

`http://www.mcs.anl.gov/petsc/petsc-as/`

Concerning the LibMesh installation one can visit the website

`http://libmesh.sourceforge.net`

Once these libraries are installed one can proceed to the installation of the reactor code from the package `RMCFD-3.0.tar.gz` (reactor model CFD version 3.0) as described below. In order to install the package one must choose a directory and run the following commands:

- uncompress `tar xzvf RMCFD-3.0.tar.gz`: the main directory `femus` is generated ;
- enter the directory: `cd femus`;
- edit the configuration script `configure_femus.sh` according to the locations of the external libraries in the current machine;
- run the configuration script: `source configure_femus.sh`;
- go to the main directory: `cd applications/femlcore/`;
- compile the program: `make`.

The configuration script is as follows

```
#!/bin/sh

############## METHOD CONF #############
if test "$METHOD" = ""; then
  export METHOD=opt
  echo "METHOD is set to opt";
fi
############## MACHINE CONF #############
if test "$MYMACHINE" = ""; then
   export MYMACHINE=grid
   echo "MYMACHINE is set to grid"
fi
## MACHINE dependent: LibMesh, PETSC, MPI ##
```

```
############## GRID CONF #############
if test "$MYMACHINE" = "grid"; then
#HDF5
export BASEPATH_TO_HDF5=$HOME/Software
export HDF5_FOLDER=hdf5-1.8.5-patch1-linux-x86_64-shared
export HDF5_INCLUDE=include
export HDF5_LIB=lib
#LIBMESH
export BASEPATH_TO_LM=$HOME/Software
export LM_FOLDER=libmesh-0.7.0.3
#PETSC
export BASEPATH_TO_PETSC=$HOME/Software
export PETSC_FOLDER=petsc-3.1-p7
export PETSC_ARCH=linux-$METHOD
#MPI
export BASEPATH_TO_MPI=/usr/lib64/mpi/gcc
export MPI_FOLDER=openmpi
export    MPI_BIN=bin
export    MPI_LIB=lib64
export    MPI_MAN=share/man
make all
fi
############## END MACHINE DEPENDENT ##########

######### PETSC ##########
export PETSC_DIR=$BASEPATH_TO_PETSC/$PETSC_FOLDER

######### MPI ##########
export MPI_DIR=$BASEPATH_TO_MPI/$MPI_FOLDER
export PATH=$MPI_DIR/$MPI_BIN:$PATH
export LD_LIBRARY_PATH=$MPI_DIR/$MPI_LIB:$LD_LIBRARY_PATH

######## FEMUS #########
export FEMUS_DIR=$PWD
```

This script can be run once the shell variables `MYMACHINE` and `METHOD` are set. The `MYMACHINE` variable designates a particular machine where the code is installed. In every machine the location of the external libraries used by `FEMuS` can be different. Before running the script, the user has to indicate by hand the paths where the external libraries used by `FEMuS` are installed. This can

be done by appropriately filling the variables defined within the `if` statement for each machine. In particular, in the variables `BASEPATH_TO_HDF5`, `BASEPATH_TO_LM`, `BASEPATH_TO_PETSC` and `BASEPATH_TO_MPI` the user must put the absolute paths (i.e. starting from the root `/` directory) where each library package is installed. The names of the respective package folders are set in the variables `HDF5_FOLDER`, `LM_FOLDER`, `PETSC_FOLDER` and `MPI_FOLDER`.

The `METHOD` variable indicates the compiling mode to be used. By setting `METHOD=opt`, the `FEMuS` code is compiled in optimized mode and the optimized version of the external libraries is called by the program. Both the code and the external libraries are used in debugging mode if one sets `METHOD=dbg`. The use of debugging mode is recommended for the development of the code as it helps the developer in finding errors in the implementation of the program. The optimized mode is much faster as many debugging function calls are excluded and it is used for performing the simulations.

In the script the `MYMACHINE` and `METHOD` variables are set by default to `grid` and `opt` respectively. For instance, in the default configuration for the machine denoted by `grid` the absolute paths of the LibMesh and PETSc libraries are in the directories

```
$BASEPATH_TO_LM/$LM_FOLDER=$HOME/Software/libmesh-0.7.0.3
```

and

```
$BASEPATH_TO_PETSC/$PETSC_FOLDER=$HOME/Software/petsc-3.1-p7
```

respectively.

1.3.2 Directory structure

The main directory of the code is divided into different folders. We have eight additional sub-directories:

- `include` that contains the header files;
- `src` that contains the source files;
- `config` that contains the configuration files;
- `contrib` that contains the external libraries;
- `applications` that contains the main programs;
- `fem` that contains the shape function values at the Gaussian points;
- `input` that contains the mesh files;

- `output` that contains the output files.

Inside these directories there are different types of files. Following the `C++` language, one may at first distinguish the source (`.C`) and header files (`.h`). Source files contain implementations of functions and header files contain the prototypes of functions and appropriate macro pre-compilation statements. Often we use the word source to denote the set of all `C++` files, including headers. Inside the code there are other types of files: script (`.sh`) and data (`.in`) files. The files with extension `.in` contain configuration data that are read at run-time. Their modification does not require the recompilation of the code. There are also some configuration scripts in Bash language that allow the setting of the environment variables necessary to set the installation paths of the external libraries. Currently, the `Makefile` and the other configuration scripts are modified manually depending on the characteristics of the machine. In the future we plan to use advanced tools such as `GNU Autotools` that allow the automatic generation of the configuration scripts and the `Makefile`. The `Makefile` defines the rules for compilation and linking and allows to perform various tasks automatically. Let us now turn to the analysis of the sub-directories, giving a brief description of their content.

The `include` and `src` directories

The `include` directory contains the header files. To each class there corresponds a different header file, that contains its declaration and the definition of its prototypes. Each class is conceptually associated to a particular data structure or computing entity. We list here the most important classes:

- `MGMesh`, defined in the file `MGMesh.h`, manages grid and its meshing;
- `MGSolBase`, defined in the file `MGSolverBase.h`, contains all the functions which are in common to a class associated with a differential equation. It is a basic common class that is used in specific equation, through the concept of inheritance of object-oriented programming;
- `MGSolDA`, defined in the file `MGSolverDA.h`, is a specialization of the generic equation MGSolBase for reactive-diffusive-convective equations;
- `MGSolNS`, defined in the file `MGSolverNS.h`, implements the Navier-Stokes equations solved with all the variables coupled (monolithic solver);
- `MGSolKE`, defined in the file `MGSolverKE.h`, implements the κ - ϵ turbulence model;

- `MGSolKW`, defined in the file `MGSolverKW.h`, implements the κ ω turbulence model;
- `MGSolT`, defined in the file `MGSolverT.h`, implements the energy equation.

These files contain the constructor and destructor definitions and all the prototype functions of the corresponding classes.

The `src` directory has the source files that define the functions declared by the headers of the `include` directory. For example, in each file devoted to a single equation, like `MGSolverNS3D.C` for the Navier-Stokes equations, the routines for the assembly of matrices and right-hand sides of that equation are implemented. In each equation file one can also find the implementation of the functions `bc_read` an `ic_read`, that contain the informations of the boundary and initial conditions. In particular, through the `bc` array, each node has a flag associated to it with value 1 to impose a Neumann condition or value 0 to impose a Dirichlet condition.

The `config` directory

The `config` directory contains all the files related to the configuration of the program. The header files with suffix `_conf.h` must be set before compilation since they contain pre-processor directives like `#define` : for example we set the spatial dimension of the problem in `Dim_conf.h`, or we activate the equations to be solved in `Equations_conf.h`. The files `parameters.in`, `param_utils.in` and `param_files.in` contain the parameters that are used at run-time. During its execution, the program reads certain informations from these files, such as the number of time steps or some physical properties. The introduction of the files with suffix `.in` is very useful because one can perform various simulations without having to recompile the source files. In the `class` subfolder there is a specific header file for the configuration of each class.

The `contrib` and `fem` directories

The `contrib` directory contains the contributions taken from external libraries that were possibly modified to meet our purposes. In particular, we find the directories `laspack`, `matrix`, `parallel`, etc.

The `laspack` directory contains a linear algebra package for the solution of sparse linear systems in scalar architectures [8]. The `matrix` and `parallel` directories contain a common interface to various linear algebra libraries such as Laspack or PETSc. In their files, common structures for dealing with

matrices, vectors and solvers are implemented, taking inspiration from the `LibMesh` library. Just by setting one preprocessor directive, the user can switch between either `Laspack` or PETSc libraries.

The previous versions of `femlcore` were dependent on LibMesh functions. Now the code is independent and modular. The linear algebra functions belonging to LibMesh that were called from our code are now replaced with simpler functions independent of LibMesh, contained in the directories `matrix` and `parallel`. A simplification in terms of reduction of function calls and redundant operations has been obtained, which has led to an improvement in performance of the program.

The `fem` directory consists of the files containing the values of the shape functions and their derivatives at the Gaussian points for a given canonical element. The finite elements of type HEX27, HEX8, QUAD9, QUAD4, EDGE3 and EDGE2 are supported in this package. Only HEX27 and HEX8 elements are considered for a three-dimensional reactor core. The names of the files have the form

- `shape(N)D_(GG)(FF).in`

In the name of each file, `N` is the spatial dimension. The first two digits after the underscore "_" sign (denoted as `(GG)`) correspond to the number of Gaussian points, while the second two digits (denoted as `(FF)`) indicate the number of element shape functions. These files are generated by the `Gengauss` program.

The `applications` directory

The classes and functions defined and implemented in the files of the `include/` and `src/` directories can be used for writing different programs. Each program is characterized by its own `main()` function and possibly other specific routines. Inside the directory `applications` various programs can be found: `femlcore`, `gencase`, `gengauss` and `datagen`.

Each program has a source file containing the `main()` function and a Makefile for the compilation and linking.

The `femlcore/` directory contains the reactor program. The class functions written here have the priority over all the other functions. The `main()` function in this directory solves the equation system which consists of momentum. energy and turbulence equations.

The `gencase/` directory contains the `gencase` program that is used for the grid generation in the pre-processing stage. It makes use of the LibMesh functions for mesh generation. Furthermore, it generates the files for the

sparse matrix structure of the global matrix and files for the prolongation and restriction operators, when one adopts a multigrid solver.

The `gengauss/` directory contains the `Gengauss` program that generates files containing the values of shape functions at Gaussian points. These files are read by the `femlcore` program at run-time.

The `datagen/` directory contains the `datagen` program that is used for the generation of the data files concerning the core power factors and core pressure drop values per assembly. These data are required for the core porous medium model. The `datagen` directory contains the files to define the core heat generation and the pressure drop factors. For details see Section 1.3.3.

The `input` and `output` directories

Inside the `input` directory the input data files, needed for the simulation program `femlcore`, are placed. These files are generated by the `gencase` program. The generated files are

- `mesh.h5`, that contains information on the grid in `HDF5` format;
- `Matrix(l).h5`, that contains the sparsity pattern for the system matrix on level `(l)`;
- `Prol(l-1)_(l).h5`, where the prolongation operator from level `(l-1)` to level `(l)` is stored;
- `Rest(l)_(l-1).h5`, that contains the restriction operator from level `(l)` to level `(l-1)`.

In the `output` directory the output files obtained from the code execution can be found. The data are stored in files with binary format `HDF5` (with extension `.h5`). In order to associate data files with the mesh points, it is necessary to create a XDMF file (with extension `.xmf`), based on XML language, which can define this association. The XDMF files can be read by the ParaView software. For a given simulation, the files containing initial and boundary conditions are defined as

`case.(N).(F)`

where (N) corresponds to each time step index of the simulation and (F) is the file extension `.XMF` or `.h5`.

The files related to the solution containing the values of all the fields obtained from the calculation are defined as

`sol.(N).(F)`

where (N) indicates the time step index and (F) is as above. The sequence of time steps starting from the time step index (B) can be viewed by opening the file

`time.(B).xmf`

The file `mesh.xmf` is the XDMF file necessary to read the HDF5 mesh file.

1.3.3 The pre-processing files

The pre-processing stage generates the input files required for the execution of the `femlcore` program. For this purpose we use the `gencase` and `datagen` programs. The files generated by these auxiliary programs are placed in the `input` directory.

Coarse mesh CAD input file

The geometry of the reactor is complex and therefore it can only be generated by a CAD program. With the CAD program we define the coarse mesh that contains all the geometrical details. Over this coarse grid a multigrid system is constructed to define the final fine mesh. The core consists of several assemblies with parallelepiped shapes. In order to impose the heat source correctly, it is required that each assembly is discretized by entire hexahedral cells and the assembly surfaces do not cut any mesh cell internally. This requires some care as the cross sections of the assemblies form a staggered pattern in the reactor design. The code can handle a limited number of mesh formats obtained from CAD programs: the `generic` mesh format obtained from GAMBIT (`.neu` extension) and the `MED` mesh format from SALOME (`.med` extension). The `generic` mesh GAMBIT format must be generated by using only HEX27 finite elements and the SALOME mesh MED format by using HEX20 finite elements. These restrictions are necessary for a successful execution of the `gencase` program.

In case the GAMBIT software is used, the `Solver` menu must be set to `generic`. This is very important since `FLUENT5` options different from `generic` do not produce a readable input file. The mesh must be generated using the HEX27 option on volume elements, the QUAD9 option on surface elements and the EDGE3 option on linear elements. Finally the mesh must be exported with the `Export --> Mesh` command in the `File` menu.

The other mesh generator for the coarse mesh file is *SALOME*. `SALOME` is a free software that provides a generic platform for pre- and post-processing

for numerical simulations. It is based on an open and flexible architecture made of reusable components. It is open source, released under the GNU Lesser General Public License and both its source code and executables may be downloaded from its official website [23]. SALOME has the following main modules: KERNEL (general services), GUI (graphical user interface), GEOM (editing CAD models), MESH (standard meshing), MED (MED data file management) and YACS (coupling codes and calculation). For mesh generation one must use the GEOM module and its mesh capabilities. The MESH module must use the Hexahedron mesh option to have a final mesh with only Hexahedral HEX8 finite elements. Then the elements must be converted to quadratic elements (from HEX8 to HEX20 elements). The HEX8 elements are not allowed as input of the program and therefore the conversion to HEX20 is necessary. After the mesh is generated one can save the mesh in MED format.

The MED data model defines in a logical way the data structures exchanged by codes. The modeled data concern meshes (structured and unstructured) and the resulting fields that can be defined on nodes, elements or Gauss points of meshes. MED supports nine element shapes: point, line, triangle, quadrangle, tetrahedron, pyramid, hexahedron, polygon and polyhedron. Each element may have a different number of nodes, depending on whether linear or quadratic interpolation is used. Since the nodes inside each element could be ordered in multiple ways, MED defines numbering conventions, which are detailed in the MED documentation.

The `gencase` application: mesh and multigrid files

Given the coarse grid the `gencase` program generates the multilevel grid used by `femlcore`. After reading the grid configuration values contained in the configuration file `config/param_utils.in`, `gencase` creates a mesh with the required number of levels. The data of interest in the configuration file are:

- `libmesh_gen`, that indicates whether the grid should be generated by `LibMesh` functions or reading a mesh file obtained from a third party software for generating grids;
- `mesh_refine` that selects whether the grid should be refined;
- `nolevels` that determines the number of levels of the multigrid solver;
- `nintervx` that, in the case of rectangular domain, sets the number of subdivisions of the grid along the direction x;

- `nintervy` that, in the case of rectangular domain, sets the number of subdivisions of the grid along the direction y;
- `nintervz` that, in the case of rectangular domain, sets the number of subdivisions of the grid along the direction z.

Inside the `gencase` program there is also a simple internal coarse mesh generator limited to regular Cartesian geometries. To obtain a parallelepiped reactor of dimensions $[a, b] \times [c, d] \times [e, f]$ the command, which uses the LibMesh library, is written as

```
MeshTools::Generation::build_cube
 (*msh[0],nintervx,nintervy,nintervz,a,b,c,d,e,f,HEX27);
```

The geometries obtained with such calls are very simple. For this reason, the need arises to interact with CAD-based mesh software as described before.

The `gencase` program has been extensively modified after the introduction of parallel computing. In particular, the parallel structure of PETSc matrices requires that the degrees of freedom associated to each process are contiguous to each other. Therefore the vector of global degrees of freedom of the problem will be divided into as many blocks as processes and each block will be assigned exactly to one process. For this reason it is necessary to rerun the `gencase` program every time you want to change the number of processors. In order to facilitate the operations of projection and restriction, *children* elements are assigned to the same process as their *father* elements and in this way a node that is *restricted* on a coarse grid is kept on the same processor. This increases significantly the complexity of the generated mesh files as the number of processors increases.

Once the configuration parameters for the `gencase` program are set, one can compile and run it by reaching the `gencase` directory in a shell terminal and typing on the command line:

```
make
./gencase-opt
```

for serial execution or

```
make
mpiexec -n N gencase-opt
```

for a parallel run, where N is the number of processors.

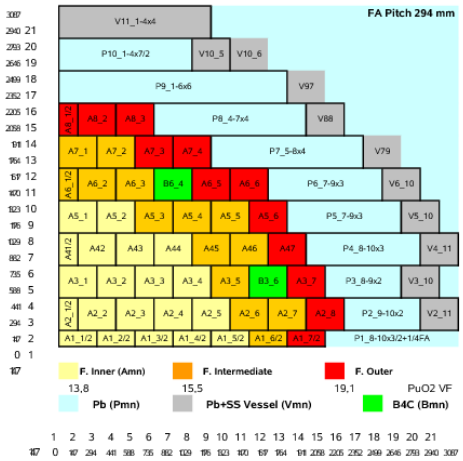

Figure 1.4: Typical core power distribution

The `datagen` application: core power and pressure drop distribution files

The power distribution and the pressure drop distribution are set in the file `input/mesh.data`. A typical power distribution is reported in Figure 1.4. The input file for the power distribution looks like this

```
 Level 0  64
0 0.375 0.375 0.375 1.03 1
1 0.875 0.375 0.375 0.91 1
2 1.375 0.375 0.375 1.02 1
3 1.875 0.375 0.375 1.12 1
4 0.375 0.875 0.375 1.04 1
5 0.875 0.875 0.375 0.78 1
6 1.375 0.875 0.375 1.02 1
7 1.875 0.875 0.375 1.1 1
........
........
```

For each element belonging to each mesh level, the file reports the element number, the x, y and z coordinates of its center point, the value of the power distribution factor and the value of the pressure drop factor, respectively. The file can hardly be edited by hand and it can be generated by the

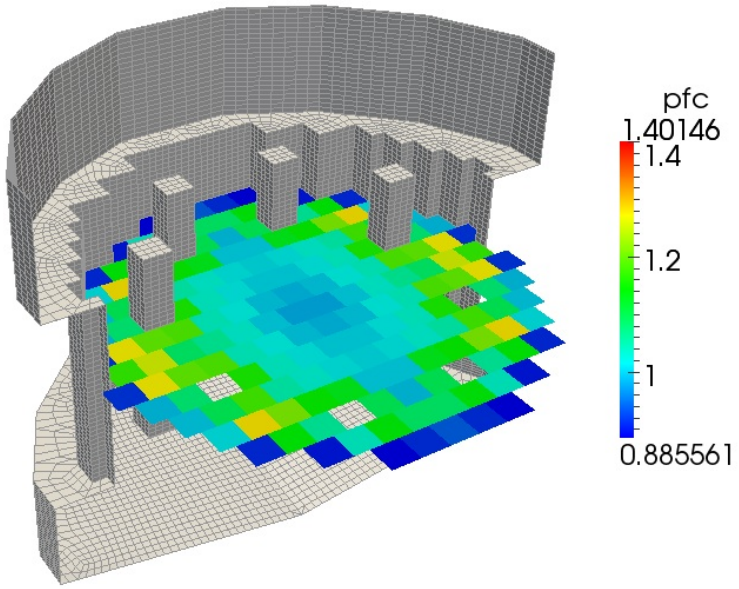

Figure 1.5: Horizontal core power distribution

`datagen` program, located in the directory `applications/datagen/`. In order to load the desired power distribution as shown in Figures 1.5 and 1.6, the file `applications/datagen/datagen.H` has the following parameters: `NLEV` (number of multigrid levels), `NZC` (number of elements in the vertical core section), `HR` (total core height), `HIN` (heat core height), `HOUT` (outlet height). A setting can be as follows

```
#define NLEV 1  // level
#define NZC 9  // number of core elements
// Geometry
#define HR   0.147 // half fuel assembly length
#define HIN  0.95  // heated core start
#define HOUT 1.85  // heated core stops
```

The horizontal power distribution of Figure 1.4 must be reported in the double-indexed array `mat_pf`, in which every assembly in the quarter of reactor is denoted by two indices, both ranging from 0 to 7. The fuel area is divided into three zones: `INNER`, `INTER` and `OUTER`. The zone with no fuel `CTRLR` and the reflector area with `DUMMY` are written in the matrix `mat_zone`.

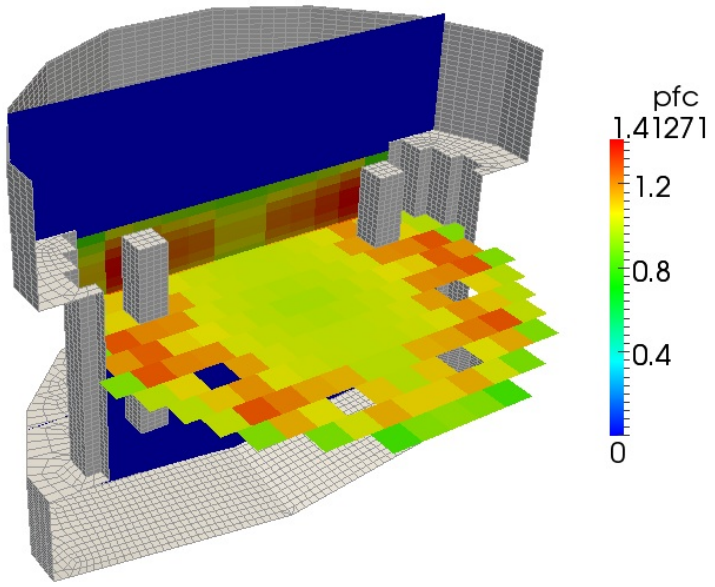

Figure 1.6: Vertical and horizontal core power distributions

```
#define INNER (0)   // zone 0
#define INTER (1)   // zone 1
#define OUTER (2)   // zone 2
#define CTRLR (3)   // zone 3
#define DUMMY (4)   // zone 4

int mat_zone[8][8]={
 {INNER,INNER,INNER,INNER,INNER,INTER,OUTER,DUMMY},//A1_1-A1_7
 {INNER,INNER,INNER,INNER,INTER,INTER,OUTER,OUTER},//A2_1-A2_8
 {INNER,INNER,INNER,INTER,CTRLR,INTER,OUTER,DUMMY},//A3_1-A3_7
 {INNER,INNER,INNER,INNER,INTER,INTER,OUTER,DUMMY},//A4_1-A4_7
 {INNER,INTER,INTER,INTER,OUTER,OUTER,DUMMY,DUMMY},//A5_1-A5_6
 {INNER,INTER,CTRLR,INTER,OUTER,OUTER,DUMMY,DUMMY},//A6_1-A6_6
 {INTER,OUTER,OUTER,OUTER,DUMMY,DUMMY,DUMMY,DUMMY},//A7_1-A7_4
 {OUTER,OUTER,OUTER,DUMMY,DUMMY,DUMMY,DUMMY,DUMMY} //A8_1-A8_3
};
```

The horizontal power factor is reported in the matrix `mat_pf`

```
#define CTL_R 0.
double mat_pf[8][8]={
{0.941,0.962,0.989,1.017,1.045,1.178,1.097,0.   },//A1_1-A1_7
{0.949,0.958,0.978,0.996,1.114,1.123,1.193,0.857},//A2_1-A2_8
{0.963,0.975,0.992,1.087,CTL_R,1.011,0.925,0.   },//A3_1-A3_7
{0.967,0.973,0.989,1.008,1.108,1.028,0.931,0.   },//A4_1-A4_7
{0.961,1.060,1.078,1.137,1.198,0.943,0.,   0.   },//A5_1-A5_6
{0.954,1.029,CTL_R,0.990,1.068,0.850,0.,   0.   },//A6_1-A6_6
{1.019,1.066,0.966,0.842,0.,   0.,   0.,   0.   },//A7_1-A7_4
{0.878,0.853,0.757,0.,   0.,   0.,   0.,   0.   } //A8_1-A8_3
};
```

The vertical power factor can be assumed to have a cosine-like distribution. This profile is reported in `axpf` as

```
 double axpf[10][3]={
{8.60089E-01,8.48697E-01,8.33685E-01},
{9.32998E-01,9.63034E-01,9.52704E-01},
{1.03749E-0,1.06399E-0,1.06801E-0},
{1.10010E-0,1.12959E-0,1.14484E-0},
{1.14410E-0,1.16319E-0,1.17922E-0},
{1.13892E-0,1.15623E-0,1.16983E-0},
{1.09049E-0,1.10313E-0,1.10469E-0},
{1.01844E-0,1.00872E-0,1.00793E-0},
{9.05869E-01,8.55509E-01,8.63532E-01},
{7.71503E-01,7.07905E-01,6.75547E-01}
};
```

In a similar way the pressure drop distribution must be reported in the matrix `axlpf` and in the vector `axllf`.

The procedure to have a new power distribution is as follows:

- delete the file `input/mesh.data`;
- run the code with no `input/mesh.data` file. Then a new `input/mesh.data` is generated and the code stops;
- copy the file `input/mesh.data` in `applications/datagen/data.in`;
- edit `applications/datagen/datagen.H` by inserting the desired power distribution factors;
- run the `datagen` program in the directory `applications/datagen`;

- copy the file `applications/datagen/data.in` in `input/mesh.data`.

The power distribution in Figure 1.4 is enclosed with the provided package. The pressure drop distribution enclosed in the code is constant and equal to 1.

Once the `datagen` program is configured, one can compile and run it by reaching the `datagen` directory in a shell terminal and typing on the command line:

```
make
./datagen-opt
```

for serial execution or

```
make
mpiexec -n N datagen-opt
```

for a parallel run, where `N` is the number of processors.

1.3.4 The post-processing ParaView application

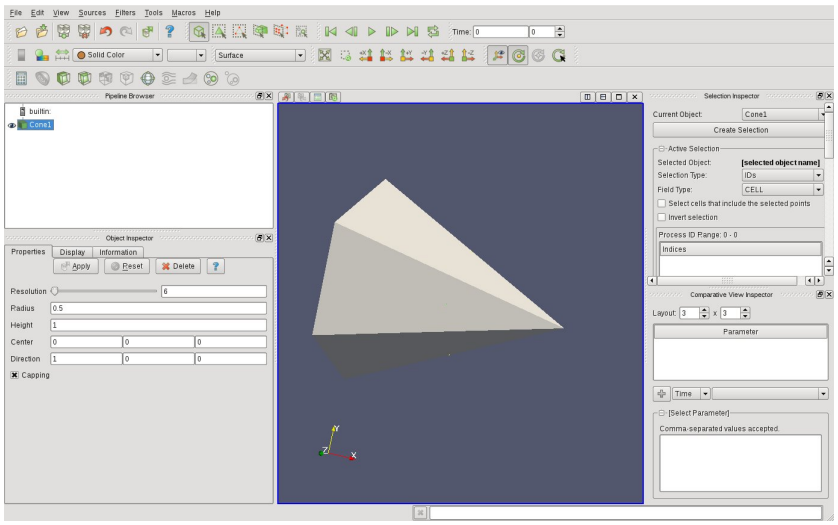

Figure 1.7: Paraview main view

The post-processing stage is the phase dedicated to display the results obtained by the code. All the output files of the program are stored in the `output` directory. The writing data formats are XDMF and HDF5. In the HDF5 files the field values are stored in a hierarchical structure and a

compressed storage format. The XDMF files are required for reading the data contained in the HDF5 files. The XDMF format can be interpreted by some visualization software such as `ParaView`. `ParaView` is open source software for viewing scientific data [20]. Its main view is shown in Figure 1.7. This application is based on the VTK library that provides services for data visualization [28]. `ParaView` is designed for viewing data obtained from parallel architectures and distributed or shared memory computers, but it can also be used for serial platforms.

1.3.5 Configuration and run

In order to run the code the following steps are necessary:
a) generate a coarse mesh file with either GAMBIT or SALOME (name it as `mesh.msh` or `mesh.med` respectively) and copy it in the `input` directory;
b) set the configuration files in the `config` directory and, if necessary, the additional class parameters for the active classes in the `config/class` directory;
c) set the physical and numerical properties in `config/parameters.in` and `config/param_utils.in`;
d) set the boundary and initial conditions in the files of the active classes (for instance, for energy and Navier-Stokes equations the corresponding files are `src/MGSolverT3D.C` and `src/MGSolverNS3D.C`);
e) generate the multigrid mesh with the `gencase` program;
f) generate the core power factor file `mesh.data` with the `datagen` program and copy it in the `input` directory;
g) run the `femlcore` program.

The default reactor configuration is provided with the code and only few changes are necessary. Explanation of steps a), e) and f) concerning the preprocessing files has already been provided in Section 1.3.3. Steps b), c), d) and g) will be described in the following.

Configuration

The code is configured mainly through the various files in the `config` directory.

In the files with suffix `_conf.h` one can find the compile-time options which are implemented through preprocessor directives like `#define`. Some of these options can be seen in Table 1.1. An option is not activated if the corresponding `#define` command is not written or commented (the comment operator is `//`). For example, if we set

```
// #define NS_EQUATIONS 1
```

Directive	Val.	Description
`#define DIMENSION`	2	2D simulation
`#define DIMENSION`	3	3D simulation

`#define HAVE_LASPACKM`
`//#define HAVE_PETSCM`
`//#define HAVE_MPI`
	1	serial solver

`#define HAVE_PETSCM`
`#define HAVE_MPI`
`//#define HAVE_LASPACKM`
	1	parallel solver

`#define NS_EQUATIONS`	1	NS coupled solver
`#define P1_EQUATIONS`	1	NS projection solver
`#define PRINT_INFO`	1	Print info

Table 1.1: Some options in the configuration header files

```
#define T_EQUATIONS 1
```

this means that the solver of the `Navier-Stokes` equations is not active while the energy equation is solved. Thanks to a modular approach we can introduce more options without effort.

In the `config/class` directory the numerical and physical parameters for each particular equation class are defined. All the parameters that are needed in the class model can be changed in these files. We briefly examine the implementation of the dependence of the physical properties on temperature, the turbulence parameters and the numerical solver algorithm.

Dependence of the physical properties on temperature. The code can run with lead properties that can be considered as a function of temperature. If the property law must be modified it is necessary to change the inline functions in the following files:
- `config/class/MGSNSconf.h` for the momentum equations; here the functions $\rho = \rho(T)$ and $\mu = \mu(T)$ are defined;
- `config/class/MGSTconf.h` for the energy equation; here the functions $\kappa = \kappa(T)$ and $C_p = C_p(T)$ are defined;

Furthermore, you have to set

```
// #define CONST 1
```

in the same files. This commented line activates the temperature dependence.

Turbulence parameters. The parameters for the κ-ϵ and κ-ω turbulence models are in the relative class configuration files `config/class/MGSKEconf.h` and `config/class/MGSKWconf.h`. The turbulence parameters of the LES

model are in the `config/class/MGSNSconf.h` file as this model consists in modifying one parameter in the Navier-Stokes configuration.

Numerical solver parameters. For each equation one can choose the solution algorithm. Among the solution methods one can choose one of the following:

```
1.  Jacobi = JacobiIter
2.  SOR forward = SORForwIter
3.  SOR backward = SORBackwIter
4.  SOR symmetric = SSORIter
5.  Chebyshev = ChebyshevIter
6.  CG = CGIter
7.  CGN = CGNIter
x number of elements 8.  GMRES(10) = GMRESIte
9.  BiCG = BiCGIter
10.  QMR = QMRIter
11.  CGS = CGSIter
12.  Bi-CGSTAB = BiCGSTABIter
13.  Test = TestIter .
```

Among the pre-conditioner matrices we can have

```
0.  none = (PrecondProcType)NULL
1.  Jacobi = JacobiPrecond
2.  SSOR = SSORPrecond
3.  ILU/ICH = ILUPrecond.
```

Default configuration uses the GMRES iterative solver with ILU preconditioner.

The files with extension `.in` contain the run-time options. The parameters for the physical values of the problem are contained in the file `parameters.in`. Through the file `param_utils.in` we set numerical and computational parameters. We also have the file `param_files.in`, which allows us to customize the folder names and input/output file names. Table 1.2 shows the main items contained in the files `parameters.in` and `param_utils.in`. We can add new items in these files by simply adding a new couple `variable VALUE` separated by a space.

In some cases the source files outside the `config` directory must be modified before running the code. This happens when one wants to set the boundary and initial conditions.

Boundary conditions. The default boundary conditions for the reactor simulation are set in the code package. In order to change them, it is necessary to edit the appropriate function. For instance, pressure and velocity

param_utils.in		
Parameter	value (es.)	description
dt	0.01	time step
itime	0	initial time
nsteps	100	number of time step
printstep	5	printing interval
restart	0	restart
libmesh_gen	1	*LibMesh* active
mesh_refine	1	mesh refinement
nolevels	4	number of levels
nodigits	4	number of printing digits
pi	3.12159265359	pi
nintervx	20	x number of elements
nintervy	20	y number of elements
nintervz	20	z number of elements

parameters.in		
Parameter	value (es.)	description
Uref	1	reference velocity
Lref	1	reference length
Tref	1	reference temperature
rho0	10562.99	density
mu0	0.0022	viscosity
kappa0	16.58	conductibility
cp0	147.3	heat capacity
komp0	0.	compressibility
qheat	1.14591e+8	heat density power
dirgx	0	x gravity
dirgy	0	y gravity
dirgz	0	z gravity

Table 1.2: Numerical and physical parameters in `param_utils.in` and `parameters.in`

boundary conditions can be edited in the function

```
void MGSolNS::bc_read(
  double xp[],   // xp[]  node coordinates
  int normal[], // normal
  int bc_flag[]  // boundary condition flag
)
```

contained in the file `src/MGSolverNS3D.C`. In this function, a flag is associated to each boundary node indicating if a Dirichlet or Neumann condition is to be enforced on that node. The corresponding boundary values are not assigned here but in the function for the initial conditions. In fact, the initial condition also contains the boundary values because it must fulfill the boundary conditions as well. For the sake of conciseness, we show here the implementation of the boundary conditions for the case of a cubic reactor with unit side. The implementation for the real geometry is similar.

```
void MGSolNS::bc_read(double xp[],
                  double normal [],int bc_flag[]) {

// xp[]=(xp,yp) bc_flag[u,v,p]=0-Dirichlet 1-Neumann
// box boundary conditions
 if (xp[0] < 0.001) { // side 1
   bc_flag[0]=0;bc_flag[1]=0;bc_flag[2]=0;
 }
 if (xp[0] > 1.-0.001) { // side 3
   bc_flag[0]=0;    bc_flag[1]=0;  bc_flag[2]=0;
 }
 if (xp[1] < 0.001)    { // side2
   bc_flag[0]=0;    bc_flag[1]=0;  bc_flag[2]=0;
 }
 if (xp[1] > 1.-0.001) { // side4
   bc_flag[0]=0;    bc_flag[1]=0;  bc_flag[2]=0;
 }
 if (xp[2] < 1.-0.001) { // top
     bc_flag[0]=0;  bc_flag[1]=0;  bc_flag[2]=0;
 }
 if (xp[2] > -0.001)   { // bottom
     bc_flag[0]=0;  bc_flag[1]=0;  bc_flag[2]=0;
 }

  return;
}
```

The coordinates `(xp[0],xp[1],xp[2])` are used to identify the node to which the flags must be associated. The value `bc_flag[0]` is the flag for the boundary condition on the x-component of the velocity field. By setting `bc_flag[0]=0` one imposes a Dirichlet boundary condition. If the flag is not set it defaults to 1 which indicates a Neumann condition. The `bc_flag[1]`

and `bc_flag[2]` flags are for the y and z velocity components respectively. The pressure flag is `bc_flag[3]`.

For the boundary conditions of the energy equation one must edit the function

```
void MGSolT::bc_read(
  double xp[],   // xp[]  node coordinates
  int normal[], // normal
  int bc_flag[]  // boundary condition flag
)
```

in the file `src/MGSolverT3D.C`. Still considering a cubic reactor, the function looks like

```
 void MGSolT::bc_read(double xp[],double normal [],int bc_flag[])
{
 #ifdef DIM2
 #else
 // xp[]=(xp,yp) bc_flag[T]=0-Dirichlet 1-Neumann
 //    boundary conditions box
 if (xp[0] < 0.0001)    bc_flag[0]=0;
 if (xp[0] > 1.-0.0001) bc_flag[0]=0;
 if (xp[1] < 0.0001)    bc_flag[0]=0;
 if (xp[1] > 1.-0.0001) bc_flag[0]=0;
 if (xp[2] < 0.0001)    bc_flag[0]=0;
 if (xp[2] > 1.-0.0001) bc_flag[0]=0;
 #endif
 return;
}
```

The same considerations as before hold for the energy equation. The only difference is that the unknown field is scalar and therefore only one flag per node is to be assigned.

Initial field conditions. If one wants to set the initial solution in pressure and velocity it is necessary to edit the function

```
void MGSolNS::ic_read(
  double xp[],
  double u_value[]
)
```

inside the file `src/MGSolverNS3D.C`. If you open the mentioned file you may find the initial solution $\mathbf{v} = 0$ and p which changes linearly from 1. to 0 along the z-axis. Therefore in the appropriate part of the function you have

```
void MGSolNS::ic_read(double xp[],double u_value[]) {
  // xp[]=(xp,yp,zp) u_value[]=(u,v,w,p)
  u_value[0] = 0.;
  u_value[1] = 0.;
  u_value[2] = 0.;
  u_value[3] = 1.-xp[2];
}
```

If one wants to set the initial temperature one must edit the function

```
void MGSolT::ic_read(
  double xp[],
  double u_value[]
)
```

inside the file `MGSolverT3D.C`. The user can find

```
void MGSolT::ic_read(double xp[],double u_value[]) {
  // xp[]=(xp,yp,zp) u_value[]=(u,v,w,p)
  u_value[0] =400.;
}
```

In this example a uniform temperature of 400^o C is enforced.

Run

In order to run the program one must execute the following commands. It is recommended to run in optimized mode (`METHOD=opt`) for a faster execution. Starting from the `applications/` directory, in the serial mode case one must type the following commands to a shell terminal:

```
cd gencase
make
./gencase-opt
cd ../datagen
make
./datagen-opt
cd ../femlcore
make
./femlcore-opt
```

For a parallel execution with `N` processors one must type instead

```
cd gencase
make
mpiexec -n N gencase-opt
cd ../datagen
make
mpiexec -n N datagen-opt
cd ../femlcore
make
mpiexec -n N femlcore-opt
```

The commands `./exe-opt` or `mpiexec -n 1 exe-opt` are totally equivalent. It is important to remark that the number of processors must be the same for the `gencase`, `datagen` and `femlcore` applications. Different numbers would not allow to perform the desired simulations.

We remark that the `make` commands are necessary after any changes in the header and source files (`.h` and `.C`). The applications do not need to be recompiled after changing the `.in` files and they can be run multiple times.
Restart the code. In order to restart the simulation from the file associated to time step `n` it is simply necessary to set the following parameter in the `param_utils.in` file:

```
restart n
```

and then launch again the `femlcore-opt` executable with the same number of processors as before.

Chapter 2

Advances in implementation of turbulence models

In order to predict velocity and temperature fields correctly the reactor model needs a description of the turbulent exchange of momentum and heat energy. This chapter is devoted to turbulence models and their implementations. Numerous turbulence models exist in literature. In our code we have implemented the LES turbulence model and two-equation turbulence models (in particular κ-ω models). Since heat exchange models based on constant turbulent Prandtl number do not agree with experimental data for liquid metal turbulent flows we then illustrate a four parametric model proposed by ENEA with some applications. In the future we are planning to implement the four parametric model on the code.

2.1 LES

Large eddy simulation (LES) is a popular technique for simulating turbulent flows. An implication of Kolmogorov theory of self-similarity is that the large eddies of the flow are dependent on the geometry but the smaller eddies are independent. This feature allows one to explicitly solve for the large eddies in a calculation and implicitly account for the small eddies by using a subgrid-scale model (SGS model).

The most simple and popular LES model is the Smagorinsky LES model. The Smagorinsky model could be summarized as

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\left(C_s\Delta\right)^2 \left|\bar{S}\right| S_{ij}\,. \tag{2.1}$$

In the Smagorinsky-Lilly model, the eddy viscosity is modeled by

$$\mu_{sgs} = \rho\left(C_s\Delta\right)^2 \left|\bar{S}\right|\,, \tag{2.2}$$

where the filter width is usually taken to be

$$\Delta = (\text{Volume})^{\frac{1}{3}} \tag{2.3}$$

and

$$\bar{S} = \sqrt{2S_{ij}S_{ij}}\,. \tag{2.4}$$

The effective viscosity is calculated from $\mu_{eff} = \mu_{mol} + \mu_{sgs}$. The Smagorinsky constant usually has the value C_s ranging from 0.1 to 0.2.

2.1.1 FEM-LCORE implementation of the LES turbulence model

The LES turbulence model is implemented in the class `MGSolverNS` in the file `MGSolverNS3D.C`. The parameters can be set in the file

`/config/class/MGSNSconf.h`

2.2 κ - ϵ turbulence model

2.2.1 Standard κ - ϵ turbulence model

In the standard κ - ϵ turbulence model the *turbulent viscosity* is modeled as

$$\mu_t = \rho\nu_t\rho C_\mu \frac{\kappa^2}{\epsilon}\,. \tag{2.5}$$

The turbulent kinetic energy κ and the turbulent dissipation energy ϵ satisfy the following equations

$$\frac{\partial\rho\kappa}{\partial t} + \nabla\cdot\rho\mathbf{u}\kappa = \nabla\cdot\left[(\frac{\mu_t}{\sigma_k}+\mu)\nabla\kappa\right] - \rho\beta_k^\epsilon\epsilon + \rho\gamma_k^\epsilon\ S^2 + P_b\,, \tag{2.6}$$

$$\frac{\partial\rho\epsilon}{\partial t} + \nabla\cdot\rho\mathbf{u}\epsilon = \nabla\cdot\left[(\frac{\mu_t}{\sigma_\epsilon}+\mu)\nabla\epsilon\right] - \rho\beta_e\epsilon^2 + \rho\gamma_e\ S^2 + \frac{\epsilon}{\kappa}C_{1\epsilon}C_{3\epsilon}P_b\,, \tag{2.7}$$

where γ_k^ϵ is the production coefficient of k and P_b the buoyancy term. The coefficient β_k^ϵ in the standard model can be assumed unitary. The $\gamma_e = C_\mu C_{1\epsilon}\kappa$ is the coefficient for the turbulent dissipation energy source and $\beta_e = C_{2\epsilon}/\kappa$ the coefficient of the dissipation term for the same equation. We remark that in this formulation γ_e and β_e depend of turbulent kinetic energy κ. The model constants are

$$C_{1\epsilon} = 1.44,\ \ C_{2\epsilon} = 1.92,\ \ C_\mu = 0.09,\ \ \sigma_k = 1.0,\ \ \sigma_\epsilon = 1.3\,. \tag{2.8}$$

The production P_k of k is defined as

$$P_k = -\overline{v_i' v_j'} \frac{\partial v_j}{\partial x_i} = \nu_t S^2 \,, \tag{2.9}$$

where S is the modulus of the mean rate-of-strain tensor, defined as

$$S \equiv \sqrt{2 S_{ij} S_{ij}} = \frac{1}{2} \| \nabla \mathbf{v} + \nabla \mathbf{v}^T \| \,. \tag{2.10}$$

The effect of buoyancy P_b is given by

$$P_b = \alpha_t \frac{\mu_t}{Pr_t} \mathbf{g} \cdot \nabla T \,, \tag{2.11}$$

where Pr_t is the turbulent Prandtl number for energy and g_i is the component of the gravity vector in the i-th direction. For the standard and realizable models, the default value of Pr_t is 0.85. The coefficient α_t is the thermal expansion coefficient.

2.2.2 Boundary conditions for κ - ϵ model

We can subdivide the boundary into inlet, outlet and wall portions, which will be denoted by Γ_i, Γ_o and Γ_w respectively.

For the *inlet boundary conditions* we set the velocity, κ and ϵ fields to their given inlet value. On the boundary Γ_i we set the inlet velocity $\mathbf{u}_{\Gamma_i} = \mathbf{u}_0$ and for κ and ϵ we may enforce

$$\kappa_{\Gamma_i} = 1.5 \bar{u}^2 I^2 \qquad \epsilon_{\Gamma_i} = \frac{C_\mu \, \kappa^{3/2}}{l} \tag{2.12}$$

where $\bar{u}$ is the mean flow velocity, I the turbulence intensity and l the turbulent length scale. For an inlet fully developed flow in a channel we have

$$I = 0.16 \; Re_D^{-\frac{1}{8}} \qquad l = 0.07 \; D \,, \tag{2.13}$$

where Re_D is the Reynolds number based on the hydraulic diameter D. For an inlet laminar flow we may set

$$I = 0.01 \qquad l = 0.07 \; D \,. \tag{2.14}$$

The *outlet boundary conditions* are relatively simple and are taken on the outlet boundary Γ_o as

$$\nabla \mathbf{u} \cdot \mathbf{n}|_{\Gamma_o} = 0 \quad \mathbf{u} \times \mathbf{n}|_{\Gamma_o} = 0 \qquad p|_{\Gamma_o} = 0 \tag{2.15}$$

and

$$\nabla\kappa \cdot \mathbf{n}|_{\Gamma_o} = 0 \qquad \nabla\epsilon \cdot \mathbf{n}|_{\Gamma_o} = 0 \tag{2.16}$$

for the velocity, pressure, turbulent kinetic energy and specific dissipation rate respectively, where $\mathbf{n}$ is the normal unit vector.

At the wall the boundary conditions for the κ-ϵ model can be imposed by using the wall functions or the near-wall boundary conditions. We consider a computational wall to be located inside the channel at a distance δ from the real wall. The mesh size is very important as is the estimation of the non-dimensional distance y^+ from the wall. We define

$$y^+ \equiv \frac{u_* \, \delta}{\nu} \,, \tag{2.17}$$

where $u_* = \sqrt{\frac{\tau_w}{\rho}}$ is the friction velocity at the nearest wall, ρ is the fluid density, δ is the distance to the nearest wall and ν is the local kinematic viscosity of the fluid.

If δ is defined in the *viscous laminar* region then we define the velocity boundary condition as

$$\tau_n|_\delta = \rho\nu\frac{u_t}{\delta} \tag{2.18}$$

where u_t is the tangential velocity and δ is the distance from the wall. The turbulent kinetic energy κ is assumed to be

$$\kappa|_{\delta\approx 0} = 0 \tag{2.19}$$

and the derivative turbulent dissipation energy ϵ vanishing

$$\nabla\epsilon \cdot n|_\delta = 0 \,. \tag{2.20}$$

If δ is not in the viscous layer the wall functions are used. In the *logarithmic layer* ($y^+ > y_c^+$) the velocity can be assessed by the logarithmic law

$$u^+ = \frac{1}{k_v} \, ln(y^+) + B \,. \tag{2.21}$$

The derivative of the turbulent kinetic energy κ is assumed to be zero

$$\nabla\kappa \cdot n|_\delta = 0 \,, \tag{2.22}$$

and the specific dissipation rate ϵ takes the following value

$$\epsilon|_\delta = \frac{C_\mu^{3/4} \, \kappa^{3/2}}{k_v \, \delta} \,. \tag{2.23}$$

2.2.3 FEM-LCORE implementation of the κ-ϵ turbulence model

We consider the turbulent kinetic energy space $K(\Omega)$ and the turbulent dissipation energy space $E(\Omega)$. If the spaces $K(\Omega)$ and $E(\Omega)$ are finite-dimensional then the solution (κ, ϵ) will be denoted by (κ_h, ϵ_h) and the corresponding spaces by $K_h(\Omega)$ and $E_h(\Omega)$. In order to solve the turbulent kinetic and turbulent dissipation energy fields we use the finite-dimensional space of piecewise-quadratic polynomials for $K_h(\Omega)$ and $E_h(\Omega)$. In this report the domain Ω is always discretized by Lagrangian finite element families with parameter h. The equations for the turbulent kinetic energy κ_h and for turbulent dissipation energy ϵ_h, after discretization with FEM, can be written as

$$\int_\Omega \frac{\partial\, \kappa_h}{\partial t}\, \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot (\mathbf{v}_h\, \kappa_h)\, \varphi_h \, d\mathbf{x} + \int_\Omega \left(\nu + \frac{\nu_t}{\sigma_k}\right) \nabla \kappa_h \cdot \nabla \varphi_h \, d\mathbf{x} = \int_\Omega (P_{kh} + P_{bh})\, \varphi_h \, d\mathbf{x} - \int_\Omega \epsilon_h\, \varphi_h \, d\mathbf{x} \qquad \forall \varphi_h \in K_h(\Omega)\,. \tag{2.24}$$

and

$$\int_\Omega \frac{\partial\, \epsilon_h}{\partial t}\, \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot (\mathbf{v}_h\, \epsilon_h)\, \varphi_h \, d\mathbf{x} + \int_\Omega \left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right) \nabla \epsilon_h \cdot \nabla \varphi_h \, d\mathbf{x} = \int_\Omega C_{1\epsilon} \frac{\epsilon}{k} (P_{kh} + C_{3\epsilon} P_{bh})\, \varphi_h \, d\mathbf{x} - \int_\Omega C_{2\epsilon}\, \frac{\epsilon^2}{k}\, \varphi_h \, d\mathbf{x} \qquad \forall \varphi_h \in E_h(\Omega)\,,$$

with all the constants defined as above.

The κ-ϵ turbulence solver is implemented in the class `MGSolverKE` which consists of the declaration file

`include/MGSolverKE.h`

and the implementation file

`src/MGSolverKE3D.C` .

The parameters can be set in the file

`config/MGSKEconf.h` .

2.3 κ-ω turbulence models

2.3.1 Standard κ-ω turbulence model

In this section we consider the κ-ω model and describe briefly the implementation that uses the logarithmic form of the transport equation for the variable ω.

Let κ and ω be the turbulent kinetic energy and the specific dissipation rate. The turbulent viscosity μ_t is defined as $\mu_t = \rho\kappa/\omega$. The standard κ-ω system is defined by [29]

$$\frac{\partial \rho\kappa}{\partial t} + \nabla \cdot \rho\mathbf{u}\kappa = \nabla \cdot \left[(\frac{\mu_t}{\sigma_k} + \mu)\nabla\kappa\right] - \rho\beta_k\kappa\omega + \rho\gamma_k \; S^2 \,, \tag{2.25}$$

$$\frac{\partial \rho\omega}{\partial t} + \nabla \cdot \rho\mathbf{u}\omega = \nabla \cdot \left[(\frac{\mu_t}{\sigma_w} + \mu)\nabla\omega\right] - \rho\beta_w\omega^2 + \rho\gamma_w \; S^2 \,, \tag{2.26}$$

with $\beta_k = \beta^* = 9/100$, $\gamma_k = \mu_t$, $\beta_w = 5/9$, $\gamma_w = \alpha = 3/40$, $\sigma_k = 2$ and $\sigma_w = 2$.

In these equations, the term $\gamma_k \, S^2$ represents the generation of turbulent kinetic energy due to mean velocity gradients. The quantity $\gamma_w \, S^2$ represents the generation of ω. The terms $\beta_k\kappa\omega$ and $\beta_w\omega^2$ represent the dissipation of κ and ω due to turbulence.

The numerical solution of these equations poses some difficulties. For instance, large variations and negative values of ω can occur. In order to avoid such numerical difficulties, the equation (2.26) can be expressed in logarithmic form. The logarithmic specific dissipation rate W is defined by

$$\omega = \exp(W) \tag{2.27}$$

and therefore $W = \ln(\omega)$. With this change of variable we have

$$\frac{\partial \rho\kappa}{\partial t} + \nabla \cdot \rho\mathbf{u}\kappa = \nabla \cdot \left[(\frac{\mu_t}{\sigma_k} + \mu)\nabla\kappa\right] - \rho\beta_k\kappa \exp(W) + \rho\gamma_k \; S^2 \,, \tag{2.28}$$

$$\begin{aligned}\frac{\partial \rho W}{\partial t} + \nabla \cdot \rho\mathbf{u}W = \nabla \cdot \left[(\frac{\mu_t}{\sigma_w} + \mu)\nabla W\right] + \\ (\frac{\mu_t}{\sigma_w} + \mu)(\nabla W)^2 - \rho\beta_w \exp(W) + \rho\gamma_w \; S^2 \exp(-W) \,.\end{aligned} \tag{2.29}$$

In this case the turbulent viscosity is defined as

$$\mu_t = \rho\kappa \exp(-W) \,. \tag{2.30}$$

In (2.30) the turbulent viscosity can be computed as a product of positive factors and divisions by zero are avoided, thus obtaining a further advantage from the numerical point of view. However we note that the exponential function increases the non-linearity in (2.29). Unfortunately the standard κ-ω model described by (2.25-2.26) is inappropriate especially for low Reynolds numbers.

The standard κ-ω model as implemented in many commercial codes is based on the Wilcox κ-ω model given by (2.25-2.26), but it also incorporates modifications for low Reynolds number effects and shear flow spreading. As the κ-ω model has been modified over the years, the production terms in κ and ω equations have been also modified to improve the accuracy of the model for predicting free shear flows. We have implemented this model in our code with no compressibility corrections.

The form of the equations for the turbulent kinetic energy κ and the specific dissipation rate ω is the same as in the standard model (2.25-2.26). Corrections are introduced in the computation of some coefficients. The turbulent viscosity μ_t is computed by combining κ and ω as [30]

$$\mu_t = \alpha^* \frac{\rho k}{\omega} \,. \tag{2.31}$$

The coefficient α^* provides the low Reynolds number correction for the turbulent viscosity, given by

$$\alpha^* = \alpha^*_\infty \left(\frac{\alpha^*_0 + \mathrm{Re}_t/R_k}{1 + \mathrm{Re}_t/R_k} \right) , \tag{2.32}$$

where

$$\mathrm{Re}_t = \frac{\rho k}{\mu \omega} \qquad R_k = 6 \qquad \alpha^*_0 = \frac{0.072}{3} \qquad \alpha^*_\infty = 1 \,.$$

At high Reynolds numbers the κ-ω model yields $\alpha^* = \alpha^*_\infty = 1$.

The quantities $(\frac{\mu_t}{\sigma_k} + \mu)$ and $(\frac{\mu_t}{\sigma_k} + \mu)$ represent the effective diffusivities of κ and ω, respectively. In order to compute them, we use (2.31-2.32) for the turbulent viscosity μ_t and $\sigma_k = 2$ and $\sigma_\omega = 2$ as turbulent Prandtl numbers for κ and ω respectively.

The production term $\rho \gamma_k S^2$ in the κ equation is computed by setting $\gamma_k = \mu_t$ where μ_t is again computed by (2.31-2.32).

For the computation of the dissipation term $\rho \beta_k \kappa \omega$ in the κ equation we use

$$\beta_k = \beta^*_k f_{\beta^*} \,, \tag{2.33}$$

where [6]

$$f_{\beta^*} = \begin{cases} 1 & \chi_k < 0 \\ \frac{1+680\chi_k^2}{1+400\chi_k^2} & \chi_k > 0 \end{cases} \tag{2.34}$$

or [30]

$$f_{\beta^*} = \begin{cases} 1 & \chi_k < 0 \\ \frac{1+680\chi_k^2}{1+80\chi_k^2} & \chi_k > 0 \end{cases} \tag{2.35}$$

where

$$\chi_k \equiv \frac{1}{\omega^3} \nabla k \cdot \nabla \omega \tag{2.36}$$

and

$$\beta_k^* = \beta^* \left(\frac{4/15 + (\mathrm{Re}_t/R_\beta)^4}{1 + (\mathrm{Re}_t/R_\beta)^4} \right) \tag{2.37}$$

with

$$\mathrm{Re}_t = \frac{\rho k}{\mu \omega} \qquad R_\beta = 8 \quad \beta^* = 0.09\,. \tag{2.38}$$

The coefficient γ_w in the production term for ω is given by

$$\gamma_w = 0.52 \left(\frac{1 + \mathrm{Re}_t/R_k}{\alpha_0^* + \mathrm{Re}_t/R_k} \right) \left(\frac{\alpha_0 + \mathrm{Re}_t/R_\omega}{1 + \mathrm{Re}_t/R_\omega} \right), \tag{2.39}$$

where

$$\mathrm{Re}_t = \frac{\rho k}{\mu \omega} \qquad R_k = 6, \qquad R_\omega = 2.95\,. \tag{2.40}$$

The coefficient β_w in the dissipation of ω is computed as

$$\beta_w = \beta_c \, f_\beta\,, \tag{2.41}$$

where

$$\beta_c = 0.072 \qquad f_\beta = \frac{1 + 70\chi_\omega}{1 + 80\chi_\omega} \tag{2.42}$$

and

$$\chi_\omega = \left| \frac{\sum_{ijk} \Omega_{ij} \Omega_{jk} S_{ki}}{(\beta^* \omega)^3} \right| \tag{2.43}$$

$$\Omega_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right) \qquad S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \tag{2.44}$$

2.3.2 Shear-stress transport (SST) κ-ω model

The shear-stress transport (SST) κ-ω model was developed by Menter [11, 12] to effectively blend the robust and accurate formulation of the κ-ω model in the near-wall region with the good behavior of the κ-ϵ model in the turbulent bulk region. To achieve this, the κ-ϵ model is converted into a κ-ω formulation. The standard κ-ω model and the transformed κ-ϵ model are both multiplied by a blending function and added together. The blending function is designed to be equal to 1 in the near-wall region, to reproduce the standard κ-ω model, and equal to 0 away from the surface, to reproduce the κ-ϵ model. We have implemented this model on our FEM in-house code as described in [11]. The SST κ-ω model is similar to the standard κ-ω model in (2.25-2.26) with some correction terms and it can be written as [11, 12]

$$\frac{\partial \rho\kappa}{\partial t} + \nabla \cdot \rho\mathbf{u}\kappa = \nabla \cdot \left[(\frac{\mu_t}{\sigma_k} + \mu)\nabla\kappa\right] - \rho\beta_k^S \kappa\omega + \rho\gamma_k^S \; S^2 \tag{2.45}$$

$$\frac{\partial \rho\omega}{\partial t} + \nabla \cdot \rho\mathbf{u}\omega = \nabla \cdot \left[(\frac{\mu_t}{\sigma_w} + \mu)\nabla\omega\right] - \rho\beta_w^S \omega^2 + \rho\gamma_w^S \; S^2 + 2\,(1 - F_1)\,\rho\sigma_{\omega,2}\nabla k \cdot \nabla\omega\,. \tag{2.46}$$

If the variable W is used then (2.46) becomes

$$\frac{\partial \rho W}{\partial t} + \nabla \cdot \rho\mathbf{u}W = \nabla \cdot \left[(\frac{\mu_t}{\sigma_w} + \mu)\nabla W\right] + (\frac{\mu_t}{\sigma_w} + \mu)(\nabla W)^2 - \rho\beta_w^S \exp(W) + \rho\gamma_w^S \; S^2 \exp(-W) + 2\,(1 - F_1)\,\rho\sigma_{\omega,2} \exp(-W)\nabla k \cdot \nabla W\,. \tag{2.47}$$

The SST κ-ω is based on two blending functions denoted by $F1$ and $F2$ that are employed for the computation of the coefficients. They are defined by

$$F_1 = \tanh\left(\Phi_1^4\right) \qquad F_2 = \tanh\left(\Phi_2^2\right) \tag{2.48}$$

with

$$\Phi_1 = \min\left[\max\left(\frac{\sqrt{k}}{0.09\omega\delta}, \frac{500\mu}{\rho y^2\omega}\right), \frac{4\rho k}{\sigma_{\omega,2} D_\omega^+ \delta^2}\right] \tag{2.49}$$

$$\Phi_2 = \max\left[2\frac{\sqrt{k}}{0.09\omega\delta}, \frac{500\mu}{\rho\delta^2\omega}\right] \qquad D^+ = \max\left[2\rho\frac{\nabla k \cdot \nabla\omega}{\sigma_{\omega,2}\,\omega}, 10^{-10}\right] \tag{2.50}$$

where δ is the distance to the closest surface and D^+ is the positive portion of the cross-diffusion term.

The turbulent viscosity μ_t is computed by combining κ and ω as [11, 12]

$$\mu_t = \frac{\rho k}{\omega} \min\left[\alpha^*, \frac{0.31\,\omega}{\sqrt{2}\,S F_2}\right] \tag{2.51}$$

where S is the strain rate magnitude. The quantity α^* is given by

$$\alpha^* = \left(\frac{0.024 + \mathrm{Re}_t/R_k}{1 + \mathrm{Re}_t/R_k}\right), \tag{2.52}$$

where $\mathrm{Re}_t = \dfrac{\rho k}{\mu\omega}$ and $R_k = 6$. At high Reynolds numbers α^* tends to 1 and at low Reynolds numbers α^* tends to 0.024. The turbulent Prandtl numbers σ_k and σ_ω are

$$\frac{1}{\sigma_k} = \frac{F_1}{\sigma_{k,1}} + \frac{(1-F_1)}{\sigma_{k,2}} \tag{2.53}$$

$$\frac{1}{\sigma_\omega} = \frac{F_1}{\sigma_{\omega,1}} + \frac{(1-F_1)}{\sigma_{\omega,2}}, \tag{2.54}$$

where

$$\sigma_{k,1} = 1.176, \quad \sigma_{\omega,1} = 2.0, \quad \sigma_{k,2} = 1.0, \quad \sigma_{\omega,2} = 1.168\,. \tag{2.55}$$

The coefficient γ_k^S in the production term of turbulent kinetic energy is defined as [11, 12]

$$\gamma_k^S = \min(\mu_t, \frac{10\rho\beta^* k\omega}{2\,S^2})\,, \tag{2.56}$$

where μ_t is computed by using (2.51).

The quantity β_k in the dissipation term of κ is given by [11]

$$\beta_k = \beta_k^*\, f_{\beta^*} \tag{2.57}$$

with $f_{\beta^*} = 1$.

For the coefficient γ_w^S in the production of ω we have [11]

$$\gamma_w^S = \bar{\alpha}\,, \tag{2.58}$$

where

$$\bar{\alpha} = F_1\alpha_1 + (1-F_1)\alpha_2 \tag{2.59}$$

with $\alpha_1 = 5/9$ and $\alpha_2 = 0.44$. This term is implemented in a different way in the *Fluent* code, see [6].

The coefficient β_w^S in the dissipation of ω is given by [11]

$$\beta_w^S = \beta^S\, f_\beta\,, \tag{2.60}$$

where $f_\beta = 1$. The quantity β^S is not constant but

$$\beta^S = F_1\beta_1^S + (1-F_1)\beta_2^S\,, \tag{2.61}$$

where $\beta_1 = 0.075$, $\beta_2 = 0.0828$ and F_1 is obtained from (2.48).

2.3.3 Boundary conditions for κ-ω models

The boundary conditions for the κ-ω model can be imposed by using the wall function or by using the so called *near-wall approach* and no wall functions are taken into account. We can subdivide the boundary into inlet, outlet and wall portions, which will be denoted by Γ_i, Γ_o and Γ_w respectively.

For the *inlet boundary conditions* we set the velocity, κ and ω fields to their given inlet value. On the boundary Γ_i we set the inlet velocity $u_{\Gamma_i} = \bar{u}_0$ and for κ and ω or W we may enforce

$$\kappa_{\Gamma_i} = 1.5\bar{u}^2 I^2 \qquad \omega_{\Gamma_i} = \frac{\sqrt{\kappa}}{l} \qquad W|_{\Gamma_i} = \ln\sqrt{1.5}\bar{u}I - \ln l\,, \tag{2.62}$$

where $\bar{u}$ is the mean flow velocity, I the turbulence intensity and l the turbulent length scale. For an inlet fully developed flow in a channel we have $I = 0.16\ Re_D^{-\frac{1}{8}}$ $\quad l = 0.07\ D$, where Re_D is the Reynolds number based on the hydraulic diameter D. For an inlet laminar flow we may set $I = 0.01$ $\quad l = 0.07\ D$.

The *outlet boundary conditions* are relatively simple and are taken on the outlet boundary Γ_o as

$$\nabla\mathbf{u}\cdot\mathbf{n}|_{\Gamma_o} = 0 \quad \mathbf{u}\times\mathbf{n}|_{\Gamma_o} = 0 \qquad p|_{\Gamma_o} = 0 \tag{2.63}$$

and

$$\nabla\kappa\cdot\mathbf{n}|_{\Gamma_o} = 0 \qquad \nabla\omega\cdot\mathbf{n}|_{\Gamma_o} = 0 \tag{2.64}$$

for the velocity, pressure, turbulent kinetic energy and specific dissipation rate respectively, where $\mathbf{n}$ is the normal unit vector.

For the wall, we consider a *computational wall* to be located inside the channel at a distance δ from the real wall. We define the boundary conditions by defining the wall shear stress $\tau_w = \mu\partial u/\partial\delta|_\delta$ at the boundary δ with μ is the dynamic viscosity and u is the flow velocity parallel to the wall.

If δ is defined in the *viscous laminar* region then we define the velocity boundary condition as

$$\tau_n|_\delta = \rho\nu\frac{u_t}{\delta} \tag{2.65}$$

where u_t is the tangential velocity and δ is the distance from the wall. The turbulent kinetic energy κ is assumed to be

$$\kappa|_{\delta\approx 0} = 0 \tag{2.66}$$

and the specific dissipation rate ω takes the following value [11, 12]

$$\omega|_\delta = \frac{\epsilon}{\beta^*\kappa} \approx 6\frac{\nu}{\beta^*\delta^2}\,. \tag{2.67}$$

For the logarithmic specific dissipation rate W we have

$$W|_\delta = \ln 6\nu - \ln \beta^* \delta^2 \,. \tag{2.68}$$

For δ tending to zero we have that κ tends to zero and both ω and W go to infinity.

If δ is not in the viscous layer the wall functions are used. In the *logarithmic layer* ($y^+ > y_c^+$) the velocity can be assessed by the logarithmic law

$$u^+ = \frac{1}{k_v}\, ln(y^+) + B \,. \tag{2.69}$$

The derivative turbulent kinetic energy κ is assumed to be zero

$$\nabla\kappa \cdot n|_\delta = 0 \,, \tag{2.70}$$

and the specific dissipation rate ω takes the following value [11, 12]

$$\omega|_\delta = \frac{\sqrt{\kappa}}{C_\mu\, \delta} \,. \tag{2.71}$$

For the logarithmic specific dissipation rate W we have

$$W|_\delta = 0.5 \ln \kappa - \ln C_\mu \delta \tag{2.72}$$

2.3.4 FEM-LCORE implementation of the κ-ω turbulence models

We consider the turbulent kinetic energy space in $K(\Omega)$ and turbulent specific dissipation rate space $W(\Omega)$. If the spaces $K(\Omega)$ and $W(\Omega)$ are finite-dimensional then the solution (κ, ω) will be denoted by (κ_h, ω_h) and the corresponding spaces by $K_h(\Omega)$ and $W_h(\Omega)$. In order to solve the turbulent kinetic energy and specific dissipation rate fields we use the finite-dimensional space of piecewise-quadratic polynomials for $K_h(\Omega)$ and $W_h(\Omega)$. In this report the domain Ω is always discretized by Lagrangian finite element families with parameter h. The equation for the turbulent kinetic energy equation κ_h and for the turbulent specific dissipation rate ω_h are implemented as

$$\int_\Omega \frac{\partial \rho\kappa}{\partial t}\, \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot \rho\mathbf{u}\kappa\, \varphi_h \, d\mathbf{x} + \int_\Omega (\frac{\mu_t}{\sigma_k} + \mu)\nabla\kappa \cdot \nabla\varphi_h \, d\mathbf{x} = - \tag{2.73}$$
$$\int_\Omega \rho\beta_k\kappa\omega\, \varphi_h \, d\mathbf{x} + \int_\Omega \rho\gamma_k\, S^2\, \varphi_h \, d\mathbf{x} \qquad \forall\varphi_h \in K_h(\Omega) \,.$$

$$\int_\Omega \frac{\partial \rho\omega}{\partial t}\,\varphi_h\,d\mathbf{x} + \int_\Omega \nabla\cdot\rho\mathbf{u}\omega\,\varphi_h\,d\mathbf{x} + \int_\Omega \Big[(\frac{\mu_t}{\sigma_w}+\mu)\nabla\omega\cdot\nabla\varphi_h\,d\mathbf{x}$$
$$= -\int_\Omega \rho\beta_w\omega^2\,\varphi_h\,d\mathbf{x} + \int_\Omega \rho\gamma_w\ S^2\,\varphi_h\,d\mathbf{x} + \tag{2.74}$$
$$\int_\Omega 2\,\varphi_h\,(1-F_1)\,\rho\sigma_{\omega,2}\nabla k\cdot\nabla\omega\,d\mathbf{x} \qquad \forall\varphi_h\in W_h(\Omega)\,.$$

If the variable W is used then (2.74) becomes

$$\int_\Omega \frac{\partial \rho W}{\partial t}\varphi_h\,d\mathbf{x} + \int_\Omega \nabla\cdot\rho\mathbf{u}W\varphi_h\,d\mathbf{x} + \int_\Omega (\frac{\mu_t}{\sigma_w}+\mu)\nabla W\cdot\nabla\varphi_h\,d\mathbf{x} =$$
$$\int_\Omega (\frac{\mu_t}{\sigma_w}+\mu)(\nabla W)^2\varphi_h\,d\mathbf{x} - \int_\Omega \rho\beta_w\exp(W)\varphi_h\,d\mathbf{x} +$$
$$\int_\Omega \rho\gamma_w\ S^2\exp(-W)\varphi_h\,d\mathbf{x} + \qquad \forall\varphi_h\in W_h(\Omega)$$
$$\int_\Omega 2\varphi_h\ (1-F_1)\,\rho\sigma_{\omega,2}\exp(-W)\nabla k\cdot\nabla W\,d\mathbf{x}\,. \tag{2.75}$$

with all the constants defined as above.

The $\kappa-\omega$ turbulence model solver is implemented in the class `MGSolverKW` which consists of the declaration file

`include/MGSolverKW.h`

with the implementation file

`src/MGSolverKW3D.C`

and the parameters can be set in the file

`config/class/MGSKWconf.h.`

2.3.5 A test for SST κ-ω model in single rod geometry

This test has been introduced to investigate the behaviour of the SST κ-ω model implemented on the code. The SST κ-ω model was developed to effectively blend the robust and accurate formulation of the κ-ω in the near-wall region with the good behavior of the κ-ϵ model in the turbulent bulk region. The blending functions are designed to be equal to 1 in the near-wall region, to reproduce the standard κ-ω model, and equal to 0 away from the surface, to reproduce the κ-ϵ model. For this reason we plan to use this model as the main model in the rest of this work.

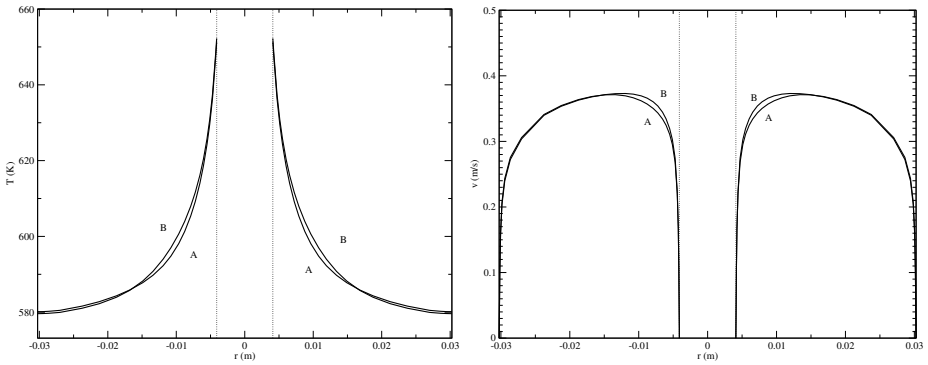

Figure 2.1: SST κ - ω test. The temperature (left) and velocity (right) profiles across the annular region at $z = 2.2\,m$ for the standard κ - ω (A) and SST κ - ω model (B).

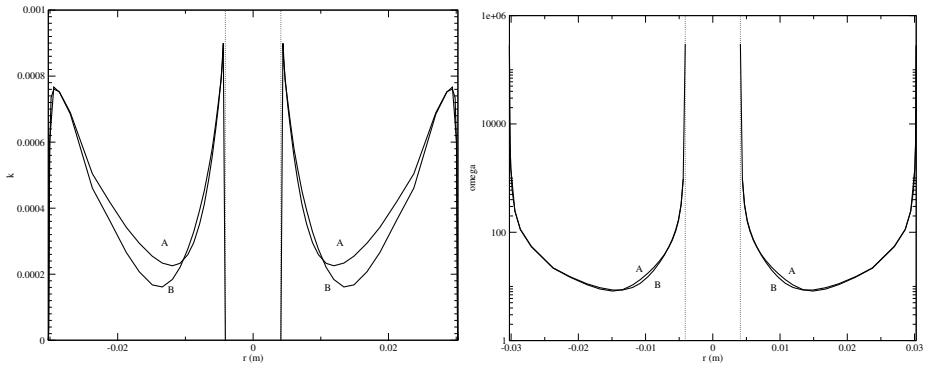

Figure 2.2: SST κ - ω test. The turbulent kinetic energy κ (left), the specific dissipation rate ω (right) profiles across the annular region at $z = 2.2\,m$ for the standard κ - ω (A) and SST κ - ω model (B).

The benchmark geometry consists of an annular geometry with total length $L = 2.25\,m$ and inner and outer diameters $D = 3.025 \times 10^{-2}\,m$ and $d = 4.1 \times 10^{-3}\,m$ respectively. The LBE liquid metal flows from the bottom to the top of the channel in the annular region. The properties of lead-bismuth eutectic alloy are taken at reference temperature of $573.15K$. A constant temperature $T_i = 573.15K$ and a flat velocity profile of $v_0 = 0.667\,\mathrm{m/s}$ are assumed on the inlet section. With these data the corresponding Prandtl number is 0.023 and the Reynolds number is approximately $1 \cdot 10^5$. On the inner cylinder we assume no slip boundary conditions and a constant surface heat flux $q = 3.1 \cdot 10^5\,W/m^2$. On the outer cylinder we set adiabatic boundary conditions. On the outlet boundary we use the standard outflow conditions

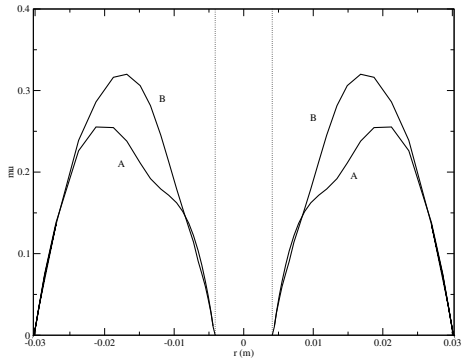

Figure 2.3: SST κ-ω test. Turbulent viscosity profiles across the annular region at $z = 2.2\,m$ for the standard κ-ω (A) and SST κ-ω model (B).

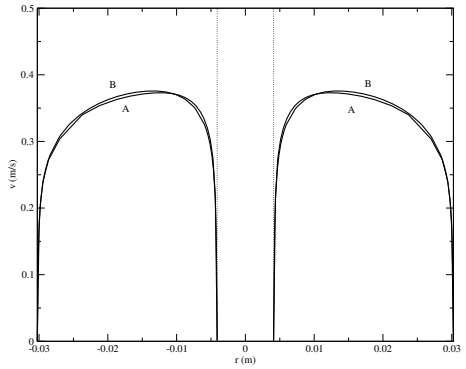

Figure 2.4: SST κ-ω test. The velocity profiles across the annular region at $z = 2.2\,m$ for the code (A) and *Fluent* (B).

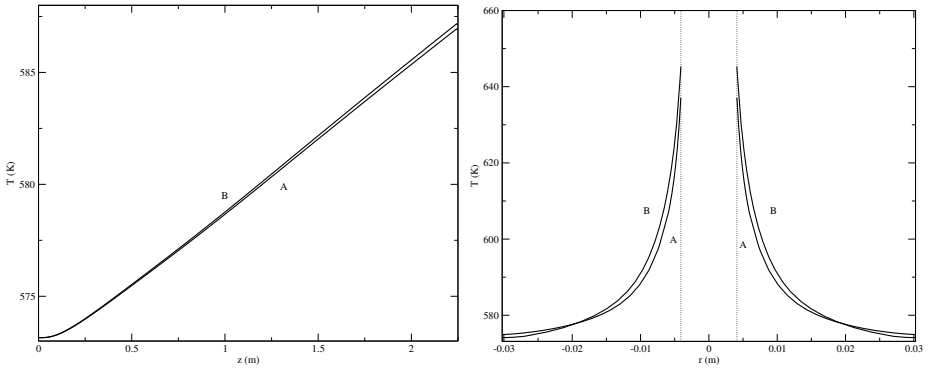

Figure 2.5: SST κ-ω test. The temperature profiles along the line $r = 0.25\,(D + d)$ (left) and across the annular region at $z = 2.2\,m$ (right) for the code (A) and *Fluent* (B).

described in (2.15). In agreement with Section 2.3.3, the boundary conditions imposed on the velocity field are imposed on a cylindrical surface at a distance δ from the real wall. We refine the mesh until the surface defined by δ lies in the viscous laminar region. When the distance from the wall δ is small enough then we set the near-wall boundary conditions. We set the boundary stress as $\tau_n|_\delta = \rho\nu\frac{u_t|_\delta}{\delta}$ and the normal velocity $u_n|_\delta = 0$. If δ tends to zero then u_t tends to zero, giving full no-slip boundary conditions. The boundary conditions for κ and ω are taken in agreement with the previous Section. For the turbulent kinetic energy and specific dissipation rate we fix the values on the inlet as $\kappa = 0.00105\, m^2/s^2$ and $\omega = 16.3\, s^{-1}$ respectively. On the outlet we use the standard outflow conditions in (2.16) which set to zero the normal derivatives of κ and ω. On the wall we set the near-wall boundary conditions $\kappa|_\delta \approx= 0$ and $\omega|_\delta \approx 6\frac{\nu}{\beta^*\delta^2}$ as described in (2.66-2.67).

In Figure 2.1 the temperature (left) and velocity (right) profiles across the annular region for the standard κ-ω (A) and SST κ-ω models (B) are shown. In a similar way the turbulent kinetic energy κ (left) and the specific dissipation rate ω (right) profiles across the annular region for the two models can be seen in Figure 2.2. The κ and ω profiles for the two models are different and the resulting turbulent viscosity can be seen in Figure 2.3.

A comparison between the finite element code and *Fluent* can be found in Figures 2.4-2.5. In Figure 2.4 the velocity profiles across the annular region at $z = 2.2\, m$ obtained from the SST κ-ω solutions of the code (A) and *Fluent* (B) are compared. In Figure 2.5 the temperature profiles along the line $r = 0.25\,(D + d)$ (left) and across the annular region at $z = 2.2\, m$ (right) for the code (A) and *Fluent* (B) are shown.

2.4 Four parameter κ-ϵ-κ_θ-ϵ_θ turbulence model

2.4.1 Introduction

Practically all the effective turbulence models like κ-ϵ dealing with the averaged fields, say velocity, pressure and temperature, contain model functions. Usually they are the functions of a distance from the wall, so called wall-functions. Applying these functions has the aim to take into account the turbulence specific near the wall (the effect of the wall on the spatial turbulence scale) and achieve the agreement between computed and experimental data. But entering in the differential equations of these empirical functions shows on incompleteness of models arising from a lack of necessary physical ideas. As an example, consider a κ-ϵ model. Applying the algebraic transformations and Reynolds averaging procedure to the fundamental equations

of fluid dynamics one can obtain the exact equations for the turbulent energy and its dissipation rate. (The last appears naturally in the equation for the turbulent energy.) Both equations contain the correlation functions which one needs to express via the averaged fields of the velocity, pressure, turbulent energy and dissipation rate for closing the equation system. The exact equation for the turbulent energy contains a small number of the correlation functions. The very simple physical ideas are used for their approximation. One of them is the Boussinesq hypothesis expressing the Reynolds stress tensor via the strain velocity tensor and eddy viscosity. With high probability the resulting model equation for the turbulent energy may be considered as close to the exact one. But the correlation functions of the exact equation for the dissipation rate are much more complex and manifold. For this reason, very often the model equation for the dissipation rate is constructed similarly to the equation for the turbulent energy and has the same terms: the generation, diffusion and dissipation one. The analysis of the different correlation functions is practically ignored. This leads to the necessity of introducing the model functions into the equation system. It is considered that the model functions have effect near the walls. Really, in some cases their effect can extend far from the walls. The utilization of the model functions becomes problematic in the case of complex channels. Another problem arises with the numerical simulation of non-isothermal flows by means of the effective turbulence models operating with the averaged fields. The similarity hypothesis relating the turbulent heat conductivity with the eddy momentum diffusion by means of the turbulent Prandtl number is usually used in this case. But different experiments show that the turbulent Prandtl number is not a generalized parameter and depends on the spatial coordinates and heat exchange conditions. To solve this problem a four-parametric κ - ϵ- κ_θ - ϵ_θ turbulence model for simulating heat exchange in fluids with different Prandtl numbers was proposed [24]. In this model, the eddy heat conductivity was determined via the four parameters: κ - ϵ-κ_θ - ϵ_θ, where κ_θ and ϵ_θ are the temperature fluctuations squared and their dissipation rate respectively. The increase of the parameters number made the model functions much more complex and increased their number. The fundamental equations governing a fluid motion include the all fluid properties and the resulting model equations should themselves take into account the turbulence specific near the walls. It means that a complete turbulence model should not contain any model functions. In the previous works [13, 14, 15, 16] the two-parametric κ - ϵ turbulence model for simulating isothermal flows of incompressible fluids was proposed. The model did not use any model functions and had shown satisfactory agreement between calculated and experimental results for flows in different channels. It was obtained by expanding the correlation func-

tions of the exact equations for the turbulent energy and its dissipation rate into the sum of ensemble-averaged velocity spatial derivatives of the first and second order. Representing the Reynolds stress tensor via the sum of the first and second order velocity spatial derivatives gives the new terms in the model equations for the momentum and turbulent energy. These terms can be called as the exchange forces. They cause the kinetic energy transfer from the main flow to the secondary flows in straight channels [13]-[16]. Neglecting the second-order velocity spatial derivatives reduces the relationship for the Reynolds stress tensor to the well-known Boussinesq formula. The combination of very productive idea of the work [24], allowing avoid the use of the turbulent Prandtl number concept, and ideas of the works [13]-[16] was used for constructing the four-parametric turbulence model for non-isothermal flows of compressible fluids [17]. In the present work, the edition of the model [17] for incompressible but non-isothermal fluid flows is presented as well as the simulation results for a turbulent natural convection along a vertical flat plate and their comparison with the experimental data.

2.4.2 Transport equations as a result of applying the Reynolds averaging procedure to the Navier-Stokes and energy equations

Consider that the flow velocity is small in comparison with the speed of sound. In this case the density is independent from the pressure and may depend from the temperature. We can put

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u_k}{\partial x_k} = 0 \, . \tag{2.76}$$

Everywhere the twice repeated index means the summation on the index. The common form of the continuity equation

$$\frac{\partial \rho}{\partial t} + \boldsymbol{u} \cdot \nabla \rho = 0 \tag{2.77}$$

is reduced to the *substantial derivative of the density*:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{u} = 0 \tag{2.78}$$

After applying the algebraic transformations and Reynolds averaging procedure to the fundamental dynamic equations of incompressible fluids we obtain the following transport equations expressed via the averaged fields.

The momentum equation (Reynolds equation)

$$\rho\frac{\partial u_i}{\partial t}+\rho u_k\frac{\partial u_i}{\partial x_k}=-\frac{\partial\rho\overline{u_k'u_i'}}{\partial x_k}+\frac{\partial\sigma_{ik}}{\partial x_k}+\rho g_i \tag{2.79}$$

The heat conductivity equation

$$\rho C_p\left(\frac{\partial T}{\partial t}+u_k\frac{\partial T}{\partial x_k}\right)=\beta T\frac{Dp}{Dt}-\frac{\partial\rho C_p\overline{u_k'T'}}{\partial x_k}+\frac{1}{2}\mu S_{ij}^2+\rho\epsilon+\frac{\partial}{\partial x_k}\left(\lambda\frac{\partial T}{\partial x_k}\right) \tag{2.80}$$

The first term in the right-hand side of the equation (2.80) contains the substantial derivative of the pressure, Dp/Dt, which appears when the common form of the energy equation with a variable density is formulated as a balance of the temperature field [25].

The turbulent energy equation

$$\frac{\partial\kappa}{\partial t}+u_i\frac{\partial\kappa}{\partial x_i}=-\overline{u_i'u_j'}\frac{\partial u_i}{\partial x_j}+\frac{\hat{u}_i}{\rho}\frac{\partial\sigma_{ik}}{\partial x_k}+\frac{\partial}{\partial x_j}\left(\frac{\nu_t}{\sigma_\kappa}+\nu\right)\frac{\partial\kappa}{\partial x_j}-\epsilon \tag{2.81}$$

The turbulent energy dissipation rate equation

$$\begin{aligned}\frac{\partial\epsilon}{\partial t}+u_i\frac{\partial\epsilon}{\partial x_i}=&\\ &-2\nu m_{ij}\frac{\partial u_i}{\partial x_j}-\nu r_{ijk}R_{ijk}-\frac{2}{3}\nu\frac{\partial\overline{u_i'u_j'}}{\partial x_j}\frac{\partial^2u_i}{\partial x_l^2}\\ &-2\nu\overline{\frac{\partial u_i'}{\partial x_j}\frac{\partial u_i'}{\partial x_k}\frac{\partial u_k'}{\partial x_j}}+\frac{\partial}{\partial x_j}\left(\frac{\nu_t}{\sigma_\epsilon}+\nu\right)\frac{\partial\epsilon}{\partial x_j}-C_{\epsilon2}\frac{\epsilon^2}{\kappa}\end{aligned} \tag{2.82}$$

The equation for the averaged temperature fluctuations squared

$$\frac{\partial\kappa_\theta}{\partial t}+u_i\frac{\partial\kappa_\theta}{\partial x_i}=-\overline{u_i'T'}\frac{\partial T}{\partial x_i}+\frac{\partial}{\partial x_i}\left(\frac{a_t}{\sigma_\kappa}+a\right)\frac{\partial\kappa_\theta}{\partial x_i}-\epsilon_\theta \tag{2.83}$$

The equation for the dissipation rate of the temperature fluctuations squared

$$\begin{aligned}\frac{\partial\epsilon_\theta}{\partial t}+u_i\frac{\partial\epsilon_\theta}{\partial x_i}=&-2a\phi_i\frac{\partial T}{\partial x_i}-2af_{ij}\frac{\partial u_i}{\partial x_j}-a\psi_{ij}\frac{\partial^2T}{\partial x_i\partial x_j}-\frac{2}{3}a\frac{\partial\overline{u'T'}}{\partial x_i}\frac{\partial^2T}{\partial x_l^2}\\ &-2a\overline{\frac{\partial T'}{\partial x_i}\frac{\partial T'}{\partial x_j}\frac{\partial u_i'}{\partial x_j}}+\frac{\partial}{\partial x_j}\left(\frac{a_t}{\sigma_\epsilon}+a\right)\frac{\partial\epsilon_\theta}{\partial x_j}-B_{\epsilon2}\frac{\epsilon_\theta^2}{\kappa_\theta}\end{aligned} \tag{2.84}$$

In the equations above, the next notations are used.

$$\kappa:=\frac{1}{2}\overline{u_i'^2}\,,\qquad \epsilon:=\nu\overline{\left(\frac{\partial u'}{\partial x_j}\right)^2}$$

$$\kappa_\theta:=\frac{1}{2}\overline{T'^2}\,,\qquad \epsilon_\theta:=a\overline{\left(\frac{\partial T'}{\partial x_i}\right)^2}$$

$$
\begin{aligned}
m_{ij} &:= \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_j'}{\partial x_k}} + \overline{\frac{\partial u_k'}{\partial x_i}\frac{\partial u_k'}{\partial x_j}} - \frac{2}{3}\delta_{ij}\frac{\epsilon}{\nu} \\
r_{ijk} &:= \overline{u_k'\frac{\partial u_i'}{\partial x_j}} + \overline{u_j'\frac{\partial u_i'}{\partial x_k}} - \frac{2}{3}\delta_{jk}\frac{\partial \overline{u_n' u_i'}}{\partial x_n} \\
R_{ijk} &:= \frac{\partial^2 u_i}{\partial x_j \partial x_k} - \frac{\delta_{jk}}{3}\frac{\partial^2 u_i}{\partial x_l^2} \\
\phi_i &:= \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial T'}{\partial x_k}} \\
f_{ij} &:= \overline{\frac{\partial T'}{\partial x_i}\frac{\partial T'}{\partial x_j}} - \frac{\delta_{ij}}{3}\frac{\epsilon_\theta}{a} \\
\psi_{ij} &:= \overline{u_i'\frac{\partial T'}{\partial x_j}} + \overline{u_j'\frac{\partial T'}{\partial x_i}} - \frac{2}{3}\delta_{ij}\frac{\partial \overline{u_k' T'}}{\partial x_k} \\
\hat{u_i} &:= \overline{\frac{\rho' u_i'}{\rho}} \simeq -\beta_T \overline{u_i' T'}
\end{aligned}
$$

We recall that the Reynolds stress tensor $\overline{u_i' u_j'}$ is the averaged product of velocity fluctuations and the turbulent heat flux density $\overline{u_i' T'}$ is the averaged product of velocity and temperature fluctuations. We also define the stress tensor $\sigma_{ij} := -p\delta_{ij} + \mu S_{ij}$, where $S_{ij} := \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}$ is the velocity deformation tensor.

2.4.3 Model transport equations

According to [13]-[16], the correlation functions above, i.e. the ensemble-averaged products of variables, are expressed via the velocity and temperature spatial derivatives as well as via κ, ϵ, κ_θ and ϵ_θ fields.

Approximation of the correlation functions of the hydrodynamic equations

With respect to the velocity spatial derivatives the second order approximation of the Reynolds stress tensor gives

$$
-\overline{u_i' u_j'} + \frac{2}{3}\delta_{ij}\kappa = \nu_t S_{ij} - C_\kappa \sqrt{\frac{\nu\kappa}{\epsilon}}\frac{\kappa^2}{\epsilon} R_{mij} n_m \,, \qquad n_m := \frac{u_m}{\sqrt{u_k^2}} \tag{2.85}
$$

The coefficients of the expansion are combinations of the turbulent quantities satisfying the necessary asymptotic behavior and disappearing when turbulence disappears. The last term in the Reynolds stress tensor equation is

important when the effects like the secondary flows in straight channels take place. When this term is neglected the equation comes to the well known Boussinesq relationship with the coefficient ν_t interpreted as eddy viscosity,

$$\nu_t := C_\mu \frac{\kappa^2}{\epsilon} \tag{2.86}$$

$$-2\nu m_{ij} = C_{\epsilon 1}\nu_t \frac{\epsilon}{\kappa} S_{ij} - C_{\epsilon 3}\kappa \sqrt{\frac{\nu\kappa}{\epsilon}} \frac{\partial S_{ij}}{\partial x_m} l_m \ , \qquad l_i := \frac{\partial \kappa}{\partial x_i} \Big/ \sqrt{\left(\frac{\partial \kappa}{\partial x_j}\right)^2} \tag{2.87}$$

$$r_{ijk} = -C_{\epsilon 4}\nu_t R_{ijk} \ , \qquad 2\overline{\frac{\partial u_i'}{\partial x_j}\frac{\partial u_i'}{\partial x_k}\frac{\partial u_k'}{\partial x_j}} = C_{\epsilon 5}\frac{\epsilon}{\kappa} S_{ij}^2 - C_{\epsilon 6}\nu R_{ijk}^2 \tag{2.88}$$

$$\hat{u}_i := \frac{\overline{\rho' u_i'}}{\rho} \simeq -\beta_T \overline{u_i' T'} \tag{2.89}$$

The second term in the right-hand side of the equation (2.81) takes into account the effect of the pressure gradient on the turbulent energy generation. As the velocity is small in comparison with u_i, the second term in the right-hand side of the equation (2.81) can be approximated in the following way:

$$\frac{\hat{u}_i}{\rho}\frac{\partial \sigma_{ik}}{\partial x_k} \simeq -\frac{\hat{u}_i}{\rho}\frac{\partial p}{\partial x_i} \simeq \frac{\beta_T \overline{u_i' T'}}{\rho}\frac{\partial p}{\partial x_i} \tag{2.90}$$

. In the equations above C_μ , C_κ , $C_{\epsilon 1}$, $C_{\epsilon 2}$, $C_{\epsilon 3}$, $C_{\epsilon 4}$, $C_{\epsilon 5}$, $C_{\epsilon 6}$, σ_κ, σ_ϵ are the model constants, n_i and l_i are the unit vectors collinear to the velocity vector and turbulent energy gradient respectively. The model constant values are: $C_\mu = 0.09$, $C_\kappa = 0.5$, $C_{\epsilon 1} = 1.5$, $C_{\epsilon 2} = 1.9$, $C_{\epsilon 3} = 0.02$, $C_{\epsilon 4} = 2$, $C_{\epsilon 5} = 0.09$, $C_{\epsilon 6} = 24$, $\sigma_\kappa = \sigma_\epsilon = 1.4$. The relationships cited above make the equation system (2.76), (2.79), (2.81), (2.82) closed and applicable for simulating isothermal fluid flows. This equation system was called as the second-order κ - ϵ turbulence model [13]-[16]. The main feature of this model is that it doesn't contain any model functions. The results of simulating liquid flows in different channels were in good agreement with the experimental data [13]-[16]. When the effects similar to the secondary flows in straight channels are not presupposed, the term in the Reynolds stress tensor enclosed the second order velocity spatial derivatives can be omitted. In this case the equation system can be called as the first-order κ - ϵ turbulence model [13]-[16].

Approximation of the correlation functions of the heat-exchange equations

The first order approximation with respect to the temperature spatial derivatives is used for the turbulent heat flux density:

$$-\overline{u_i' T'} = a_t \frac{\partial T}{\partial x_i} \tag{2.91}$$

Then

$$\begin{aligned}
-2a\phi_i &= B_{\epsilon 1}\frac{a_t}{\tau_h}\frac{\partial T}{\partial x_i} - B_{\epsilon 3}\frac{a_t}{\tau_h}L_{h2}\frac{\partial^2 T}{\partial x_i \partial x_j}l_j \\
\psi_{ij} &= -B_{\epsilon 4} a_t F_{ij} \\
2\overline{\frac{\partial T'}{\partial x_i}\frac{\partial T'}{\partial x_j}\frac{\partial u_i'}{\partial x_j}} &= B_{\epsilon 5}\frac{\epsilon}{\kappa}\left(\frac{\partial T}{\partial x_i}\right)^2 - B_{\epsilon 6}\frac{a}{Pr^{0.25}}F_{ij}^2 \\
-2af_{ij} &= D_\epsilon \frac{\kappa \epsilon_\theta}{\epsilon} S_{ij}
\end{aligned}$$

In the equations above

$$F_{ij} := \frac{\partial^2 T}{\partial x_i \partial x_j} - \frac{\delta_{ij}}{3}\frac{\partial^2 T}{\partial x_l^2} \tag{2.92}$$

In the most common way, the eddy heat diffusivity a_t can be determined as the product of the velocity and length scales characteristic for turbulence:

$$a_t := C_\lambda \sqrt{\kappa} L_{ef} \tag{2.93}$$

The square root of κ can naturally be taken as the velocity scale. In determination of the eddy viscosity the ratio of κ^2 to ϵ was used as the length scale but in determination of eddy heat diffusivity a_t the length scale should take into account as well the effect of the heat-exchange characteristics of turbulence i.e. κ_θ and ϵ_θ respectively. Moreover, if the velocity fluctuations exist together with the temperature gradient the temperature fluctuations exist necessarily. This circumstance should be taken into account too. The length scale entering in the eddy heat diffusivity a_t is determined as the effective length scale L_{ef} expanded via the length scales of different magnitudes:

$$L_{ef} = \begin{cases} L_e & L_e > L_0 \\ L_0 & L_e \le L_0 \end{cases}, \tag{2.94}$$

$$L_e := L_h - f_{Pr} L_{h1} + L_0$$

$$L_h := \sqrt{\kappa}\tau_h\,, \quad L_{h1} := \sqrt{\frac{\nu\kappa_\theta}{\epsilon_\theta}}\,, \quad L_0 := 0.2a(\nu\epsilon)^{-0.25}\,, \quad \tau_h := \sqrt{\frac{\kappa}{\epsilon}\frac{\kappa_\theta}{\epsilon_\theta}}$$

τ_h is the time scale. The function entering in the determination of the length scale depends on the Prandtl number as below

$$f_{Pr} = \frac{1 + 0.7\exp(-R) - \exp(-R_1)}{1 + \exp(-R_2)} \tag{2.95}$$

$$R = \left(\frac{1.2}{Pr}\right)^2, \quad R_1 = \sqrt{\frac{110}{Pr}}, \quad R_2 = \sqrt{\frac{25000}{Pr}} \tag{2.96}$$

In the determinations above the additional length scale is used as well:

$$L_{h2} = \sqrt{\frac{a\kappa}{\epsilon}} \tag{2.97}$$

The model constant values are: $C_\lambda = 0.09$, $B_{\epsilon 2} = 1.9$, $B_{\epsilon 3} = 0.2$, $B_{\epsilon 4} = 2$, $B_{\epsilon 5} = 0.09$, $B_{\epsilon 6} = 24$, $D_\epsilon = 0.01$. In the list above $B_{\epsilon 1}$ is absent. It is considered here as the coefficient dependent on the Prandtl number in the following way:

$$B_{\epsilon 1} = 1 - \frac{1}{2}\frac{Pr^2}{1 + Pr^2} \tag{2.98}$$

When the correlation functions are replaced by their approximations the transport equations (2.79)-(2.84) read:

The momentum equation (Reynolds equation)

$$\rho\frac{\partial u_i}{\partial t} + \rho u_k\frac{\partial u_i}{\partial x_k} = -\frac{\partial}{\partial x_i}\left(p + \frac{2}{3}\rho\kappa\right) + \frac{\partial}{\partial x_j}\left[(\mu_t + \mu)S_{ij}\right] + \rho g_i - C_\kappa\frac{\partial}{\partial x_j}\left(\sqrt{\frac{\nu\kappa}{\epsilon}}\frac{\kappa^2}{\epsilon}R_{mij}n_m\right) \tag{2.99}$$

The heat conductivity equation

$$\rho C_p\left(\frac{\partial T}{\partial t} + u_k\frac{\partial T}{\partial x_k}\right) = \beta T\frac{Dp}{Dt} + \frac{\partial}{\partial x_k}\left[(\lambda_t + \lambda)\frac{\partial T}{\partial x_k}\right] + \frac{1}{2}\mu S_{ij}^2 + \rho\epsilon \tag{2.100}$$

The turbulent energy equation

$$\frac{\partial\kappa}{\partial t} + u_i\frac{\partial\kappa}{\partial x_i} = \frac{1}{2}\nu_t S_{ij}^2 - \frac{\beta_T a_t}{\rho}\frac{\partial T}{\partial x_i}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\frac{\nu_t}{\sigma_\kappa} + \nu\right)\frac{\partial\kappa}{\partial x_j} - \epsilon - C_\kappa\sqrt{\frac{\nu\kappa}{\epsilon}}\frac{\kappa^2}{\epsilon}\frac{\partial u_i}{\partial x_j}\frac{\partial^2 u_m}{\partial x_i\partial x_j}n_m \tag{2.101}$$

The last term in the right-hand side of the equation (2.101) can be omitted simultaneously with the last term in the right-hand side of the equation (2.99) if the effects similar to the secondary flows are not important.

The turbulent energy dissipation rate equation

$$\frac{\partial \epsilon}{\partial t} + u_i \frac{\partial \epsilon}{\partial x_i} = \frac{1}{2} C_{\epsilon 1} \nu_t \frac{\epsilon}{\kappa} S_{ij}^2 - C_{\epsilon 3} \kappa \sqrt{\frac{\nu \kappa}{\epsilon}} \frac{\partial u_i}{\partial x_j} \frac{\partial S_{ij}}{\partial x_m} l_m + C_{\epsilon 4} \nu \nu_t R_{ijk}^2 - \frac{2}{3} \nu \frac{\partial \overline{u_i' u_j'}}{\partial x_j} \frac{\partial^2 u_i}{\partial x_l^2} - C_{\epsilon 5} \frac{\nu \epsilon}{\kappa} S_{ij}^2 + C_{\epsilon 6} \nu^2 R_{ijk}^2 + \frac{\partial}{\partial x_j} \left(\frac{\nu_t}{\sigma_\epsilon} + \nu \right) \frac{\partial \epsilon}{\partial x_j} - C_{\epsilon 2} \frac{\epsilon^2}{\kappa} \quad (2.102)$$

The equation for the averaged temperature fluctuations squared

$$\frac{\partial \kappa_\theta}{\partial t} + u_i \frac{\partial \kappa_\theta}{\partial x_i} = a_t \left(\frac{\partial T}{\partial x_i} \right)^2 + \frac{\partial}{\partial x_i} \left(\frac{a_t}{\sigma_\kappa} + a \right) \frac{\partial \kappa_\theta}{\partial x_i} - \epsilon_\theta \quad (2.103)$$

The equation for the dissipation rate of the temperature fluctuations squared

$$\frac{\partial \epsilon_\theta}{\partial t} + u_i \frac{\partial \epsilon_\theta}{\partial x_i} = B_{\epsilon 1} \frac{a_t}{\tau_h} \frac{\partial T}{\partial x_i}^2 + \frac{\partial}{\partial x_j} \left(\frac{a_t}{\sigma_\epsilon} + a \right) \frac{\partial \epsilon_\theta}{\partial x_j} - B_{\epsilon 2} \frac{\epsilon_\theta^2}{\kappa_\theta} + \frac{1}{2} D_\epsilon \frac{\kappa \epsilon_\theta}{\epsilon} S_{ij}^2 - B_{\epsilon 3} a_t \sqrt{\frac{a \epsilon_\theta}{\kappa_\theta}} \frac{\partial T}{\partial x_i} \frac{\partial^2 T}{\partial x_i \partial x_j} l_j + B_{\epsilon 4} a a_t F_{ij}^2 - \frac{2}{3} a \frac{\partial \overline{u'T'}}{\partial x_i} \frac{\partial^2 T}{\partial x_l^2} - B_{\epsilon 5} \frac{a \epsilon}{\kappa} \left(\frac{\partial T}{\partial x_i} \right)^2 + B_{\epsilon 6} \frac{a^2}{Pr^{0.25}} F_{ij}^2 \quad (2.104)$$

2.4.4 Simulating the natural convection boundary layer along a vertical flat plate

To study the turbulent natural convection boundary layer in air along a vertical plate, the natural convection inside a two-dimensional rectangular cavity inserted in air medium was really simulated by means of the proposed model [27, 22]. The cavity was 8.2 meter in height and 2 meter wide. Starting 0.2 meter from the bottom, one side wall of the cavity was kept at the uniform constant temperature $T_w = 60^o C$. All the other solid walls were kept at the ambient temperature of air $T_0 = 16^o C$. The temperature values of T_w and T_0 correspond to the conditions of the experiment [27]. At the upper liquid boundary, the normal derivative of the temperature was equal to zero. The following boundary conditions were imposed for other physical quantities as well.

The Dirichlet zero boundary condition was used for the velocity, turbulent energy and temperature fluctuations at the solid walls and for the velocity component V, normal to the gravity acceleration vector, at the upper liquid

boundary considered as the dividing line between the cavity and infinite volume of air medium. The Dirichlet zero boundary condition was used as well for the dissipation rates ϵ and ϵ_θ at the bottom and side wall opposite to the heating wall (no turbulence at the walls). Neumann zero boundary condition was used: a) for all the quantities at the top liquid boundary except the velocity component V; b) for the dissipation rates ϵ and ϵ_θ at the heating wall; c) for the pressure at all the boundaries.

For all the quantities, the zero initial conditions were imposed except the turbulent energy k and dissipation rate ϵ. For the last quantities, the following initial conditions which may be considered as an initial disturbance were imposed:

$$\kappa = 15U_R^2 \exp(-\frac{y}{0.01}) , \qquad \epsilon = 0.3\frac{U_R^4}{\nu_0} \exp(-\frac{y}{0.01}) \tag{2.105}$$

Laminar natural convection boundary layer

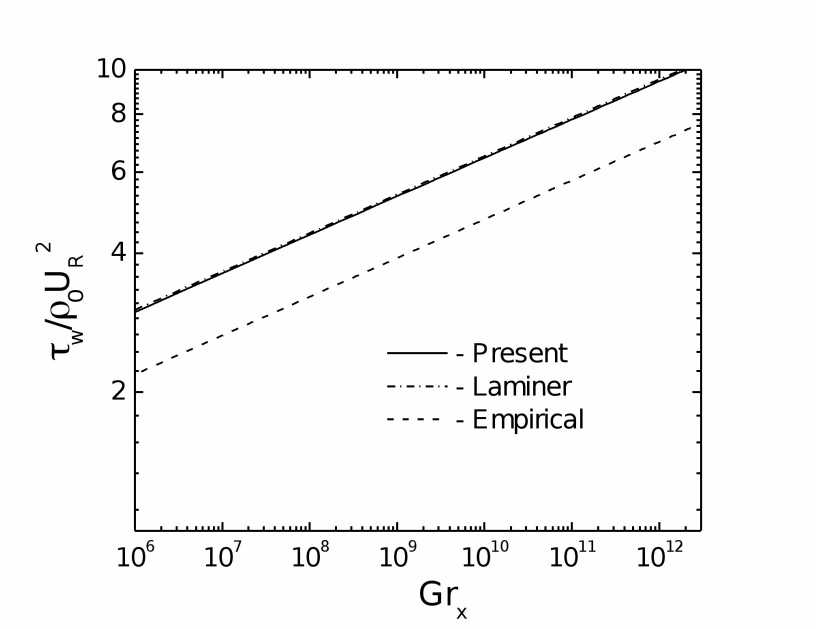

Figure 2.6: Wall shear stresses for laminar natural convection ('Empirical' – for turbulent convection)

With aim of testing the numerical analysis scheme and choosing the parameters for the presentation of the results in non-dimensional form, the simulation of the laminar natural convection boundary layer had been carried

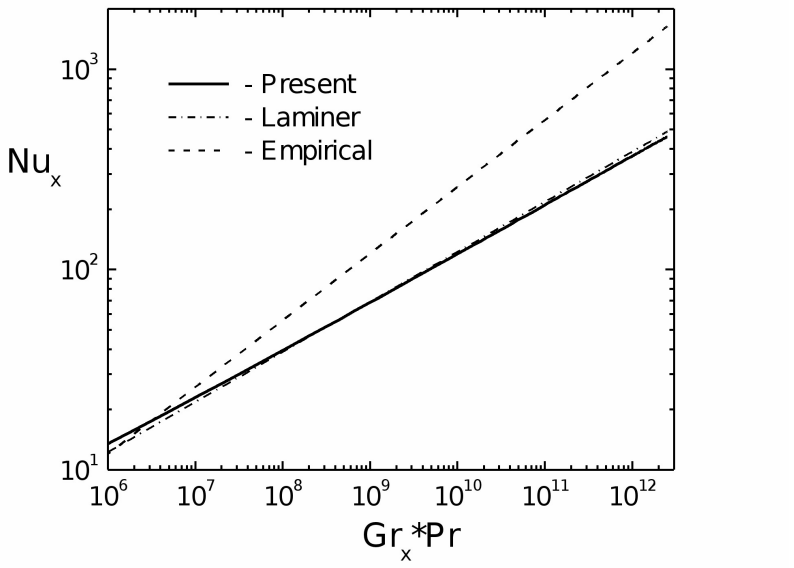

Figure 2.7: Heat transfer rates for laminar natural convection ('Empirical' – for turbulent convection)

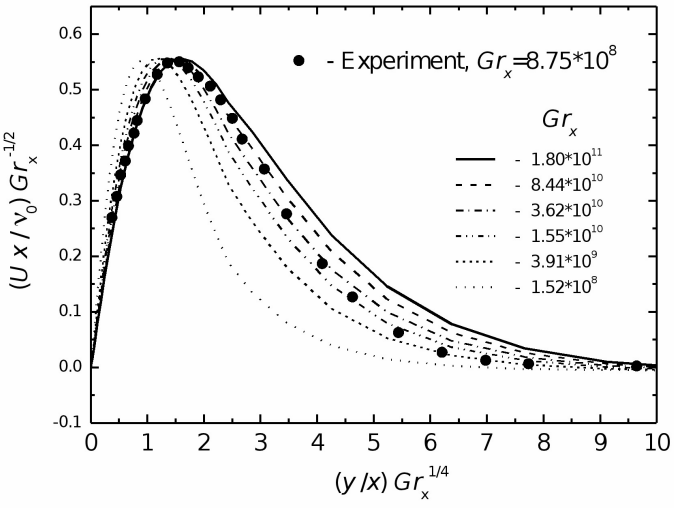

Figure 2.8: Velocity profiles in the laminar boundary layer

out at first. Figure 2.6 shows the coincidence of calculated and theoretical results for the wall shear stress τ_w in relation to the Grashof number Gr_x. The

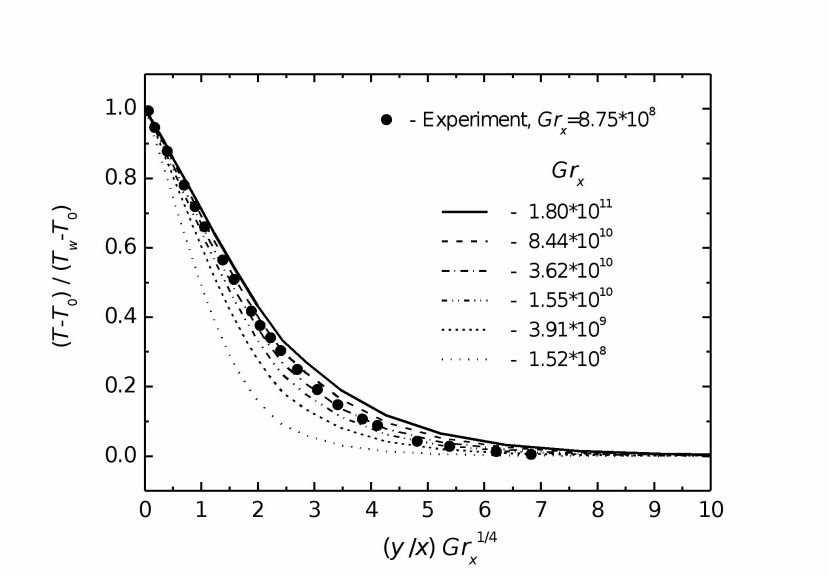

Figure 2.9: Temperature profiles in the laminar boundary layer

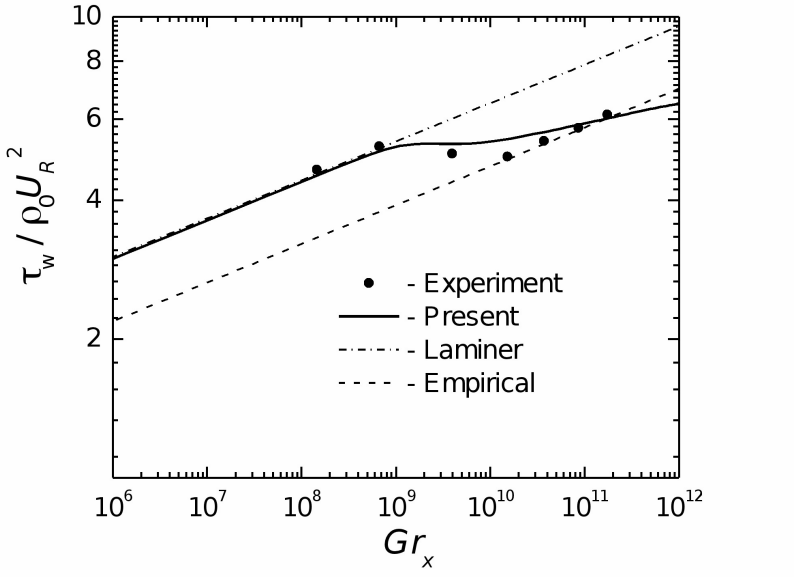

Figure 2.10: Wall shear stresses

theoretical results for the laminar convection are described by the equation

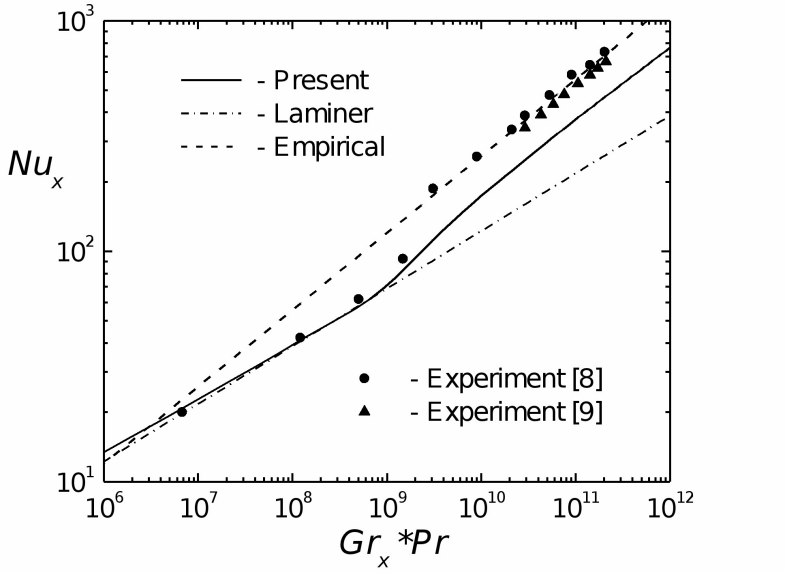

Figure 2.11: Heat transfer rates

[27]:

$$\frac{\tau_w}{\rho U_R^2} = 0.953 Gr_x^{1/12} \tag{2.106}$$

The empirical results for the turbulent natural convection are shown also for comparison. These results are described by the formula [27]:

$$\frac{\tau_w}{\rho U_R^2} = 0.684 Gr_x^{1/11.9} \tag{2.107}$$

Figure 2.7 compares the calculated and theoretical results for heat transfer rates in the relation between Nusselt, Nu_x, and Rayleigh, $Gr_x Pr$, numbers for the laminar convection. The theoretical results for the laminar boundary layer are described by the equation [27]:

$$Nu_x = 0.387 (Gr_x Pr)^{1/4} \tag{2.108}$$

The slope of the theoretical line representing equation (2.108) in logarithmic coordinates is a little larger than the slope of the calculated one. For comparison, the empirical heat transfer rates for the turbulent natural convection boundary layer are shown in Figure 2.7 as well. These empirical results are described by the formula [27]:

$$Nu_x = 0.120 (Gr_x Pr)^{1/3} \tag{2.109}$$

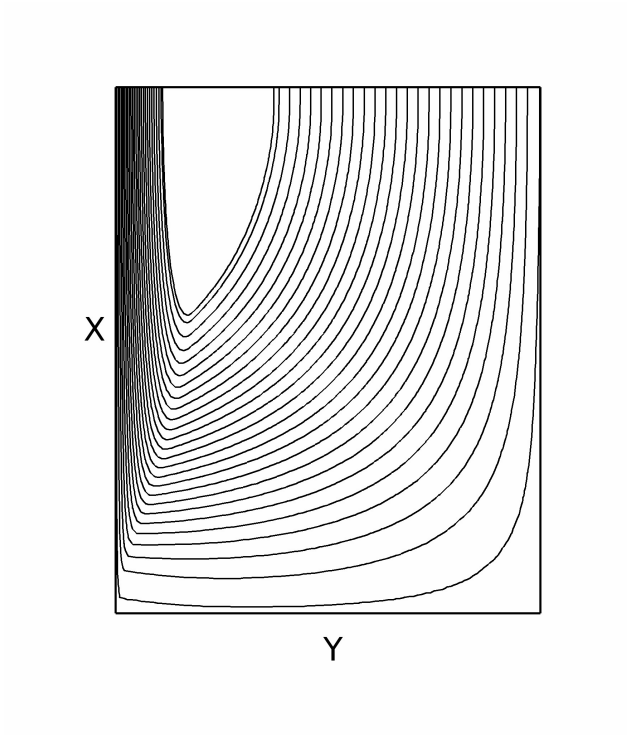

Figure 2.12: Stream lines of the natural convection in the cavity. The left side wall is heated

The calculated velocity and temperature profiles in the laminar boundary layer are shown in Figs. 2.8 and 2.9 respectively and compared with the experimental profiles indicated by dots. The calculated results generalized in conventional variables [27] are not described by the unique curves in the wide range of Grashof number.

Turbulent natural convection boundary layer

The results of the numerical simulation of the turbulent natural convection boundary layer obtained by means of the four parametric turbulence model (2.99)-(2.104) by neglecting the last terms in the right-hand side of the equations (2.99), (2.101) i.e. by neglecting the second order term in the relationship for the Reynolds stress tensor, are presented below. Practically, the same results were obtained by taking into account this second order term. The distributions of the calculated wall shear stresses and heat transfer rates for the turbulent natural convection boundary layer are shown in Figs. 2.10 and 2.11 respectively, together with the experimental data. Both figures demonstrate well that the turbulence model gives the same transition region between the

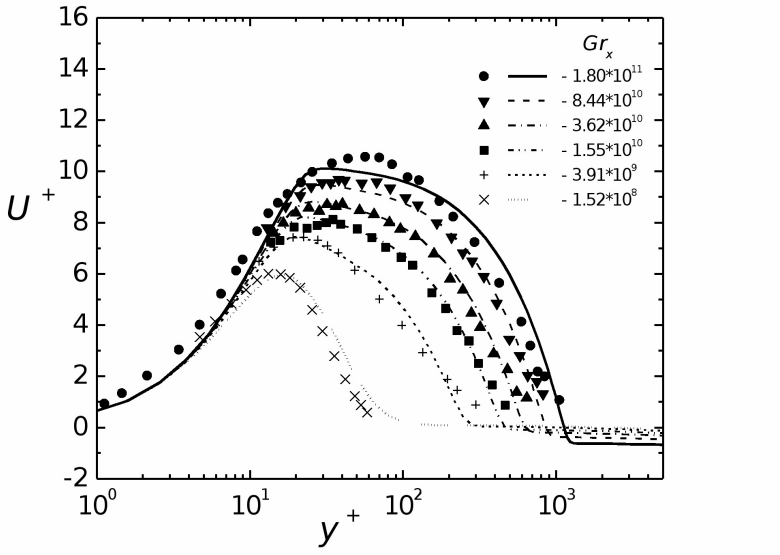

Figure 2.13: Mean velocity profiles. Lines denote calculated data. Dots denote experimental data [27]

laminar and turbulent regimes in terms of Grashof number values as it was observed in the experiment [27]. The calculated wall shear stresses agree satisfactorily with the experimental values measured up to $Gr_x = 2 \times 10^{11}$. (In the experiment [27], the flat surface generating heat flux was a plate 4 m high.) For $Gr_x > 2 \times 10^{11}$ the calculated wall shear stresses deviate from the empirical dependence (2.109). This effect can be explained by the interference between ascending and descending flows in the upper part of the cavity as it can be seen from the pattern of streamlines, Figure 2.12. To prevent this interference a partition was installed at the upper part of the experimental assembly [27]. It is necessary to emphasize that the empirical dependence (2.109) was drawn via the four empirical values measured at high Grashof numbers, see Figure 2.10.

The dependence on the Grashof number of the calculated heat transfer rates in the turbulent boundary layer has the same character as the experimental one but with values which are approximately 1.4 times lower than the experimental values, see Figure 2.11. Figure 2.13 compares the calculated and experimental mean velocity profiles in the turbulent boundary layer as well as in the transition and laminar regions in the relation between U^+ and y^+. The calculated and experimental mean velocity profiles indicate the same boundary layer thickness along the heated wall but the velocity values

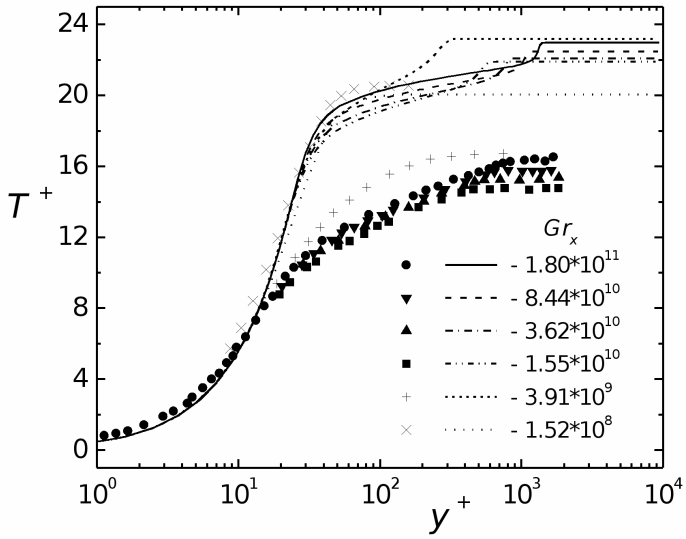

Figure 2.14: Mean temperature profiles. Lines denote calculated data. Dots denote experimental data [27]

and profile forms are slightly different. Figure 2.14 compares the calculated and experimental mean temperature profiles in the relation between T^+ and y^+. The calculated non-dimensional temperature profiles have practically the universal form inside the turbulent boundary layer. As a consequence of the differences in calculated and experimental heat transfer rates, the calculated temperature profiles normalized by the friction temperatures have the maximum temperature of about 1.4 times higher than the experimental value. The calculated and experimental profiles coincide in the laminar boundary layer. Figure 2.15 compares the calculated and experimental longitudinal velocity fluctuation profiles normalized by the friction velocity u_τ. The experimental profiles show the significant velocity fluctuations in the wall vicinity known as viscous sub-layer in the turbulent forced convection boundary layer bounded by $y^+ \leq 5$. The calculation results show the significantly larger wall neighborhood, $y+ \leq 10$, with zero velocity fluctuations. This is distinction in kind. But for the momentum transfer, the Reynolds stress $\overline{u'v'}$ values are more important. Figure 2.16 shows that the difference between calculated and experimental distributions of $\overline{u'v'}$ is not so large as among $\overline{u'u'}$ values.

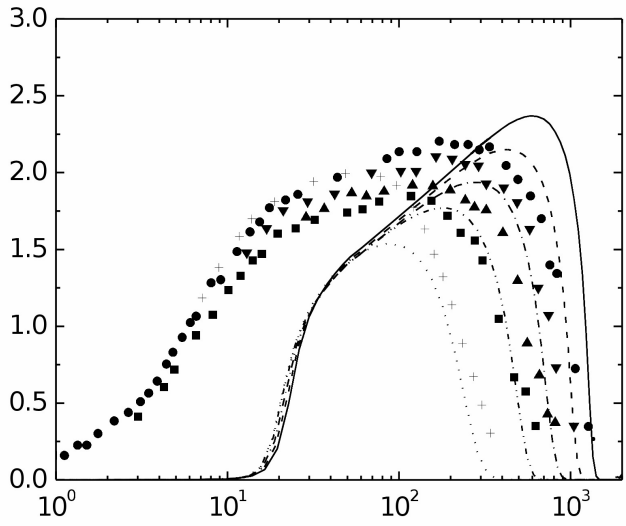

Figure 2.15: Profiles of velocity fluctuation intensities. Lines denote calculated data. Dots denote experimental data [27]

2.4.5 Simulating the natural convection boundary layer by means of the two parametric turbulence model using the turbulent Prandtl number

To understand better the distinction between the calculated and empirical heat transfer rates of Figure 2.11, the numerical simulation of the turbulent natural convection boundary layer was performed by means of four equations (2.99)-(2.102). Two equations, the equation for the turbulent energy (2.101) and dissipation rate (2.102), are necessary for the determination of the eddy momentum diffusivity. Instead of solving the equations (2.103) and (2.104), the eddy heat diffusivity was determined via the eddy momentum diffusivity by using the turbulent Prandtl number Pr_t. This model may be called two parametric. It is reasonable to expect that an averaged value of the turbulent Prandtl number will work well in such a complex flow as natural convection inside a rectangular cavity. The usually used value of Pr_t for air equal to 0.9 was implemented in the numerical simulation.

The distribution of the heat transfer rates calculated by means of the two-parametric turbulence model is shown in the Figure 2.17. The difference between calculated and experimental results does not exceed 20%. Of the same order difference appears for the calculated and experimental tem-

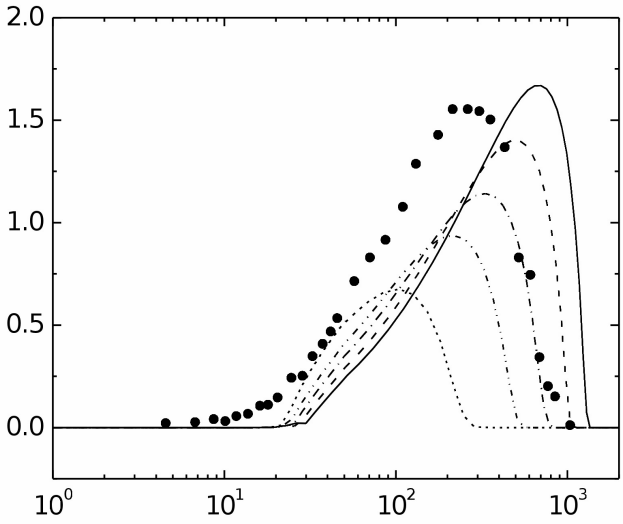

Figure 2.16: Profiles of Reynolds stresses. Lines denote calculated data. Dots denote experimental data [26]

perature distributions, Figure 2.18. These differences show on satisfactory agreement between experimental results and results obtained by means of the two-parametric turbulence model. Figure 2.19 compares the calculated and experimental turbulent heat flux distributions. Being equal to zero for $y^+ \leq 5$, the calculated and experimental heat fluxes practically coincide in the layer region $y^+ \geq 30$ and differ in the interval $5 < y^+ < 30$. This difference explains well the differences between the calculated and experimental heat transfer rates of Figure 2.17 and temperature distributions of Figure 2.18. Not shown here, the turbulent heat fluxes calculated by means of the four-parametric turbulence model differ more from experimental values. This explains bigger differences between the calculated and experimental heat transfer rates of Figure 2.11 and temperature distributions of Figure 2.14. The wall shear stresses, velocity profiles, velocity fluctuation intensities and profiles of Reynolds stresses calculated by means of the four-parametric turbulence model and shown on Figs. 2.10, 2.13, 2.15 and 2.16 respectively coincide with those calculated by means of the two-parametric turbulence model.

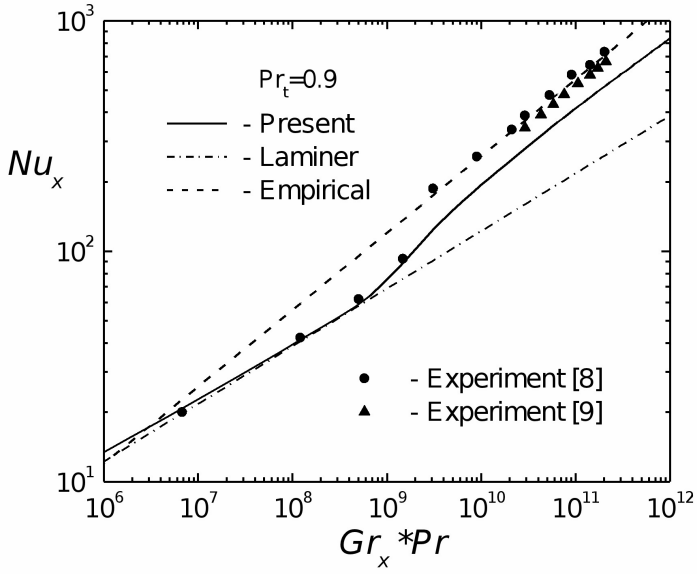

Figure 2.17: Heat transfer rates calculated by using the turbulent Prandtl number

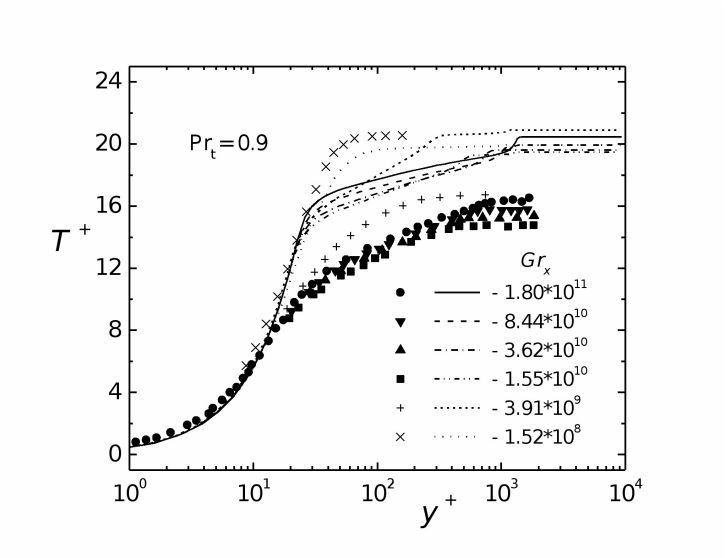

Figure 2.18: Mean temperature profiles calculated by using the turbulent Prandtl number. Lines denote calculated data. Dots denote experimental data [27]

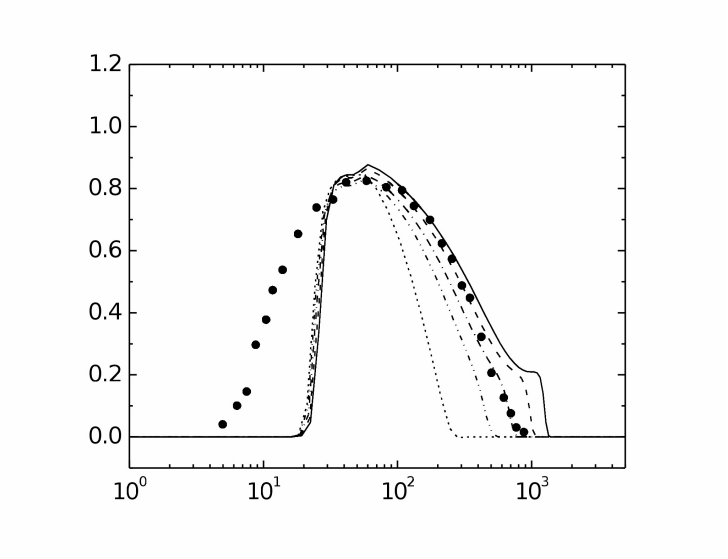

Figure 2.19: Profiles of turbulent heat fluxes. Lines denote calculated data. Dots denote experimental data [26]

2.4.6 Final considerations

In the previous work [17] the four-parametric κ - ϵ-κ_θ - ϵ_θ turbulence model for simulating the turbulent flows and heat exchange in fluids with different Prandtl numbers was proposed. The model was tested by simulating the forced convection flows in different channels and comparing the calculation results with known experimental data. In the present work, the proposed previously model has been edited for incompressible but non-isothermal fluid flows. This model contains six transport equations: momentum, energy equation, the equations for turbulent energy and its dissipation rate and equations for turbulent temperature fluctuations and dissipation of temperature fluctuations as well. The last two are necessary for determination of the turbulent heat diffusivity via the four turbulent characteristics of the flow: turbulent energy, temperature fluctuations and their dissipation rates. But the first four transport equations can be used for simulation of the heat exchange in liquids with different Prandtl numbers independently from the last two. In this case the turbulent heat diffusivity may be determined via the turbulent momentum diffusivity by means of the turbulent Prandtl number and the first four equations can be called as the two- parametric κ - ϵ turbulence model. Both these models have been tested by simulating the natural convection along a vertical flat plate held at constant temperature and comparing the simulation results with experimental data for air. In the case of two-

parametric model the commonly used value of the turbulent Prandtl number equal to 0.9 was employed. The natural convection along a vertical flat plate can be considered as one of the most interesting problems from the point of view of testing any turbulence model because it is characterized first of all by three different regimes present simultaneously: laminar, transition and turbulent. The transition point from laminar to turbulent regimes is obviously the most exactly measured characteristic of the flow. Both models have shown the same transition point coinciding with the experimental one. Both models give the same velocity and wall shear stress distributions being in a satisfactory agreement with the experimental ones. The principle difference is found in the velocity fluctuation distributions in the wall vicinity. While the experimental results show on the absence of the viscous sub-layer the calculated results fined large enough interval $0 \leq y^+ \leq 10$ where velocity fluctuations are equal to zero. In spite of that the difference between calculated and experimental distributions of Reynolds stresses responsible for momentum transfer is not so large and this explains the satisfactory agreement found for velocity and wall shear stress distributions. The difference of 40% between experimental values of the heat transfer rate and values found by means of the four-parametric turbulence model should be considered as large difference. It means that this model needs in further development. The difference less than 20% between experimental values of the heat transfer rate and values found by means of the two-parametric turbulence model can be considered as satisfactory agreement. This model can be recommended for simulating the natural convection flows of fluids with different Prandtl numbers.

Chapter 3

Advances in implementation of parallel computing features

3.1 MPI-PETSc implementation

3.1.1 Introduction

Code parallelization is nowadays very popular since it reduces CPU time and makes the use of numerical computations more attractive. It is of fundamental importance for three-dimensional time-dependent simulations for which huge computational resources and a great number of machine-hours are required. The previous version of FEM-LCORE was running only with one processor but a basic parallelization has been implemented in this new version so as to take advantage of multiprocessor architectures. In the future we plan to improve the current parallel version to take full advantage of supercomputing architectures such as the CRESCO-ENEA grid [2, 1].

The transition of a code from a monoprocessor to a multiprocessor architecture is a difficult and long task requiring great efforts. To make this passage more easily attainable, the MPI libraries and various tools made available in the scientific community have been used. However, the calls to external libraries may lead to a slowdown in the execution of the code which should not affect however the overall increase in performance obtained with parallel computing.

The de facto standard protocol for parallelization is called *MPI* (Message Passing Interface) [18]. We use the implementation of this protocol known as the OpenMPI library. MPI is a programming paradigm defined by APIs (Application Programming Interfaces) that allow different processes to communicate with one another through the transmission and reception of messages. Over the years, MPI has become the standard for parallel pro-

gramming on clusters and modern supercomputers. This protocol supports communications between two single processes (point to point) and global (collective) communications. The MPI library contains the specific communication language defined in the documents MPI-1 (1994) and MPI-2 (1996), with the aim of ensuring portability and ease of use.

The **OpenMPI** library is an open source implementation of MPI-1 and MPI-2 released under the BSD license [19]. OpenMPI does not represent an additional version of MPI, but corresponds to the union of three implementations: FT-MPI, LA-MPI, LAM-MPI, also with the contribution of PACX-MPI. Each implementation excels in one or more areas and the idea behind OpenMPI is to write a library that excels in as many areas as possible. In this report we plan to test this basic implementation and therefore we introduce some assessment parameters. The quality of a parallelization can be assessed by three parameters: speedup (S), efficiency (E) and scalability (s). The speed up is defined as

$$S = \frac{t_1}{t_n}\,, \tag{3.1}$$

which is the ratio between the execution time t_1 on a sequential architecture and the execution time t_n with n parallel architecture processors. The efficiency is defined as

$$E = \frac{S}{n} \cdot 100\ [\%]\,, \tag{3.2}$$

which is given by the ratio between the speedup S and n number of processors, expressed in percent. There are two notions of scalability: strong and weak. Strong scalability is defined as how the speedup varies with the number of processors for a fixed total problem size. We have perfect strong scalability if the speedup is linear. Weak scalability is defined as how the speedup varies with the number of processors for a fixed problem size per processor. A perfect weak scalability results in a time invariant behaviour with respect to a varying number of processors. The objectives of the MPI and Open-MPI libraries are high performance, high scalability and high portability.

In order to avoid writing the specific MPI commands and to facilitate the implementation of the parallel operations we use the **PETSc** library, containing the MPI commands embedded in matrix and vector algebraic operations. The PETSc (Portable Extension Toolkit for Scientific Computation) library is a set of data structures and functions for scientific applications, in particular for problems modeled by partial differential equations and solved by parallel algorithms [21]. This library can operate on sets of indices, vectors, matrices, preconditioners and linear/nonlinear solvers. It can be used to

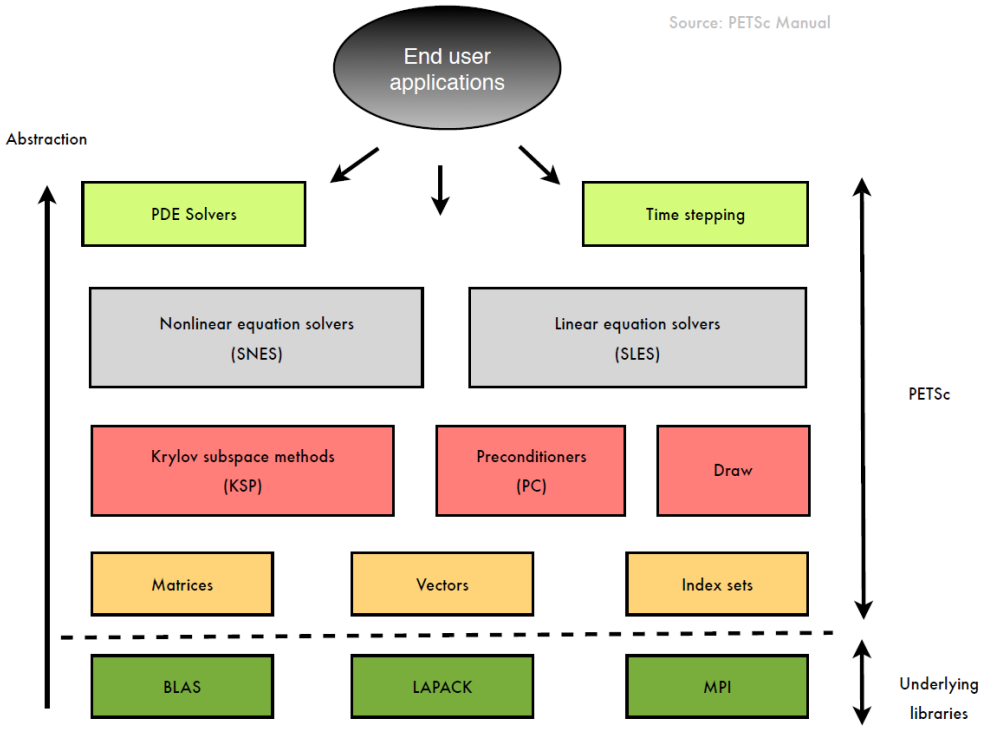

Figure 3.1: PETSc library

manage objects with standard C and FORTRAN languages through abstract interfaces (see Figure 3.1).

With its flexibility, the PETSc library is designed to split the effort between the tasks of parallelization and model development. The PETSc functions and variables use the MPI communication protocol among processes. Normal operations are guaranteed if working on a sequential architecture and their use does not prevent the explicit reference to the MPI functions for special operations. It is important to emphasize that this library does not operate any load balancing and does not generate grids.

The first and very important task to be carefully investigated in the parallelization with PETSc is mesh generation. In fact the mesh structure defines the degrees of freedom and therefore the structure of the discretized matrices and vectors to be solved in parallel. In our code the mesh generator is built on the **LibMesh library**, which is a platform for numerical simulation of partial differential equations using the finite element method on serial or parallel computers [9]. It is an open-source library developed to facilitate the use of parallel computing and mesh adaptive refinement. The use of LibMesh is particularly suited to multi-level simulations through a strategy

of adaptive refinement and coarsening of the grid discretization. LibMesh allows the use of different formulations of the finite element method such as Galerkin, Petrov-Galerkin and discontinuous Galerkin. In order to parallelize the problem, it operates a domain decomposition by partitioning the mesh on different processes. Each process can read the global mesh, but only operates on a partition of it. The use of the LibMesh communication between different processes allows the programmer to focus on the physics of the problem and to promote the reuse of the code. The LibMesh library allows to import functions from different libraries. For example one can interface with CUBIT for mesh generation and PARMETIS and METIS for domain decomposition. The use of derived classes can be used to interface with third party packages of linear solvers such as LasPack, PETSc and SLEPc.

3.1.2 Mesh generation for parallel computation

As previously described the first and very important task in the use of the PETSc library is mesh generation. In fact the definition of the degrees of freedom depends on how the mesh has been generated. The mesh input file is generated by the `gencase` program. It performs three important steps for which it makes use of the LibMesh library. In the first step `gencase` generates coarse grid mesh by reading from a file produced by an external program or by calling a function to generate a simple rectangular or parallelepiped grid mesh. In the second step the LibMesh library splits the domain on different processors and generates the multigrid operators for all levels in accordance with the specifications of the parallel requirements of the PETSC library. In the final step of the `gencase` code the multigrid mesh and all the operators are printed to various files.

The generation of the mesh starts reading a mesh input file. The LibMesh library can read a large number of format files. The main data structures of a mesh file are nodes and elements. Each node is defined by its position in space. The element defines the interface for a geometrical element based on a global numbering. Over each node LibMesh defines a degree of freedom. The class DofObject in this library connects the corresponding degree of freedom to a node in relation with the considered variable. The constraints on the assignment of nodes to different processors, along with the hierarchy levels, lead to a non-trivial implementation of the program (see Figure 3.2). A further complexity comes from considering various kinds of finite elements for different variables, for example linear elements for pressure and quadratic for velocity in the case of the Navier-Stokes equations. The refinement is done by using the standard midpoint refinement method. With this method one gets 2^d sub-elements out of one d-dimensional element. Also by refining a

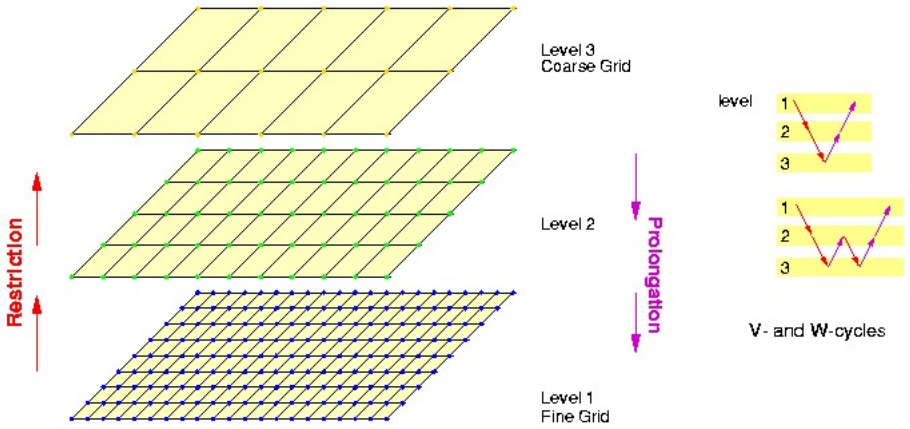

Figure 3.2: Multilevel mesh and multigrid V cycle

grid of elements of unique type one gets a mesh of the same type of elements. By operating according to this procedure we get a natural tree where every element has a pointer to the "father" and "an array of pointers to children". It is important to emphasize that in the case of refinement or coarsening the mesh is checked to eliminate any duplicate and/or orphan nodes.

The partitioning of the domain into several sub-domains related to processors is done by using the `Parmetis` library. In agreement with the PETSc library the order of rows in the matrix must be based on the rank of the processor. Also, in order to minimize any exchange time, all nodes belonging to the same processor must remain on the same processor at all levels. In order to obtain a correct partition it is important to start from a coarse mesh and then refine it. A bad partitioning can generate a lack of balance between different processors. In particular this latter problem can occur for large mesh sizes.

In the final step of the `gencase` program the multi-level mesh and all the multigrid operators are printed to various files. The multigrid operators are the sparse matrix patterns and the sparse non-square matrix restriction and prolongation operators at all levels. In the parallel PETSc libraries the sparse matrix and distributed vectors are loaded in memory in order to minimize the communication between the different processes. After the calculation of matrix data on individual processes, the PETSc library must perform communications together to ensure proper assembly of the global matrix. The PETSC assembly function must perform several actions with no waste of time. The remote data required for the local element matrix assembly should be made available to each local process and the assembly cycle should be only over the active elements on the local process with a smart evaluation

of element matrices that are then inserted in the global matrix.

The first and third steps are performed automatically by the library while the second step must be carried out through a code provided by the operator. Some of these instructions are carried out by LibMesh using parallel function calls, while others are performed by a single processor, like file printing. The file input/output is not parallel at the moment and is one of the objectives to be pursued in the future.

3.1.3 Parallel sparse matrices and vectors

PETSc currently provides two basic vector types: sequential and parallel (MPI based). Both of them are used in the code. The old solution (in time) of the discretized partial differential equation is stored as a sequential vector while all the other vectors involved in the parallel solution are parallel. To create a **sequential vector** `V` with `m` components, we use the command

```
VecCreateSeq(PETSC COMM SELF,int m,Vec *V)
```

A **parallel vector** `V` with a specified number of components that will be stored on each process can be created with the command

```
VecCreateMPI(MPI Comm c,int m_l,int m_g,Vec *V)
```

which creates a vector that is distributed over all processes in the communicator `c` , where `m_l` indicates the number of components to store on the local process, and `m_g` is the total number of vector components. Once all the values have been inserted with `VecSetValues()`, one must call

```
VecAssemblyBegin(Vec x);
VecAssemblyEnd(Vec x);
```

These commands perform any needed message passing of non-local components. In order to allow the overlap of communication and calculation, one can perform any series of actions between these two calls while the messages are in transition. The vector types are implemented in the directory

```
/contrib/matrix
```

in the files

```
numeric_vectorM
dense_vectorM, dense_vector_baseM
distributed_vectorM, dense_subvectorM
petsc_vectorM, laspack_vectorM.
```

In the `NumericVectorM` class we have the general definition of a vector as an interface to various vector types defined in external libraries. This is the basic class needed to call both sequential and parallel vectors implemented in the various libraries. Here we have implemented the Laspack and the PETSc library vectors suited for one processor and parallel execution respectively. These classes are implemented in the `petsc_vectorM` and `laspack_vectorM` files.

PETSc provides a variety of matrix implementations. We consider mainly the dense matrix and compressed sparse row storage matrix implementations. The **dense matrix** `A` is the standard matrix where all the components are memorized. This entity can be created by the commands

```
MatCreate(MPI Comm comm,Mat *A)
MatSetSizes(Mat A,int m_l,int n_l,int m_g,int n_g).
```

The global matrix dimensions, given by `m_g` and `n_g`, and the local dimensions, given by `m_l` and `n_l`, are specified so that PETSc completely controls memory allocation.

The dense matrix is not very useful for large dimensions since it needs too much memory. This is particularly true in the discretization of partial differential equations where the resulting matrix is full of zeros. Therefore the default matrix representation in the PETSc library is the general sparse format. In the **PETSc sparse AIJ matrix formats**, the nonzero elements are stored by rows, along with an array of corresponding column numbers and an array of pointers to the beginning of each row. The sparse matrix can be sequential or parallel.

A **sequential AIJ sparse matrix** `A` with `m_l` rows and `n_l` columns can be created with the command

```
MatCreateSeqAIJ(PETSC COMM SELF,int m,int n,
                int nz,int *row_length,Mat *A),
```

where `nz` or `row_length` can be used to preallocate matrix memory. One can set `nz=0` and `row_length=PETSC_NULL` in order to ask PETSc to control all matrix memory allocation. The dynamic process of allocating new memory and copying from the old storage to the new is very expensive. Thus, to obtain good performance when assembling an AIJ matrix, it is crucial to preallocate the necessary memory for the sparse matrix. The user has two choices for preallocating matrix memory. One can use the scalar `nz` to specify the expected number of non-zeros for each row. This is generally fine if the number of non-zeros per row is roughly the same throughout the matrix. If one underestimates the actual number of non-zeros in a given row, then

during the assembly process PETSc will automatically allocate additional needed space. However, this extra memory allocation can slow the computation. If different rows have very different numbers of non-zero entries, one should attempt to indicate the exact number of elements intended for the various rows with the optional array `row_length` of length `m`, where `m` is the number of rows. In this case, the assembly process will require no additional memory allocations if the `row_length` estimates are correct. If, however, the `row_length` estimates are incorrect, PETSc will automatically obtain the additional needed space with loss of efficiency. Using the array `row_length` to preallocate memory is especially important for efficient matrix assembly if the number of non-zeros varies considerably among the rows.

Parallel sparse matrices with the AIJ format are created with the command

```
MatCreateMPIAIJ(MPI Comm comm,int m_l,int n_l,int m_g,int n_g,
           int nd,int * n_diag, int  nod,int *n_offdiag,Mat *A).
```

The variable `A` is the newly created matrix, while the arguments `m_l`, `m_g`, and `n_g`, are the number of local rows and the number of global rows and columns, respectively. In the PETSc partitioning scheme, all the matrix columns are local and `n_l` is the number of columns corresponding to the local part of a parallel vector. One must ensure that these numbers are chosen to be compatible with the vectors. One can set `nd=0`, `nod=0`, `n_diag=PETSC_NULL`, and

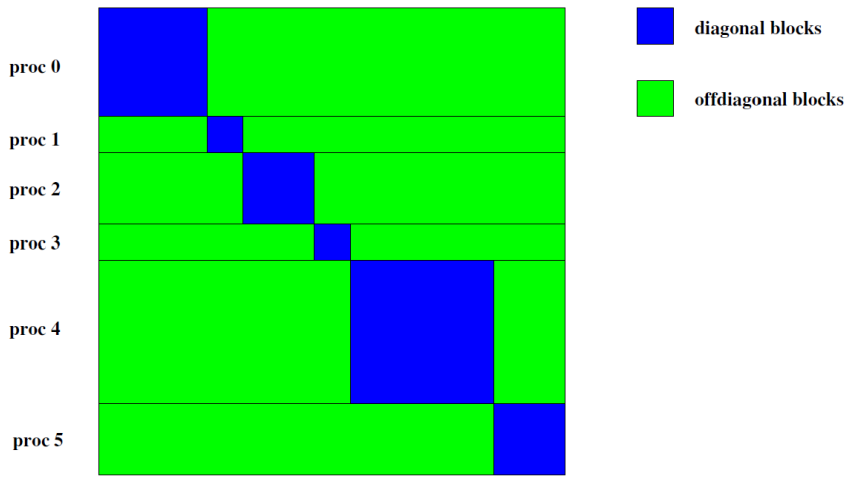

Figure 3.3: Diagonal and off-diagonal submatrices for 5 processors

`n_offdiag=PETSC_NULL` for PETSc to control dynamic allocation of matrix memory space. The matrix can be divided into a local square submatrix and

and off-diagonal part for each process. For a square global matrix, we define each row that is in a process to be the diagonal portion of the process. As one can see in Figure 3.3 a submatrix can be formed together with the corresponding columns to form a square submatrix that we call the diagonal part for that processor. All the rest formed by the off-diagonal portion is called off-diagonal rectangular submatrix. The rank in the MPI communicator determines the absolute ordering of the blocks. That is, the process with rank 0 in the communicator given to `MatCreateMPIAIJ` contains the top rows of the matrix. The `i`-th process in that communicator contains the `i`-th block of the matrix.

The matrices are implemented in the directory

```
/contrib/matrix
```

in the files

```
dense_matrix_baseM, dense_matrixM,dense_submatrixM
sparse_matrixM,sparse_MmatrixM
petsc_matrixM,petsc_MmatrixM
laspack_matrixM,laspack_MmatrixM.
```

The `dense_matrixM` files contain the implementation of dense matrices. These are used to assemble FEM elements which are in a form of a dense block, i.e., a dense matrix. These dense matrices are then inserted inside a parallel matrix. The sparse matrix implementation is in the class `sparse_matrixM` and then specialized in the PETSc and Laspack libraries.

3.2 Evaluation of computational time

The temporal analysis of the performance of a code is an efficient tool to assess the parallelization of the implemented instructions. In order to calculate the execution time of a given block of instructions one can simply set, at the beginning and at the end of the block, a call to a function that returns the time interval. It is clear that the numbers are characteristic of the machine. In order to have independent-machine values, CPU times obtained must be normalized with respect to a single processor value. In addition, in order to mitigate as much as possible any effects of disturbance due to the operating system, the values are averaged over ten simulations for a given number of processors.

With the purpose of time evaluation one can consider the standard library `ctime` and its functions. The `ctime` header refers to the library that contains the standard functions designed to obtain and manipulate information about

dates and times. The temporal resolution of the *clock_t* variable belonging to the `ctime` library is less than 10 ms for our machine. Therefore we decide to use the functions declared in `sys/time.h` which can measure time intervals of the order of microseconds.

3.2.1 Speedup for parallel vectors and sparse matrices

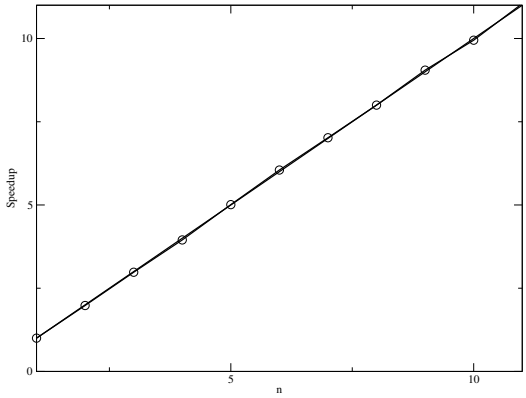

Figure 3.4: Speedup for the vector norm l_2 of a parallel vector as a function of processor number

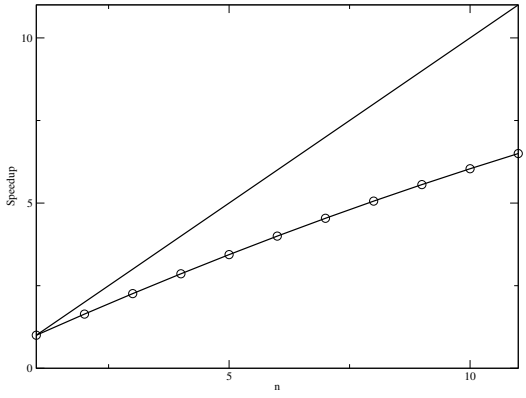

Figure 3.5: Speedup for the vector norm l_∞ of a matrix as a function of processor number

The first evaluation regards the calculation of the l_2-norm of a vector and the l_∞-norm of a matrix. The objectives are to verify that the obtained values were invariant with respect to the number of processes, as well a to show the performance as a function of the number of active processes. This

analysis is carried out by using the following simple procedure. We create two vectors, a serial one and a parallel one, with a fixed number of elements (of the order of 10^8). The matrix is created in a similar way. If the number of processors specified in the launch command is greater than 1 then libraries with parallel structures are used, otherwise the computation is serial. The norm used is l_2 for the vector and l_∞ for the matrix since the implementation of the l_2 matrix norm was not available.

Figure 3.4 shows the evolution of the normalized time required for evaluating the l_2-norm of the considered vector. The speedup is also shown in Figure 3.4. As we can see the implementation of the computation of the vector l_2-norm assumes a pattern which differs very little from ideal. We then evaluate the matrix norm. Figure 3.5 shows an improved performance as a function of the number of processes. In this case the behavior deviates from the ideal one, probably because of the increase in complexity of the functions involved in the management of matrices.

3.2.2 Speedup for Navier-Stokes and turbulence equations in an annular duct

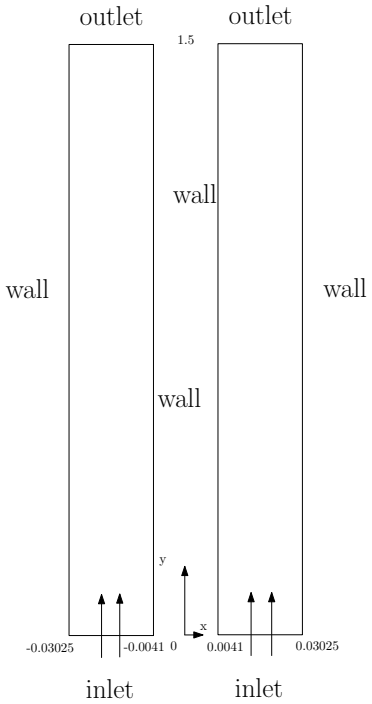

Figure 3.6: Geometry of the speedup test for Navier-Stokes and turbulence equations

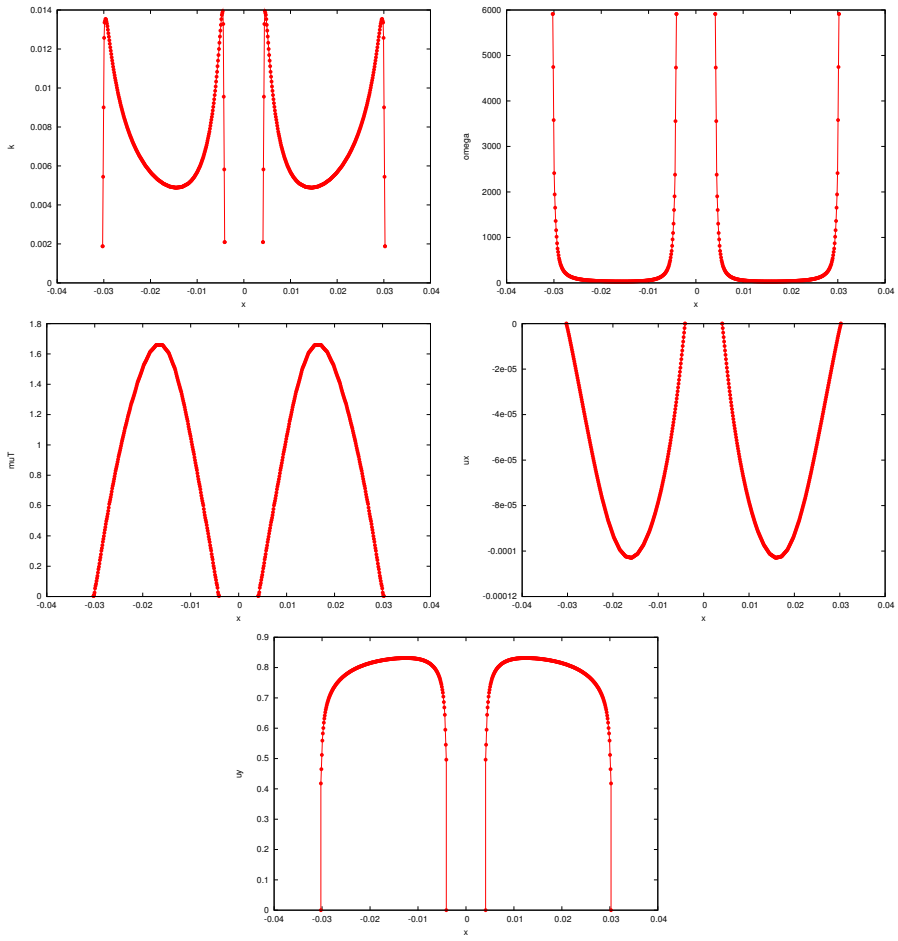

Figure 3.7: Annular duct test: $\kappa\,[(m/s)^2]$, $\omega\,[s^{-1}]$, $\mu_T\,[kg\cdot(m^{-1}s^{-1})]$, $u_x\,[m/s]$ e $u_y\,[m/s]$ as a function of $x[m]$ at $0.75m$ from the inlet section

The geometry of this problem is given in Figure 3.6. The simulation corresponds to the turbulent motion of lead within an annular duct with the inner and external radii of 0.0041 m and 0.03025 m, respectively. The main objective of these simulations is to check the computational time between the serial version and the parallel one. For simplicity the problem is simulated in axisymmetric geometry. In this case we use cylindrical coordinates with the symbolism of Cartesian coordinates, namely $u_r = u_x$ and $u_z = u_y$.

As explained above, the problem is well-posed if we provide the appropriate boundary conditions. We divide the boundary of our domain $\partial\Omega$ into three sub-domains: inlet, outlet and walls ($\partial\Omega = \Gamma_{Inlet} \cup \Gamma_{Wall} \cup \Gamma_{Outlet}$).

Focusing on the velocity field in the inlet section we have

$$u_x = 0\,\frac{m}{s}\,, u_y = 0.777\,\frac{m}{s}\,.$$

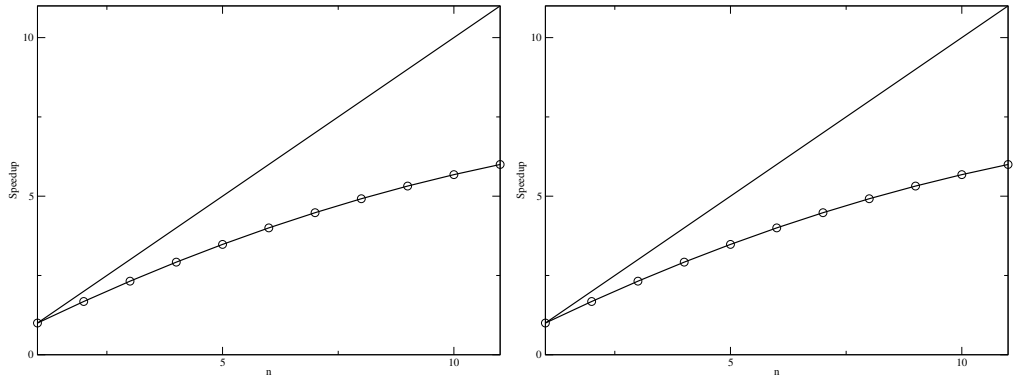

Figure 3.8: Speedup as a function of processor number for 3 (left) and 4 level (right) simulation

For the turbulent kinetic energy we have

$$\kappa = 1.5\bar{u}^2 I^2$$

where $\bar{u} = 0.777\, m/s$ is the average velocity and I is the turbulent intensity. For the turbulent kinetic energy dissipation rate ω we set

$$\omega = \frac{\sqrt{k}}{l}$$

with κ the turbulent kinetic energy and l the turbulence scale. For a fully developed flow as in this configuration we have

$$I = 0.16 Re_D^{-\frac{1}{8}}$$

where Re_D is the Reynolds number, evaluated on the hydraulic diameter D, and

$$l = 0.07D\,.$$

For the boundary conditions on the wall we first have to define the computational wall, placed at a distance y from the real wall. The non-dimensional distance y^+ is defined as

$$y^+ \equiv \frac{u_* y}{\nu}$$

where u_* is the speed friction on the wall, y is the distance from the closest wall and ν is the local dynamic viscosity of the fluid.

Therefore, the computational domain does not correspond to the geometric domain which is cut at a certain distance from the geometric walls. Usually in the boundary layer the velocity is approximated linearly near the wall and then with a logarithmic profile.

In the logarithmic layer the velocity can be assessed by the logarithmic law

$$u^+ = \frac{1}{k_v} ln(y^+) + B \tag{3.3}$$

for $y^+ > y_c^+$. In the viscous sublayer close to the wall ($y^+ < y_c^+$) we assume a linear profile $u^+ = y^+$. In (3.3) u^+ is the non-dimensional velocity, y^+ the non-dimensional distance from the wall, k_v the Von Karman constant (≈ 0.41) and C a constant (≈ 5.1). For these simulations y^+ was placed at 15 and the normal component along the y of the shear stress is assumed in agreement with the law defined in (3.3).

In Figure 3.7 the profiles of the variables as a function of $r(= x)$ at $z(= y) = 0.75\,m$ are reported. In particular we can remark the typical behavior of the velocity field $u_z = u_y$ in a turbulent flow, which assumes a flatter profile than a Poiseuille profile because of the greater amount of redistribution of the momentum. The simulations lead to results perfectly in agreement with the values of serial computation, further confirming the goodness of the changes in the code.

Figure 3.8 shows the speedup results of two simulations with 3 and 4 levels on the left and the right respectively. In both cases we can see that the behaviour detaches from the linear ideal profile as the number of processors increases.

Chapter 4

Simulation of assembly blockage events

4.1 Reactor computational model

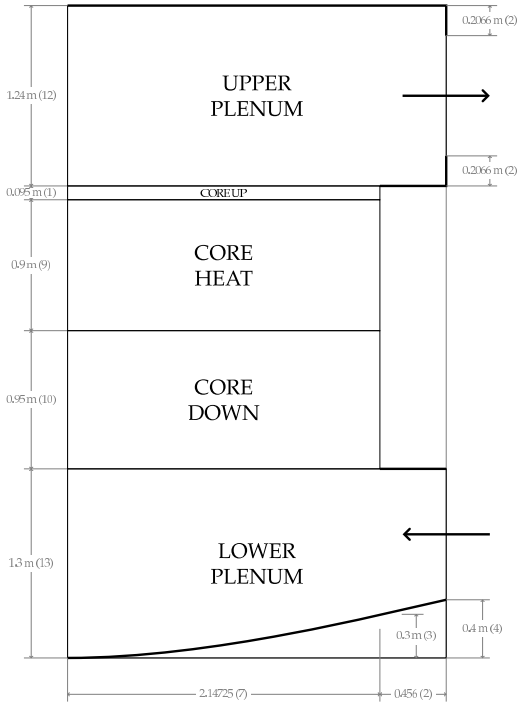

Figure 4.1: Vertical section of the reactor model.

In this chapter we present some results of the code concerning a blockage event occurring in the core. In a blockage event the coolant flow is prohibited

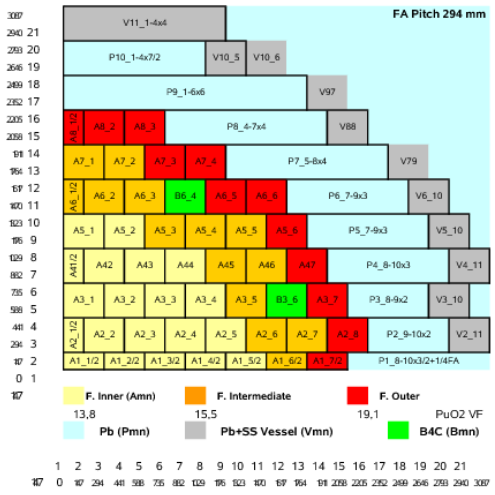

Figure 4.2: Desired horizontal power generation profile

in the corresponding assembly. This may happen in a reactor core by various reasons, like the failure of some components such as the cladding of some fuel rods. A peak temperature in a small portion of the core may in fact cause damage phenomena such as cladding ballooning, which may block a substantial portion of the flow area and restrict the flow of coolant. In such cases the heat removal process is strongly diminished and a temperature increase of the coolant is expected. We study both the closed and the open assembly cases. In the closed assembly situation, blockage occurring at any vertical quote of the core results in a zero velocity in all the considered assembly, as the LBE fluid is assumed to be nearly incompressible. In the open assembly case, convection heat transfer is diminished but still takes place and lower peak temperatures are expected with respect to the closed assembly case.

Let us consider the active (upper) and non-active (lower) core sections and the upper and lower plena, as shown in Figure 4.1. If we set the zero vertical coordinate at the lower core inlet, the core region goes from 0 m to 1.85 m. The active core (upper core) where heat is generated ranges between $H_{in} = 0.95$ m and $H_{out} = 1.85$ m. Below the core we have the lower plenum with the inlet between -0.9 m and 0 m. The lower plenum has an approximate hemispherical form with the lowest region at -1.2 m. Above the core for a total height of 1.2 m there is the upper plenum with the coolant

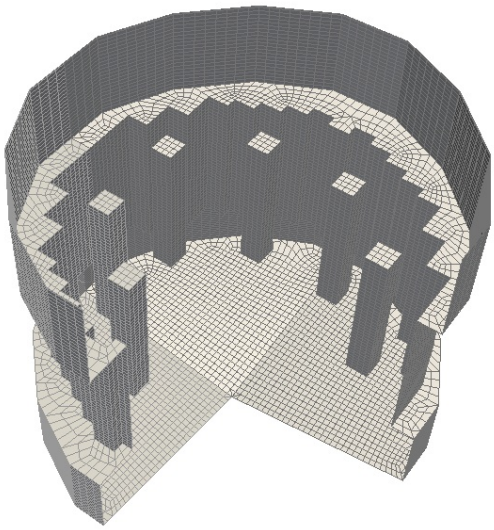

Figure 4.3: Core, lower and upper plenum reactor coarse boundary mesh

outlet.

The horizontal quarter section of the core is shown in Figure 4.2. Each fuel assembly consists of a 21×21 pin lattice. The overall number of assembly positions in the core is 170. Eight of these positions are dedicated to house special control rods (see [5]) and therefore the global number of fuel assemblies is 162. The transversal core area is approximately circular but the axial symmetry is not satisfied. Therefore, in order to predict the behaviour of the reactor, a three-dimensional simulation must be performed. However, we can argue from Figure 4.2 that two symmetry planes passing through the reactor axis can be identified so that only a quarter domain has to be taken into account.

The model design distributes the fuel assemblies in three radial zones: 56 fuel assemblies in the inner zone, 62 fuel assemblies in the intermediate zone and the remaining 44 fuel assemblies in the outer one. The power distribution factors, i.e. the power of a fuel assembly over the average fuel assembly power, are mapped in Figure 4.2. The maximum power factor is 1.17, while the minimum is 0.74.

We label the assemblies as in Figure 4.2; the first row is labeled `A1_i` for `i`$= 1, \ldots, 8$, the second row `A2_i` for `i`$= 1, \ldots, 7$ and so on. We remark that the fuel assembly configuration is not based on a Cartesian grid but rather on a staggered grid.

In Figure 4.3 the boundary reactor mesh is shown. The mesh is generated by using the GAMBIT mesh generator with geometrical dimensions taken from Figures 4.1 and 4.2. The generation of this mesh is not trivial. In fact care must be taken in such a way that every assembly is exactly discretized by entire cells. In this way all the data attributes related to a given assembly are associated to it exclusively. For details we refer to [3]. The reactor is

properties		value
Density ρ		$(11367 - 1.1944 \times 673.15) = 10562$
Viscosity μ		0.0022
Thermal conductivity κ		$15.8 + 108 \times 10^{-4}\,(673.15 - 600.4) = 16.58$
Heat capacity C_p		147.3

Table 4.1: Lead properties at T=400^o C

cooled by lead that enters at the temperature of 673.15 K. In Table 4.1 we report the lead physical properties at the inlet temperature. Since our model describes the reactor at the assembly level the sub-assembly composition is seen as a homogeneous medium. Data about the assembly geometry are reported in Table 4.2. In particular we note that the coolant/assembly ratio is 0.548. Each assembly has a square section with a side length of $L = 0.294$ m, as shown in Table 4.3 and this completely defines the horizontal core structure. For the vertical geometry we refer to Figure 4.1.

	area (m^2)
Pin area	370.606×10^{-4}
Corner box area	5.717×10^{-4}
Central box beam	2.092×10^{-4}
Channel central box beam area	12.340×10^{-4}
Coolant area	473.605×10^{-4}
Assembly area	864.360×10^{-4}
Coolant/Assembly ratio	0.5408

Table 4.2: Coolant assembly area ratio data

The heat power generation due to fission $W_0(x, y, z)$ is assumed to be given, where $W_0(x, y)$ is the two-dimensional distribution. At the middle section of the upper core $z = (H_{out} - H_{in})/2$ the desired cross-sectional heat power distribution is shown in 4.2. The heat power generation due to fission is a function of the horizontal and vertical plane as one can see in Figures 1.5 and 1.6 in Chapter 1. We remark the presence of the eight control assemblies inside the core which are used to house special control rods (see [5]) and where

no power is generated. In order to obtain the assembly averaged specific $\dot{q}_v$ and linear $\dot{q}_l$ heat power, the total core power is to be divided over 162 assemblies instead of 170. We obtain

$$\dot{q}_v = \frac{\dot{Q}}{A} = \frac{1.482 \times 10^9}{\times 0.294 \times 0.294 \times 162} = 1.0584 \times 10^8 \frac{W}{m^2}$$

and

$$\dot{q}_l = \frac{\dot{q}_v}{H_{out} - H_{in}} = \frac{1.0086 \times 10^8}{0.9} = 1.176 \times 10^8 \frac{W}{m}\,.$$

		value
Mass flow rate $\dot{m}$		$124539 Kg/s$
Heat Power $\dot{Q}_{tot}$		$1482.235 MW$
Number of Assemblies		170
Assembly length L		0.294m
Channel Equivalent Diameter D_{eq}		0.0129

Table 4.3: Core characteristic values at working temperature

In this chapter some tests are proposed which examine the cases of open and closed assemblies. We denote these two main Cases as A and B respectively. For the Case A of open assemblies, we investigate some blockage events occurring at different core heights. In particular, we consider a blockage occurring in a small area at the inlet of the upper core, i.e. at the beginning of the heated core part, and a blockage at the core outlet, i.e. at the end of the heated core, where the LBE fluid enters the upper plenum region. The variation of the height of the blockage area does not bring differences in the results for the Case B of closed assemblies, as the fluid is assumed to be nearly incompressible in our model. Thus, we will subdivide the discussion of the results into three cases:

- Case A1: open assemblies and blockage at the inlet of the heated core;
- Case A2: open assemblies and blockage at the outlet of the heated core;
- Case B: closed assemblies.

In order to enforce numerically the blockage condition, we simply set to zero all the velocity components in the assumed blockage volume. In all the following cases, we consider the blockage of the assembly with cross section centered at $x = 1.323$ and $y = 1.176$ in the simulated quarter domain. Due to the previously discussed symmetries, the blockage of four assemblies is then

simulated. We recall that the case of closed assemblies is enforced in the code by imposing the velocity fields on the lateral surface of each assembly to be parallel to the z-direction. The u and v components on those surfaces are set to zero, so that no cross flow takes place. On the other hand, when the assemblies are assumed to be open a flow exchange between them is allowed with non-vanishing transversal velocities.

As we have shown before, the reactor is divided into four regions: the lower plenum, the lower core, the upper core and the upper plenum. Let Ω_c be the core region and Ω_{lp}, Ω_{up} be the lower and the upper plenum respectively. In the lower and upper plenum we solve the three-dimensional Navier-Stokes and energy system while in the core we use the appropriate model described in Section 1.2.1.

The reactor regions of the core and the plena must be connected with appropriate yielding conditions which can be defined by conservation of mass and momentum equations. Since the mass flow rate at the core inlet must match the mass flow rate at the top of the lower plenum and the same holds for the core outlet section, then the z-component of the velocity field cannot be continuous due to a sudden variation of the cross section, depending on the occupation factor ratio r. The occupation factor ratio, which is assumed to be 0.5408, is the ratio between the coolant and the total assembly cross section areas. Since the volume coolant rate is continuous in all the reactor we define a new vector field

$$\mathbf{v}^* = \begin{cases} \mathbf{v} & \text{on } \Omega_{lp} \cup \Omega_{up} \\ \frac{\mathbf{v}}{r} & \text{on } \Omega_c \end{cases} \tag{4.1}$$

which is continuous across the reactor.

4.2 Configuration

In the `parameters.in` file we set the fluid properties as reported in Table 4.1 and the fission heat source as discussed above.

In `Equations_conf.h` we activate the Navier-Stokes equations and the energy equation with

```
#define NS_EQUATIONS
#define T_EQUATIONS.
```

Let us consider the class parameters inside the files `config/MGSNSconf.h` and `config/MGSTconf.h`. In `MGSNSconf.h` we set the parameter

```
#define CONST 1
```

in order to neglect the temperature dependence of density and viscosity. The results do not change too much as already seen in [4]. We set the LES option with $\alpha = 1$ as

```
#define ALPHA 1.
```

This option sets a simple Smagorinsky LES turbulence model. In `MGSTconf.h` we set the LES option $Pr_t = 0.85$ with

```
#define PRT 0.85
```

This sets the standard heat exchange model. Under the default settings both the momentum and the energy system are solved by using GMRES method with an ILU preconditioner.

The boundary conditions are rather complex due to the reactor geometry. For pressure and velocity boundary conditions one must edit the function

```
void MGSolNS::bc_read(
  double xp[],   // xp[]  node coordinates
  int normal[], // normal
  int bc_flag[]  // boundary condition flag
)
```

in the file `src/MGSolverNS3D.C`. In this file we set Dirichlet boundary conditions over the inlet boundary and symmetry conditions for the planes $x = 0$ and $y = 0$. The inlet has an assigned velocity profile in agreement with the initial conditions. On the outlet we enforce Neumann boundary conditions. Over the reactor walls we assume slip boundary conditions. The implementation of the boundary conditions is quite complex since the reactor surface is not always aligned with the axes.

For boundary conditions of the energy equation one must edit the function

```
void MGSolT::bc_read(
  double xp[],   // xp[]  node coordinates
  int normal[], // normal
  int bc_flag[]  // boundary condition flag
)
```

in the file `src/MGSolverT3D.C`. We set a Dirichlet inlet condition with temperature 573.15 K. Over the rest of the surface we set homogeneous Neumann boundary conditions, namely zero heat flux.

We want to set the inlet velocity $0.82\,m/s$ to be perpendicular to the inlet surface with zero initial pressure. We set this velocity distribution in the function

```
void MGSolNS::ic_read(
  double xp[],
  double u_value[]
)
```

inside the file `src/MGSolverNS3D.C`.

We initialize the inlet temperature to the value of 573.15 K inside the file `MGSolverT3D.C` in the function

```
void MGSolT::ic_read(
  double xp[],
  double u_value[]
) .
```

4.3 Inlet open core blockage test (A1)

We assume that the assemblies are open, so that a flow exchange between them with transversal velocities is permitted. In order to enforce numerically the blockage condition, we simply set to zero all the velocity components in the assumed blockage volume. We recall that the case of closed assemblies is enforced in the code by imposing the velocity fields on the lateral surface of each assembly to be parallel to the z-direction. The u and v components on those surfaces are set to zero, so that no cross flow takes place. On the other hand, when the assemblies are assumed to be open a flow exchange between them is allowed with non-vanishing transversal velocities. In this case, we consider the blockage of the assembly with cross section centered at $x = 1.323$ and $y = 1.176$ at the inlet of the heated core. In the blockage region near the first part of the heated core the coolant cannot flow freely. The flow cannot enter the assembly and therefore an increase of temperature is expected. In Figure 4.4 the temperature profile along the z-line at the center of the blocked assembly is shown in comparison with the normal condition of non blocked assembly. The temperature of the blocked assembly case at the top core is slightly higher than the case of absence of blockage. The blockage case is plotted in solid line and the latter in dotted line. In Figures 4.5-4.6 there is an overview of the temperature distribution T over various sections of the reactor. In Figure 4.5 one can see the temperature field over a vertical section of the reactor. In Figure 4.6 the overview of the temperature

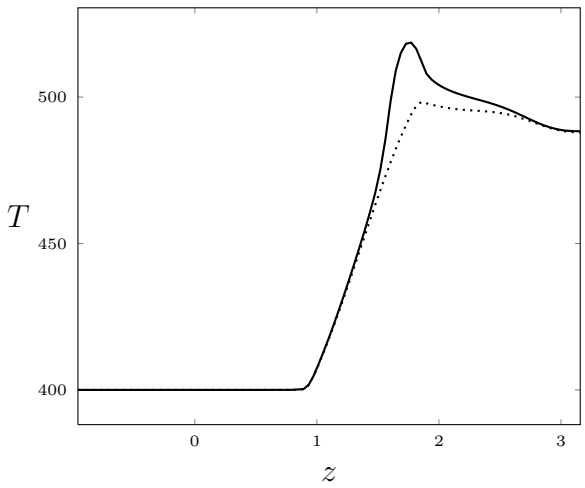

Figure 4.4: Case A1 (open assembly). Temperature along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ (solid line) and temperature along the same line in the absence of blockage (dotted line)

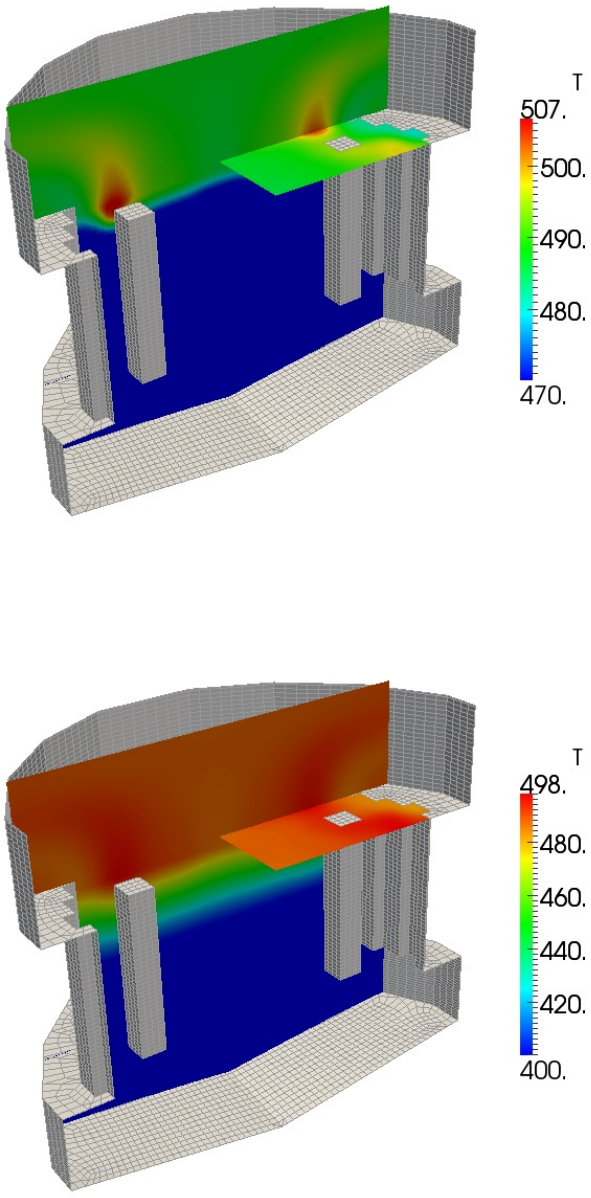

Figure 4.5: Case A1 (open assembly). Temperature field with blockage (top) and in the absence of blockage (bottom)

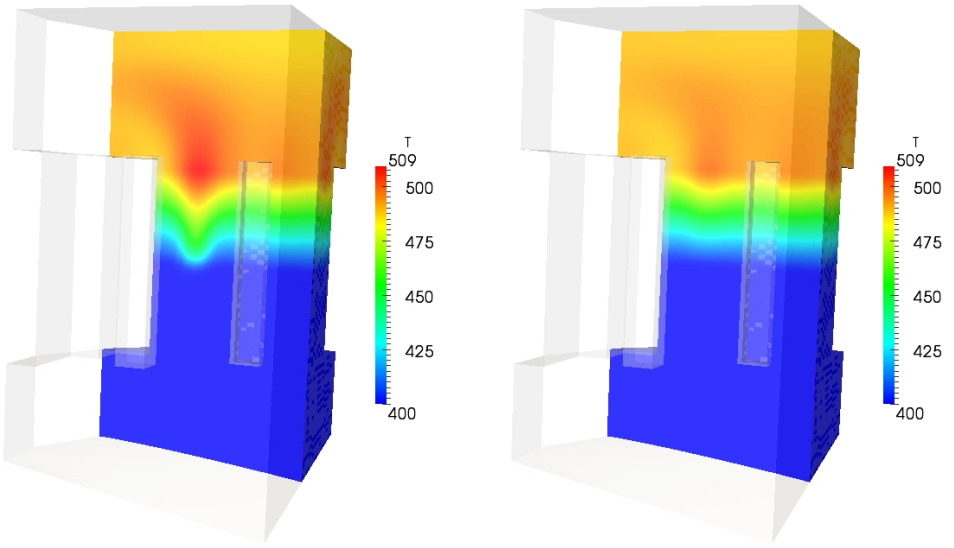

Figure 4.6: Case A1 (open assembly). Temperature field with blockage (left) and in the absence of blockage (right)

distribution T is over the section of the reactor that includes the blocked assembly. The temperature field with blockage is shown on the left and the temperature field in the absence of blockage on the right. The same temperature scale is assumed in order to see the comparison. In this case the increase of temperature diffuses all around the blocked assembly creating a small hot spot. One can note that the increase of temperature starts at the inlet of the heated core where the blockage is located. Then, since this is allowed in the open core model, heat propagates in all the directions by convection flows that cross the neighbouring assemblies. It is very interesting to see the temperature on different plane sections of the reactor as shown in Figures 4.7-4.8. In Figures 4.7 we can see, from top left to right bottom, the temperature distributions at the plane sections of the lower core, upper core inlet, upper core outlet and upper plenum. The heights correspond to $z = 0.6, z = 1.2, z = 1.8$ and $z = 2.4$ starting from the lower core inlet. We can see the increase of temperature immediately close to the blockage height up to the upper plenum inlet. In Figure 4.8 more temperature distributions in different plane sections are shown. The figure on the top is the case with blockage and on the bottom in the absence of blockage. In this case the temperature scale blends the hot spot of the blockage with the global temperature distribution.

The velocity field is shown in Figures 4.9-4.11. In Figure 4.9 one can see

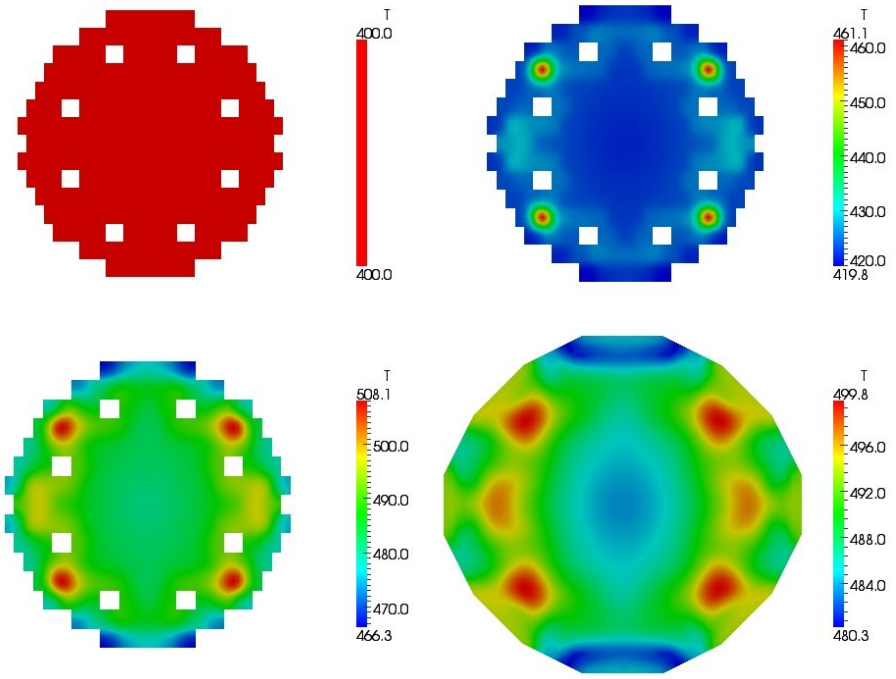

Figure 4.7: A1 (open assembly). Temperature distributions at the plane sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

the overall distribution of the z-component w^* as defined in (4.1). In Figure 4.9 the z-component of the velocity field w^* at the center of the blocked assembly is shown in comparison with the working condition of non blocked ones over a plane section containing the blocked assembly itself. The figure on the left describes the blocking case and the region with zero velocity is clearly visible. In Figure 4.30 the velocity profile along the z-line at the center of the blocked assembly $x = 1.323$ and $y = 1.176$ is shown with solid line, in comparison with the normal condition of non blocked assembly in dotted line. It is clear that before the blockage region the solid and dotted lines are very close. Then, in the blockage case, the solid line goes to zero and finally tends to the dotted line near the core outlet. The velocity on different plane sections of the reactor is shown in Figures 4.11-4.13. In Figures 4.11 we can see, from top left to right bottom, the velocity distributions at the plane sections of the lower core, upper core inlet, upper core outlet and upper plenum corresponding to $z = 0.6, z = 1.2, z = 1.8$ and $z = 2.4$ starting from the lower core inlet. We can clearly see zero velocity only in the section for $z = 1.2$ where the assembly is blocked. In Figure 4.13 more temperature

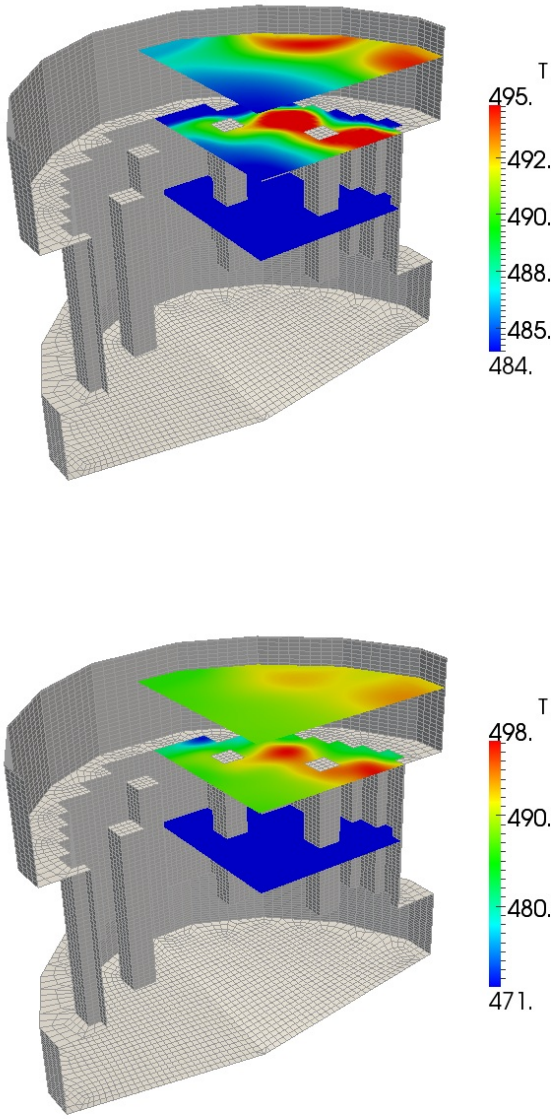

Figure 4.8: Case A1 (open assembly). Temperature field with blockage (top) and in the absence of blockage (bottom)

distributions in different plane sections in these two different situations are shown. The figure on the top is the case with blockage and on the bottom in the absence of blockage. In this case one can see the blockage only in the

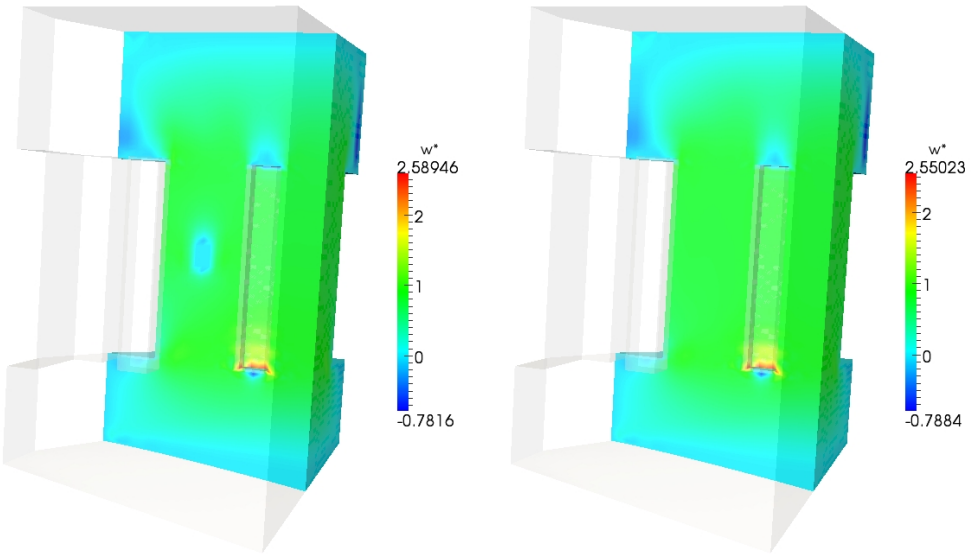

Figure 4.9: Case A1 (open assembly). Velocity w^* field with blockage (left) and in the absence of blockage (right)

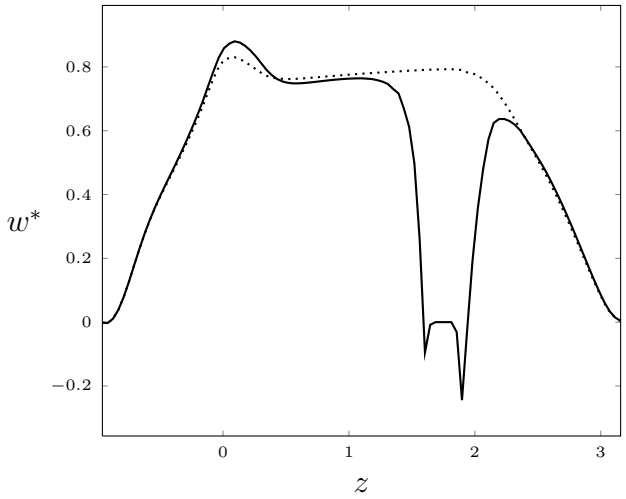

Figure 4.10: Case A1 (open assembly). z-component of the velocity field w^* along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ (solid line) and along the same line in the absence of blockage (dotted line)

bottom plane section of the reactor on the bottom of Figure 4.13. In Figure 4.12 the pressure along the same z-line of the blocked assembly is shown. The solid line represents the blocked case and the dotted line is for the absence of blockage. We can clearly see that the blockage leads to a discontinuous

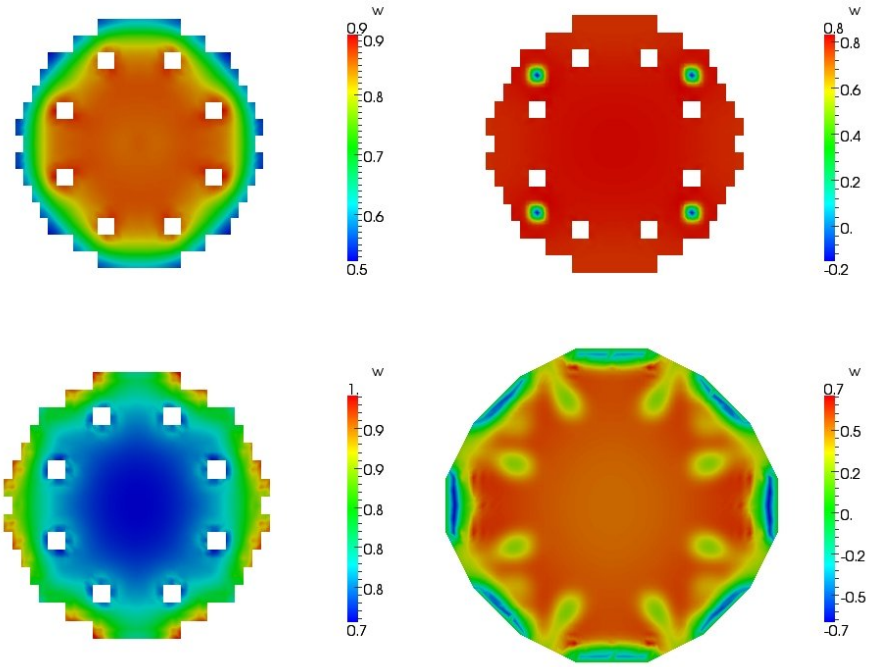

Figure 4.11: Case A1 (open assembly).Cases Distributions of the z-component of the velocity $\boldsymbol{v}^*$ at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

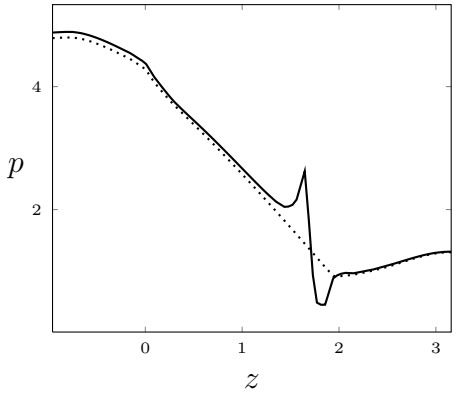

Figure 4.12: Case A1 (open assembly) Pressure along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$(solid line) and along the same line in the absence of blockage (dotted line)

profile in the pressure along the center of the assembly where there is no pressure drop due to zero velocity of the fluid. With the exception of the

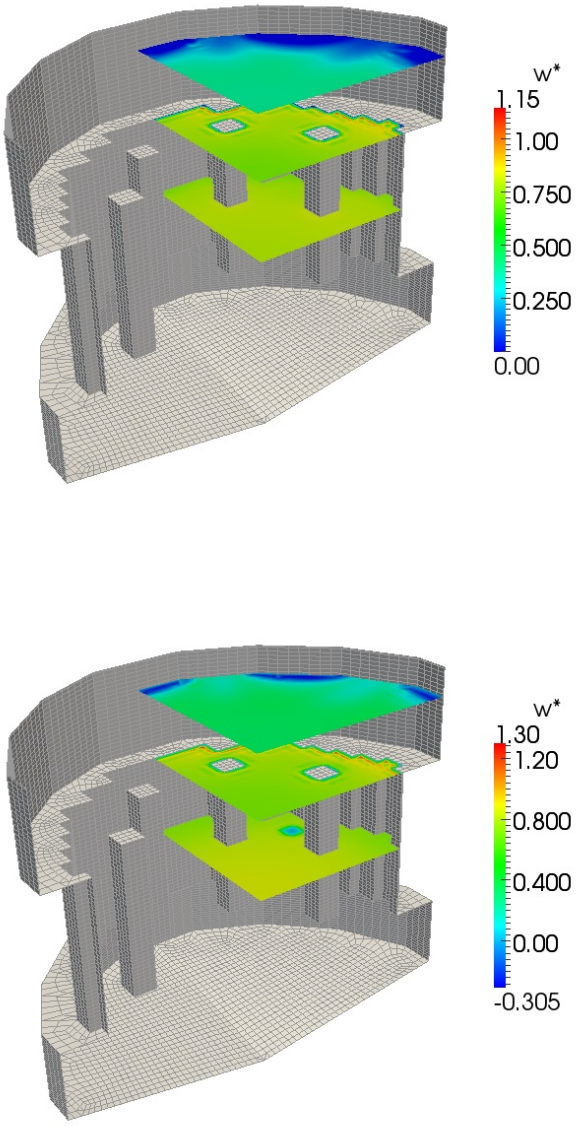

Figure 4.13: Case A1 (open assembly). Temperature field with blockage (left) and in the absence of blockage (right)

blockage region the solid and the dotted lines match closely. Large peaks and fast oscillations of the pressure close to the boundary of the blockage may be considered numerical errors which disappear if one uses discontinuous elements for the pressure. We recall that we are using Taylor-Hood finite elements which consist of piecewise-continuous quadratic polynomials for velocity and piecewise-continuous linear polynomials for pressure.

4.4 Outlet open core blockage test (A2)

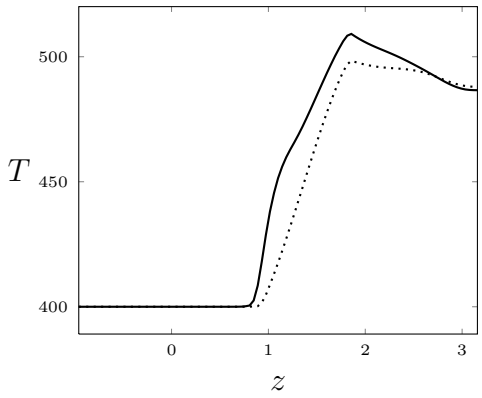

Figure 4.14: Case A2 (open assembly). Temperature along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$(solid line) and temperature along the same line in the absence of blockage (dotted line)

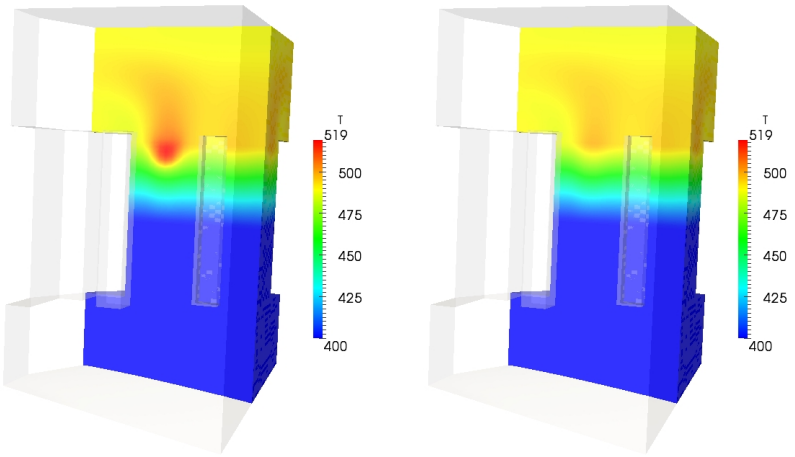

Figure 4.15: Case A2. Temperature field with blockage (left) and in the absence of blockage (right)

In this case we assume again that the assemblies are open, so that a flow exchange between them is permitted. The blockage is set at the end of the assembly with center $(1.323, 1.176)$. Differently from the previous case, in the first part of the core the coolant can flow freely but it is blocked at the end of the assembly. The flow cannot exit the assembly in a standard way and therefore a temperature increase is expected. In Figure 4.14 the temperature

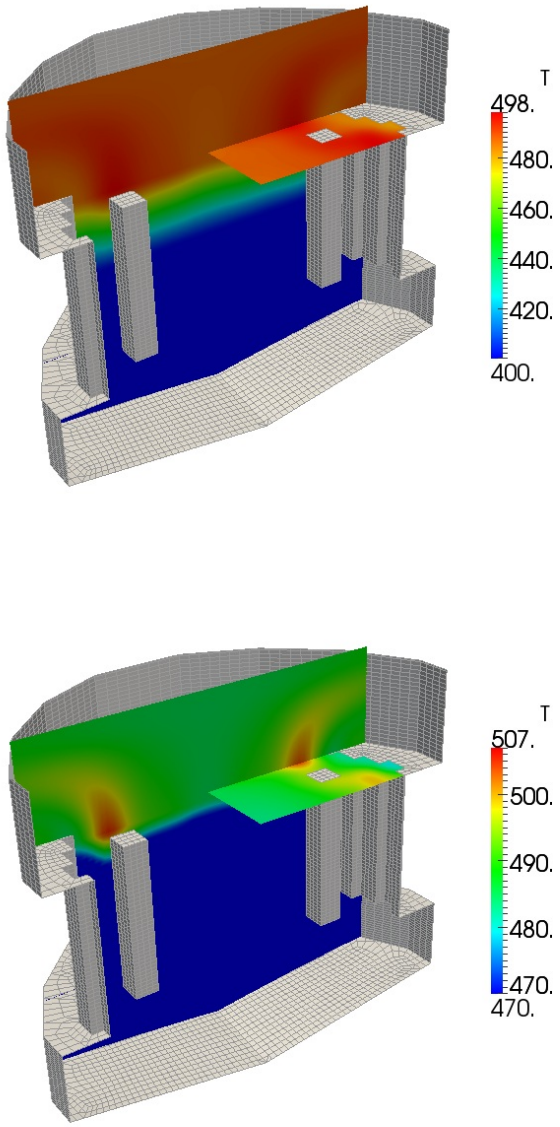

Figure 4.16: Case A2 (open assembly). Temperature field with blockage (left) and in the absence of blockage (right)

profile along the z-line at the center of the blocked assembly $x = 1.323$ and $y = 1.176$ is shown in comparison with the normal condition of non blocked assembly. The temperature of the blocked assembly case at the top core is

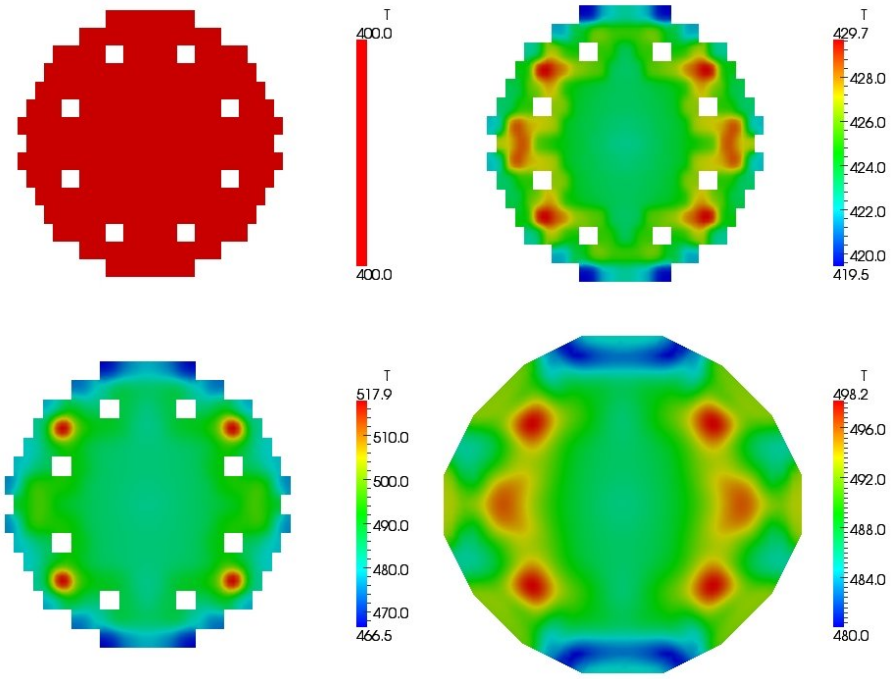

Figure 4.17: Case A2 (open assembly). Temperature distributions at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

slightly higher than the case of absence of blockage. The blockage case is in solid line and the latter in dotted line. In Figures 4.15-4.16 there is an overview of the temperature distribution T over the section of the reactor. In Figure 4.15 the overview of the temperature distribution T is over the section of the reactor that includes the blockage assembly. The temperature field with blockage is shown on the left and the temperature field in the absence of blockage on the right, with the same temperature scale. In this case the increase of temperature diffuses all around the blocked assembly creating a hot spot. The same observation can be drawn from Figure 4.16 where the temperature field over the reactor is shown.

In Figures 4.17-4.18 the temperature distributions are shown over different planes. In Figure 4.17 we report the temperature distributions over the planes $z = 0.6, z = 1.2, z = 1.8$ and $z = 2.4$. These heights correspond to the lower core, upper core inlet, upper core outlet and upper plenum region respectively. One can see that the temperature is slightly higher in correspondence of the blocked cross sections, where the convection heat exchange is absent as a consequence of the zero velocity. In Figure 4.18 on the top and

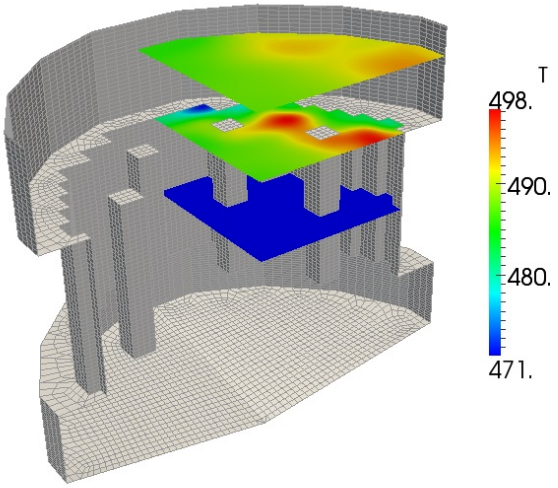

Figure 4.18: Case A2 (open assembly). Overall temperature field with blockage (left) and in the absence of blockage (right)

on the bottom there is a reactor overview of various temperature sections

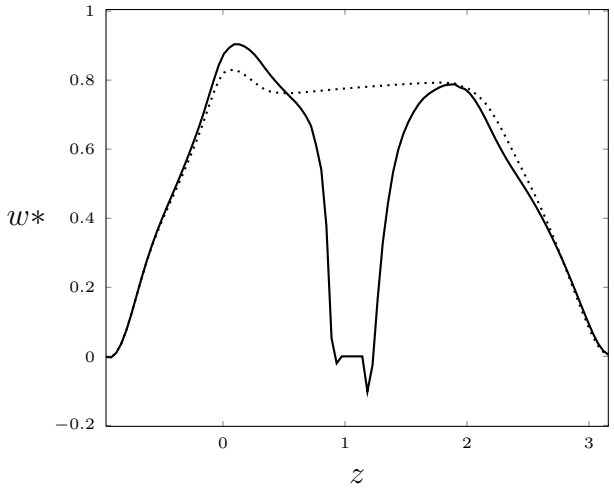

Figure 4.19: Case A2 (open assembly). z-component of the velocity field w^* along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ (solid line) and along the same line in the absence of blockage (dotted line)

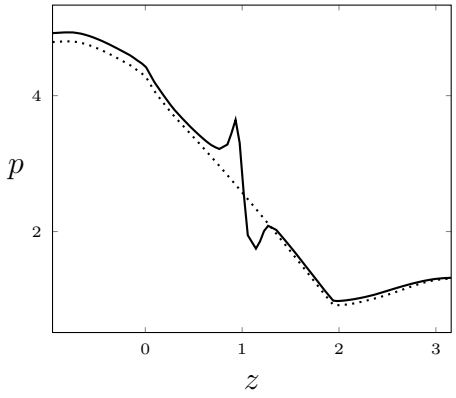

Figure 4.20: Case A2 (open assembly). Pressure along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ (solid line) and along the same line in the absence of blockage (dotted line)

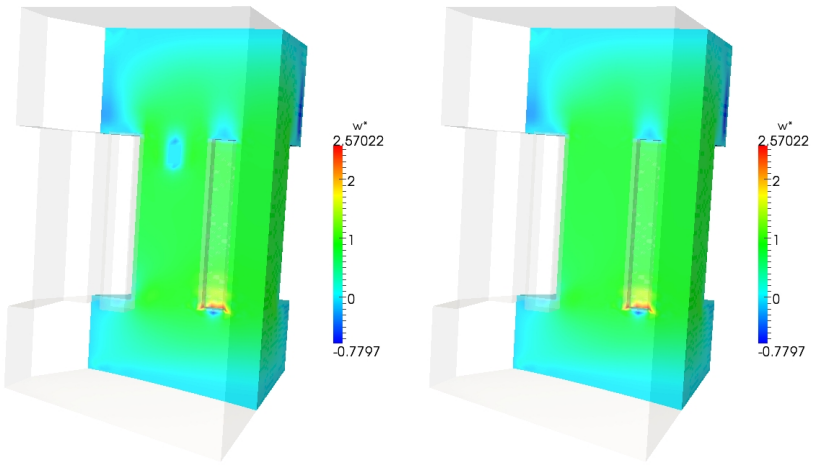

Figure 4.21: Case A2 (open assembly). Velocity w^* field with blockage (left) and in the absence of blockage (right)

with and without blockage.

In Figure 4.20 the z-component of the velocity field w^*, defined in (4.1), along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ is shown. The velocity in the the blocked assembly is defined with solid line and with dotted line in the absence of blockage. In a similar way in Figure 4.20 pressure along the same z-line in the blocked assembly is shown with solid line and dotted line in the absence of blockage. We can clearly see that the blockage imposes a discontinuous profile in the pressure along the center of the assembly where there is no pressure drop due to zero velocity of the fluid.

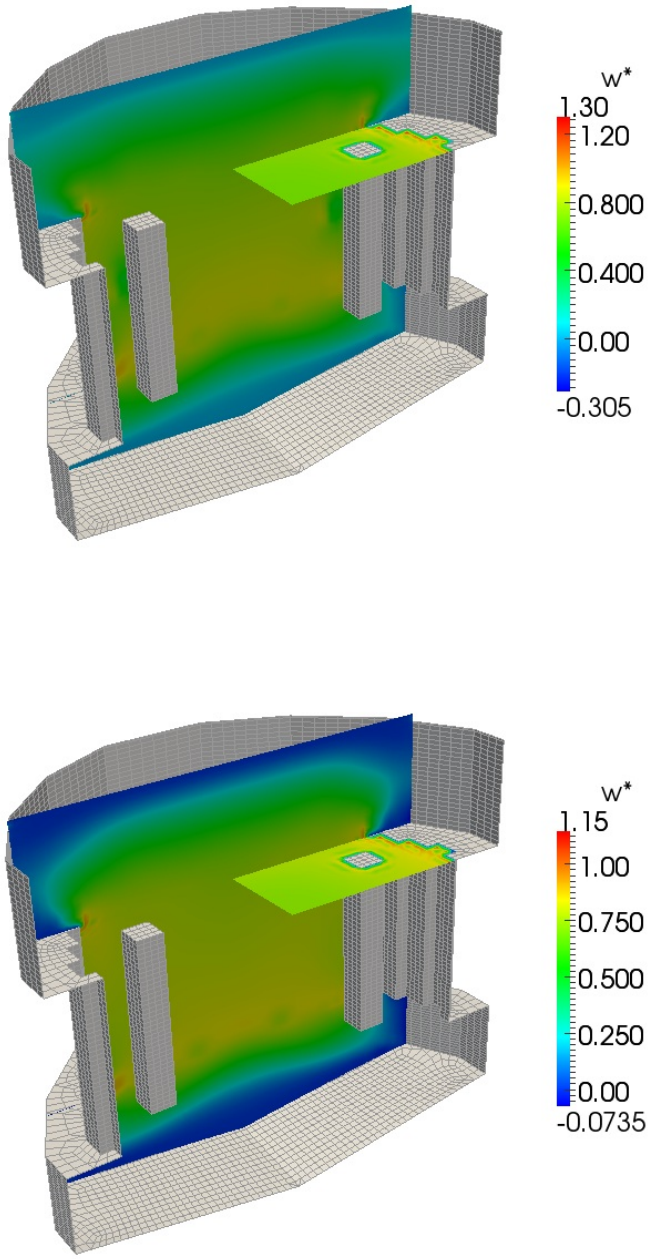

Figure 4.22: Case A2 (open assembly). Velocity field with blockage (top) and in the absence of blockage (bottom)

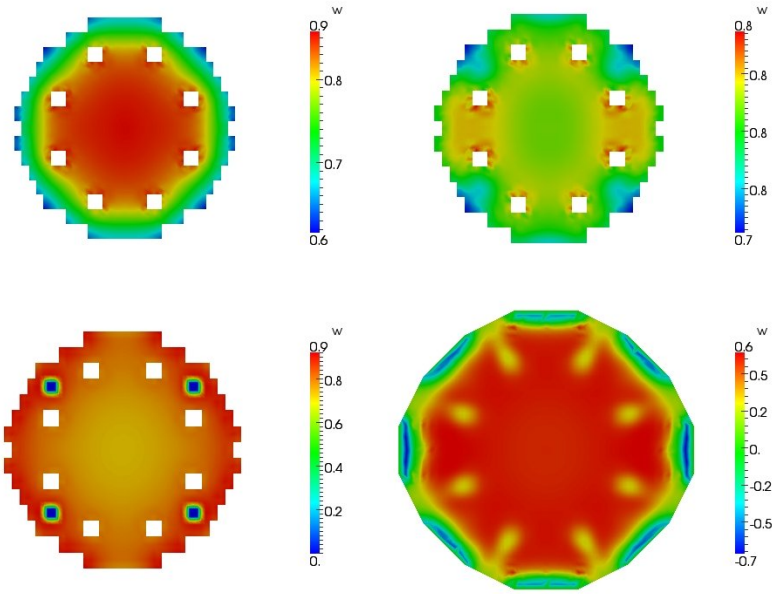

Figure 4.23: Case A2. Distributions of the z-component of the velocity w^* at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

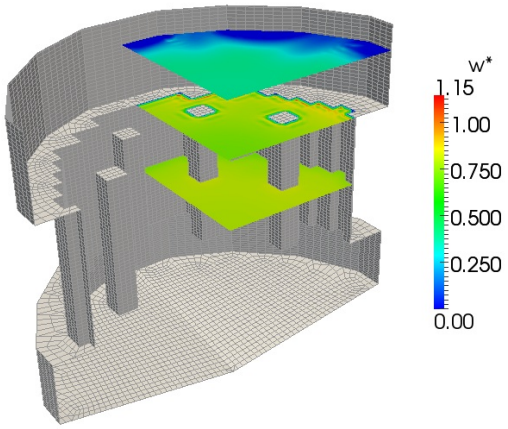

Figure 4.24: Case A2 (open assembly). Velocity field with blockage (left) and in the absence of blockage (right)

The velocity field is shown in Figures 4.21-4.24. In Figure 4.21 the overview of the z-component of the extended velocity field w^* is shown over the section of the reactor that includes the blockage assembly. The velocity field with blockage is shown on the left and the velocity field in the absence of blockage on the right. In this case one can see very well the blockage area where the coolant velocity vanishes. The velocity field is shown over the whole reactor in Figure 4.22. The hot spot cannot be seen since it is behind a main control assembly. In Figure 4.23 we show the distributions of the z-component of the velocity w^* at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum). One can clearly see the blockage only in the plane with $z = 1.8$ where it is located. The overall velocity field over different planes at different heights with blockage can be seen in Figure 4.24.

4.5 Closed core blockage test (B)

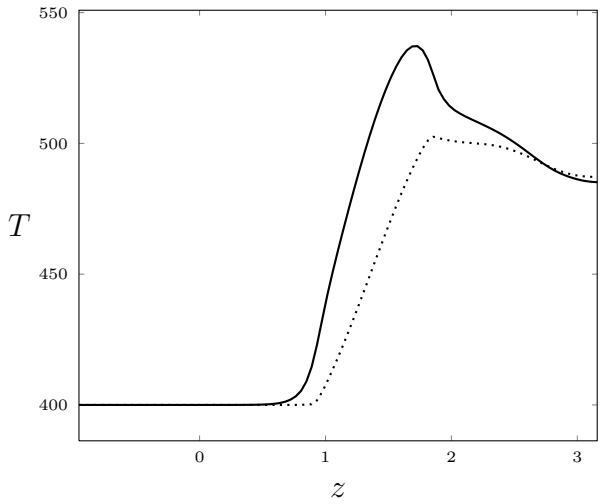

Figure 4.25: Case B (closed assembly). Temperature along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$(solid line) and temperature along the same line in the absence of blockage (dotted line)

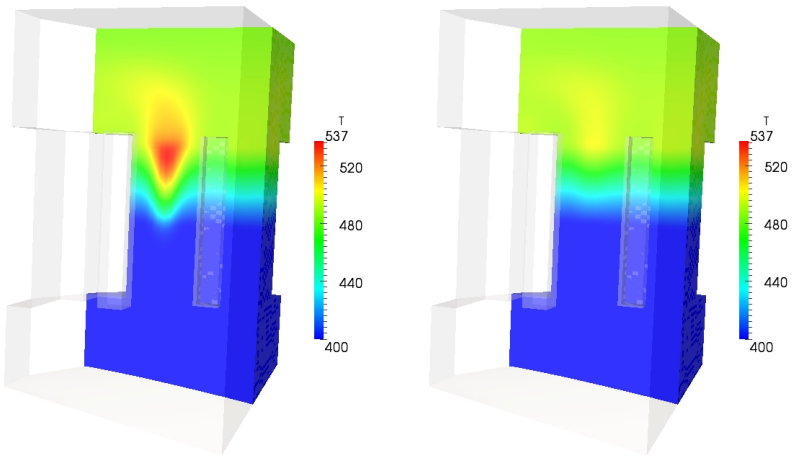

Figure 4.26: Case B (closed assembly). Temperature field with blockage (left) and in the absence of blockage (right)

In this case we assume that the assemblies are closed, so that there is not a flow exchange between them with no transversal velocities. The blockage is set at the beginning of the assembly with center $(1.323, 1.176)$. The flow

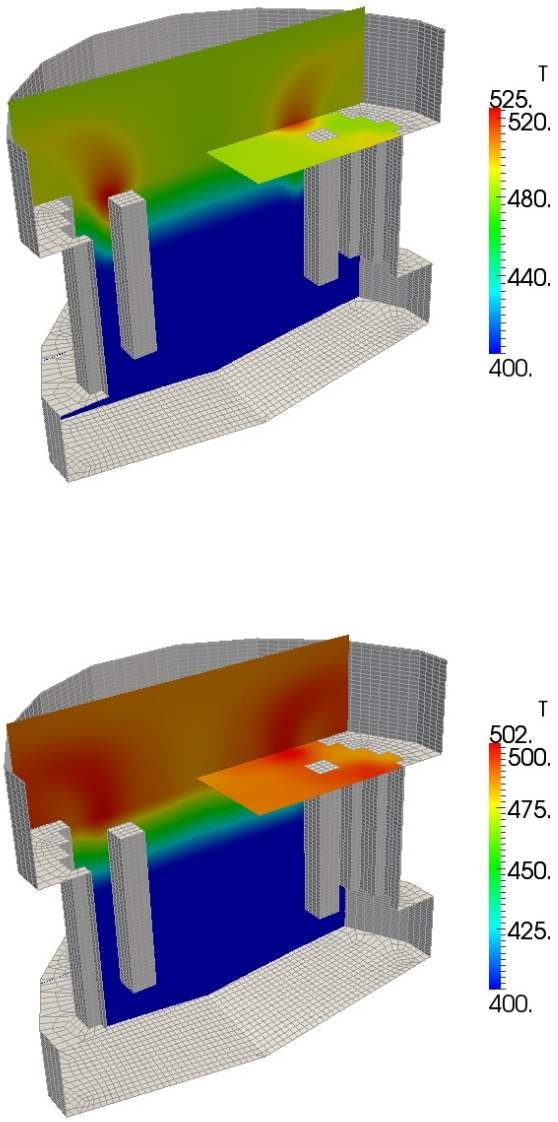

Figure 4.27: Case B (closed assembly). Temperature field with blockage (top) and in the absence of blockage (bottom)

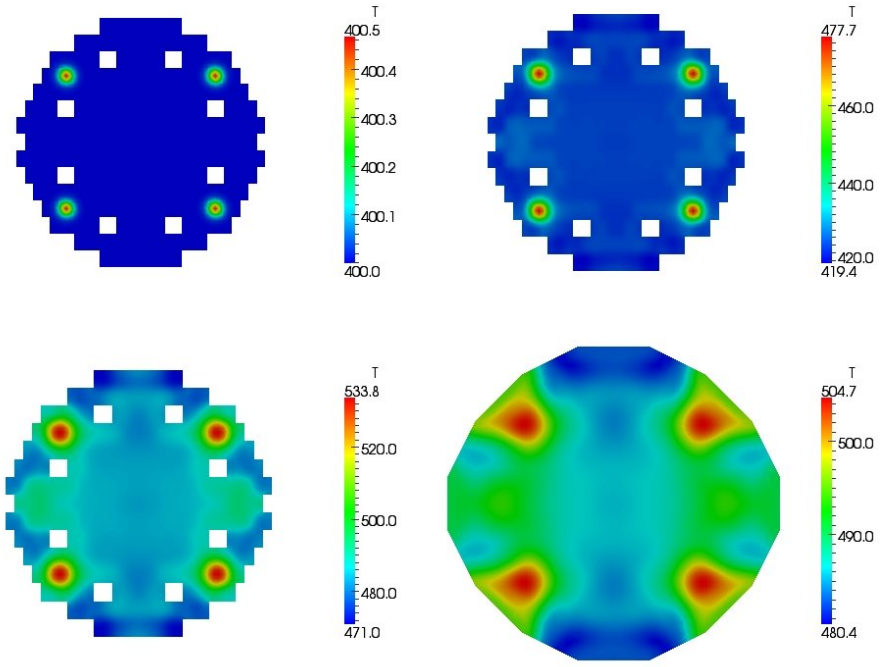

Figure 4.28: Case B (closed assembly). Temperature distributions at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

cannot go through the assembly surface and therefore an increase of temperature is expected. In Figure 4.25 the temperature profile along the z-line at the center of the blocked assembly $x = 1.323$ and $y = 1.176$ is shown with solid line, in comparison with the normal condition of non blocked assembly (dotted line). It is evident that the temperature peak is higher in the case of blocked assembly.

In Figures 4.26-4.27 there is an overview of the temperature distribution T over the section of the reactor. In Figure 4.26 the overview of the temperature distribution T is over the section of the reactor that includes the blockage assembly. The temperature field with blockage is shown on the left and the temperature field in the absence of blockage on the right. A large hot spot around the blocked assembly is created. In Figure 4.27 the overview is more general. In these figures the temperature scale is different in order to see the global effect. In this case one can see that the large hot spot of Figure 4.26 is a local one and the extra heat generated in the blocked assembly is cooled quickly by the neighbouring assemblies.

In Figures 4.28-4.29 the temperature distributions are shown over differ-

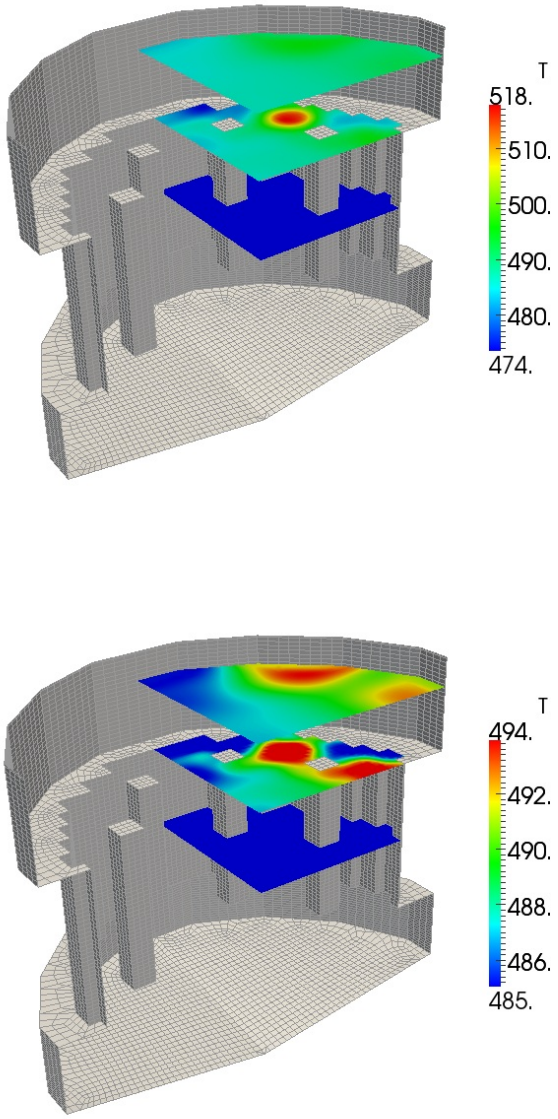

Figure 4.29: Case B (closed assembly). Overview of temperature sections with blockage (left) and in the absence of blockage (right)

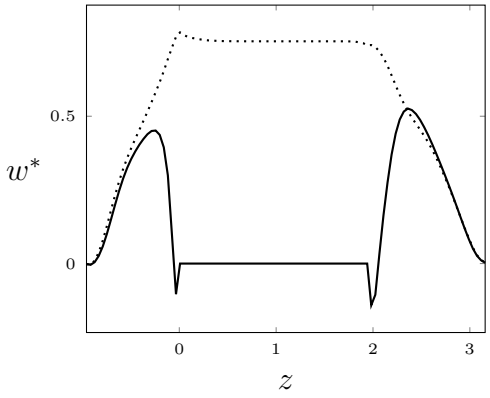

Figure 4.30: Case B (closed assembly). z-component of the velocity field w^* along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ (solid line) and along the same line in the absence of blockage (dotted line)

ent planes. In Figure 4.28 we report the temperature distributions over the planes $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet) and $z = 2.4$ (upper plenum). One can easily see that the temperature

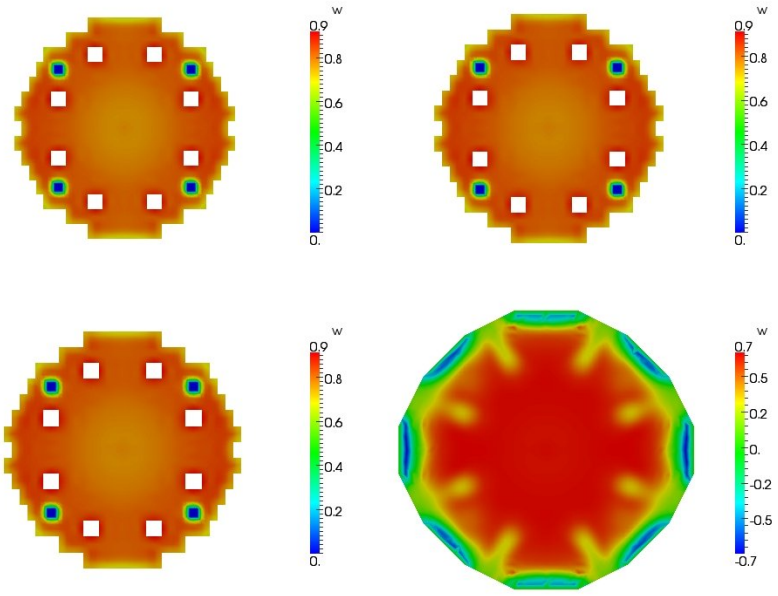

Figure 4.31: Case B (closed assembly). Distributions of the z-component of the velocity $\boldsymbol{v}^*$ at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum)

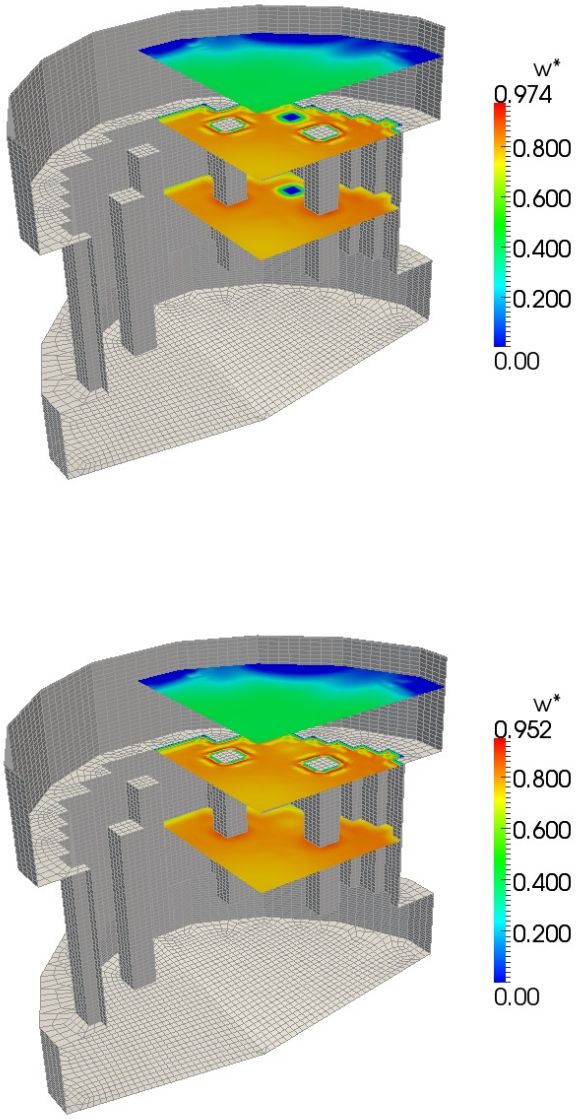

Figure 4.32: Case B (closed assembly). Overall of the velocity field with blockage (top) and in the absence of blockage (bottom)

is higher in correspondence of the blocked cross sections, where the convection heat exchange is absent as a consequence of the vanishing velocity. In Figure 4.29 on the top and on the bottom there is a reactor overview of temperature maps at various sections with and without blockage, respectively.

The velocity field is shown in Figures 4.30-4.32. In Figure 4.30 the z-component of the velocity field w^* along a z-line in the blocked assembly at $x = 1.323$ and $y = 1.176$ is shown. The velocity in the blocked assembly is defined with solid line and with dotted line in the absence of blockage. We recall that w^* is not the real velocity inside the core as defined in (4.1). The distributions of the z-component of the velocity $\boldsymbol{v}^*$ at the sections $z = 0.6$ (lower core), $z = 1.2$ (upper core inlet), $z = 1.8$ (upper core outlet), $z = 2.4$ (upper plenum) are in Figure 4.31. The overall velocity field over different planes at different heights with blockage and in the absence of blockage can be seen in Figure 4.32 on the top and bottom, respectively.

4.6 Closed and open core blockage comparison

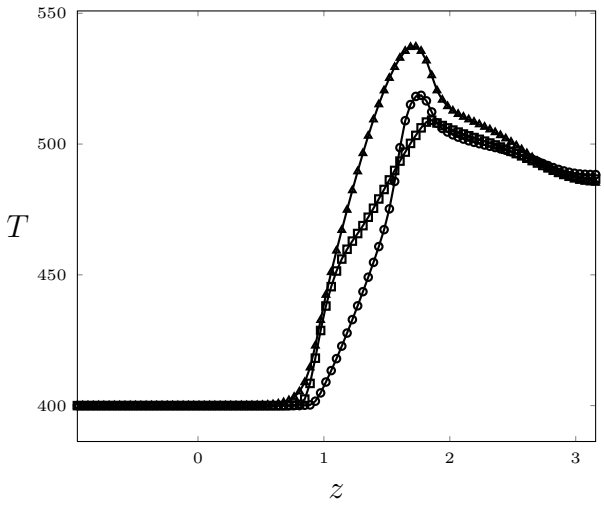

Figure 4.33: Temperature along a z-line at $x = 1.323$ and $y = 1.176$ for Cases $A1$ (circle sign), $A2$ (square sign) and B (triangle sign).

We now report some brief considerations that can be drawn from the comparison of Cases A1, A2 and B. In Figure 4.33 we plot the temperature profiles along a z-line parallel to the longitudinal axis at $x = 1.323$ and $y = 1.176$ for Cases A1 (circle sign), A2 (square sign) and B (triangle sign). One can observe that the highest temperature is attained when the blockage occurs in the closed assembly case. This is clearly due to the fact that the fluid running into the obstacle is not allowed to overcome it by going in the transversal directions. Therefore no convection can contribute to heat exchange in that case.

We also notice that in all the three cases the temperature peak is reached approximately at the end of the core heated zone, right before the entrance in the upper plenum. At that point in fact the LBE fluid has increased its thermal energy by the exchange with the total length of the upper core. Nevertheless, we notice different behaviours in terms of the slope of the curves. In particular, when the blockage takes place at the inlet of the upper core, the fluid undergoes a steeper temperature increase in that region with respect to the other cases. After that increase, the thermal flux diminishes because of a smaller temperature gradient from the fuel rods to the fluid. Hence, the slope of the longitudinal temperature profile is smaller than the other cases.

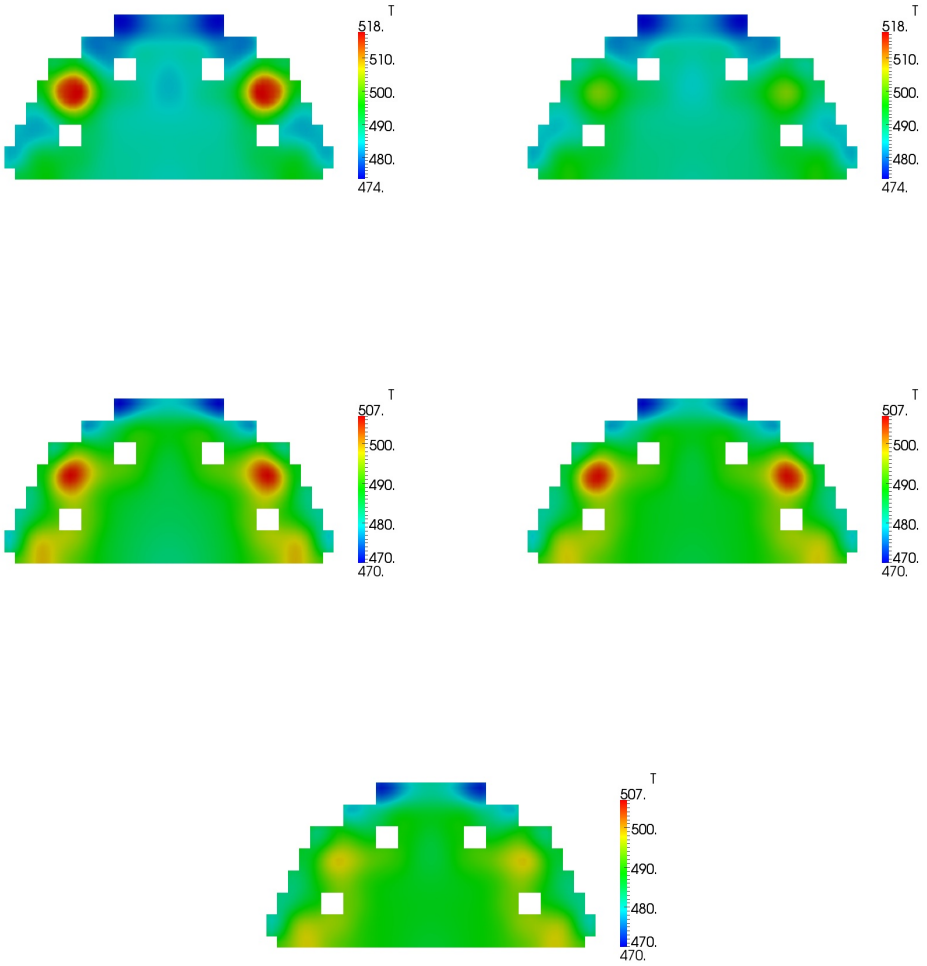

Figure 4.34: Comparison between the temperature section profiles at $z = 1.94$ m in the closed assembly configuration with (top left) and without (top right) blockage and in the open assembly configuration with upper (middle left), lower (middle right) and without (bottom) blockage

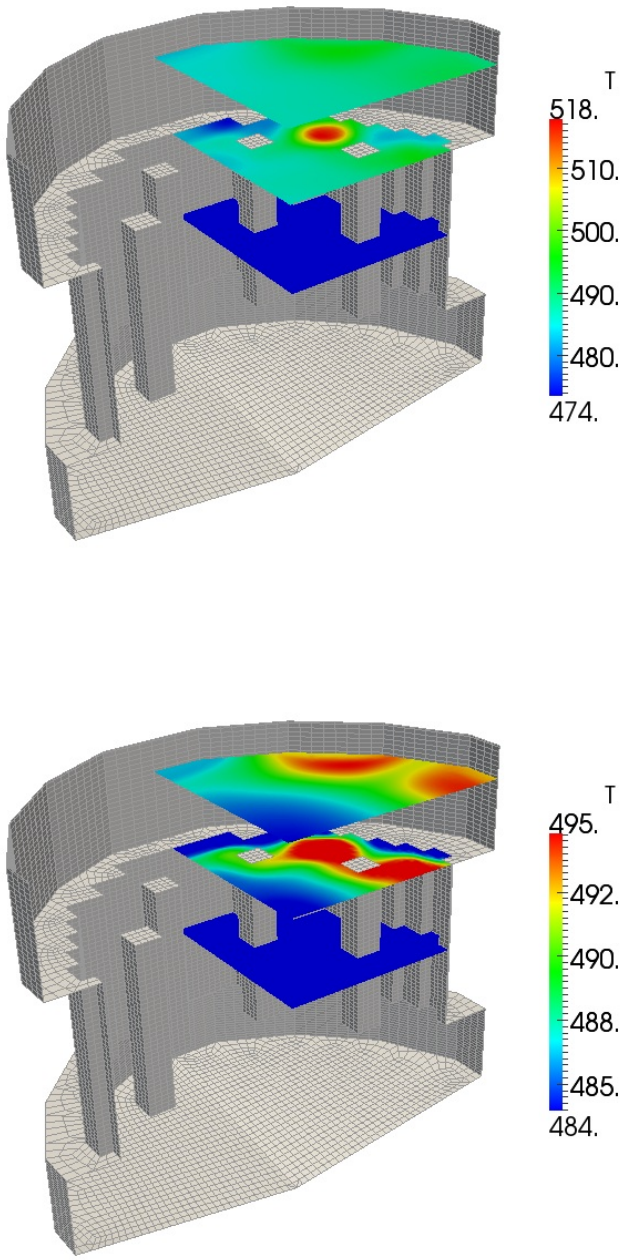

Figure 4.35: Comparison between the temperature profiles in the closed (top) and open (bottom) assembly configuration

In Figures 4.34-4.35 a comparison between the temperature profiles in the closed and open assembly configurations is reported with and without blockage. In particular on the top part of Figure 4.34 one can see the temperature on the section at $z = 1.94$ m in the closed assembly configuration with (left) and without (right) blockage. In the second row of the figure the temperature maps in the open assembly configuration with upper (middle left) and lower (middle right) blockages are shown and on the bottom of the figure the temperature without blockage for open assemblies. In Figure 4.35 a direct comparison between different temperature maps at some sections in the closed (top) and open (bottom) assembly configurations is reported. The temperature peak is clearly visible for the closed assembly configuration where no convection occurs within the entire assembly due to a complete blockage.

Bibliography

[1] F.Bassenghi, G.Bornia, L.Deon and S. Manservisi, *Implementation and validation of the NURISP platform*, Technical report CIRTEN (2011) 81

[2] F.Bassenghi, G.Bornia, A. Cervone and S. Manservisi, *The ENEA-CRESCO platform for simulating liquid metal reactor*, Technical report LIN-THRG 210, (2010) 81

[3] S. Bnà, S. Manservisi and O. Le Bot, *Simulation of the Thermal-hydraulic behaviour of Liquid Metal Reactors using a Three-Dimensional Finite Element Model, Technical Report DIENCA-UNIBO*, 2010. 8, 21, 98

[4] A. Cervone and S. Manservisi, *A three-dimensional CFD program for the simulation of the thermo-hydraulic behaviour of an open core liquid metal reactor*, Technical report lin-thrg 108, (2008) 15, 16, 17, 21, 101

[5] *ELSY Work Program. European Lead-cooled SYstem (ELSY) Project*, Technical report, EURATOM, Management of Radioactive Waste (2006) 97, 98

[6] Fluent Inc., *FLUENT 6.3 User's Guide*, Fluent Incorporated, USA (2006) 52, 54

[7] OECD/NEA, *Handbook on Lead-bismuth Eutectic Alloy and Lead Properties, Materials Compatibility, Thermal-hydraulics and Technologies*, OECD/NEA No. 6195, ISBN 978-92-64-99002-9 (2007) 12

[8] LASPACK (Linear Algebra Sparse Matrix Package): http://www.mgnet.org/mgnet/Codes/laspack/html/laspack.html 26

[9] LIBMESH package: http://libmesh.sourceforge.net/ 83

[10] R. N. Lyon, *Liquid-Metals Handbook*, 2nd ed., Atomic Energy Comm., Washington D.C. (1952) 12

[11] F.R. Menter, *Zonal Two Equation κ-ω Turbulence Models for Aerodynamic Flows*, AIAA Paper. pp.93-2906 (1993) 53, 54, 55, 56

[12] F.R. Menter, *Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications*, AIAA Journal, vol. 32, no 8. pp. 1598-1605 (1994) 53, 54, 55, 56

[13] V.I. Mikhin, Second-order κ-ϵ turbulence model, The VI International Congress on Mathematical Modeling, Nizhny Novgorod, Russia (2004) 61, 62, 64, 65

[14] V.I Mikhin, *Two κ-ϵ turbulence models (first & second order respectively), not containing model functions*, ENEA-RT-FIS; 05-50, ISSN 0393-3016, ENEA, Italy (2005) 61

[15] V.I Mikhin, Second-order κ - ϵ turbulence model not containing model functions, Proceedings of the Fourth Russian National Conference on Heat Exchange, 2006, Moscow. Vol. 2, pp.202-205. 61

[16] V.I Mikhin, *Turbulence models κ - ϵ of first and second orders. Test results*, Proceedings of the Fourth Russian National Conference on Heat Exchange, Vol. 2, pp.206-209, Moscow (2006) 61, 62, 64, 65

[17] V.I Mikhin, Voukelatou K, *Four-parametric κ - ϵ-κ_θ - ϵ_θ turbulence model for compressible fluids*, ENEA-RT-FPN; ISSN 0393-3016, ENEA, Italy (2007) 62, 79

[18] MPI (Message Passing Interface) forum, http://www.mpi-forum.org/ 81

[19] OpenMPI library, http://www.open-mpi.org/ 82

[20] PARAVIEW visualization software, http://www.paraview.org 37

[21] PETSc (Portable Extension Toolkit for Scientific Computation), http://www.mcs.anl.gov/petsc/petsc-as/ 82

[22] A. Pirovano, S. Viannay and M. Jannot, *Convection naturelle en regime turbulent le long d'une plaque plane verticale*, Proc. 9th Int. Heat Transfer Conf., Natural Convection, Vol.4, NC 1.8, pp. 1-12. Elsevier, Amsterdam (1970) 68

[23] CEA/DEN, *SALOME Documentation*, CEA/DEN, EDF R&D, OPEN CASCADE, (2007-2008) 30

[24] Sato H., Shimada M., Nagano Y, *A two-equation turbulence model for predicting heat transfer in various Prandtl number fluids*, Proceedings of the Tenth International Heat Transfer Conference, Brighton, UK, Vol.2, p.443-448 (1994) 61, 62

[25] H. Schlichting , K. Gersten, *Boundary-Layer Theory*, 8th Revised and Enlarged Edition, Springer (2000) 63

[26] T. Tsuji, Y. Nagano, *Turbulence measurements in a natural convection boundary layer along a vertical flat plate*, Int. J. Heat Mass Transfer. Vol. 31, n. 10, pp. 2101-2111 (1988) 77, 79, 135

[27] T. Tsuji and Y. Nagano, *Characteristics of a turbulent natural convection boundary layer along a vertical flat plate*, Int. J. Heat Mass Transfer. Vol. 31, n. 8, pp. 1723-1734 (1988) 68, 72, 73, 74, 75, 76, 78, 135

[28] *VTK library*, http://www.vtk.org 37

[29] D.C. Wilcox, *Re-assessment of the scale-determining equation for advanced turbulence models*, AIAA Journal, vol. 26, no. 11, pp. 1299-1310 (1988) 50

[30] D.C. Wilcox, *Turbulence Modeling for CFD*, ISBN 1-928729-10-X, 2nd Ed., DCW Industries Inc. (2004) 51, 52

List of Figures

List of Tables