Agenzia Nazionale per le Nuove Tecnologie,
l'Energia e lo Sviluppo Economico Sostenibile

Ministero dello Sviluppo Economico

RICERCA DI SISTEMA ELETTRICO

Simulation of the Thermo-Hydraulic Behaviour of Liquid Metal Reactors Using a Three Dimensional Finite Element Model

Doc. CIRTEN RL 1301/2010

S. Bnà, S. Manservisi, O. Le Bot

Report RdS/2010/107

SIMULATION OF THE THERMO-HYDRAULIC BEHAVIOUR OF LIQUID METAL REACTORS USING A THREE-DIMENSIONAL FINITE ELEMENT MODEL

S. Bnà, S. Manservisi, O. Le Bot (Università di Bologna)

Settembre 2010

Report Ricerca di Sistema Elettrico
Accordo di Programma Ministero dello Sviluppo Economico – ENEA
Area: Produzione e fonti energetiche
Tema: Nuovo Nucleare da Fissione

Responsabile Tema: Stefano Monti, ENEA

CIRTEN

CONSORZIO INTERUNIVERSITARIO

PER LA RICERCA TECNOLOGICA NUCLEARE

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

DIPARTIMENTO DI INGEGNERIA ENERGETICA,
NUCLEARE E DEL CONTROLLO AMBIENTALE
LABORATORIO DI MONTECUCCOLINO

SIMULATION OF THE THERMO-HYDRAULIC BEHAVIOUR OF LIQUID METAL REACTORS USING A THREE-DIMENSIONAL FINITE ELEMENT MODEL

CIRTEN-UNIBO RL 1301/2010

AUTORI

S. Bnà, S. Manservisi, O. Le Bot

PISA, LUGLIO 2010

Lavoro svolto in esecuzione della linea progettuale LP3 punto F1 - AdP ENEA MSE del 21/06/07
Tema 5.2.5.8 – "Nuovo Nucleare da Fissione".

Nuclear Engineering Laboratory of Montecuccolino
DIENCA - UNIVERSITY OF BOLOGNA
Via dei Colli 16, 40136 Bologna, Italy

SIMULATION OF THE THERMO-HYDRAULIC BEHAVIOUR OF LIQUID METAL REACTORS USING A THREE-DIMENSIONAL FINITE ELEMENT MODEL

September 1 2010

Authors: S. Bnà, S. Manservisi and O. Le Bot
sandro.manservisi@unibo.it

Abstract. A thermo-fluid model for a liquid metal reactor has been implemented on a finite element code with the purpose of investigating three-dimensional profiles of pressure, velocity and temperature fields. All the sub-channel details in the core are summarized in parametric coefficients. The core fields are coupled with the upper and lower plenum where the standard Navier-Stokes system with turbulence model equations is solved. Some preliminary tests for generic geometries with open and closed assembly model are reported.

Contents

Introduction

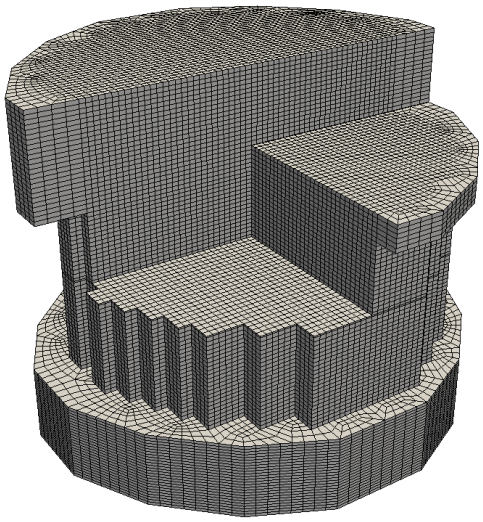

Figure 1: Computational three-dimensional reactor model

In this report a full 3D CFD code with the purpose of analyzing the thermal hydraulic behaviour of a liquid metal reactor core has been developed. The purpose of this code is to investigate three-dimensional pressure, velocity and temperature fields inside nuclear reactors at the coarse fuel assembly level. Due to the complexity of the geometry, approximate CFD models have been developed for the core region, for the upper and lower plenum and for sub-assembly computations.

Chapter 1 introduces the mathematical model for the core and the upper/lower plenum region. The reactor upper and lower plenum region model is introduced by coupling the Navier-Stokes and energy system with a turbulence model. The $\kappa - \epsilon$, $\kappa - \omega$ and LES turbulence models have been implemented in the finite element code. For the core region a two-level finite element Navier-Stokes model is proposed. Boundary, initial and yielding conditions are illustrated.

In Chapter 2 a CFD analysis is proposed for sub-assembly heat exchange investigation. Experimental results from a single fuel rod are reproduced with CFD tools and the computation of the heat exchange coefficient is presented. This parameter may be used for temperature computations inside the fuel pins starting from the average coolant field computed by the code.

Chapter 3 is a brief User Guide for the finite element code which is provided with this report. The User Guide is divided into three steps: code pre-processing, code running and code post-processing. A brief description of the code structure and main compilation commands is reported to allow a user to reproduce the results and manage the basic commands. A description of the mesh tools available to design the reactor geometry, to configure the solver class for Navier-Stokes, energy and turbulence models are included in the User Guide. The distributions of the core power factors and pressure losses are introduced by using external files which can be generated automatically by appropriate tools. We discuss how to view the

results by using the PARAVIEW open source software in output files with HDF5 and XDMF formats.

Chapter 4 is devoted to CFD computations. In this chapter some basic tests are performed for different core geometries in order to compare this three-dimensional approach with the standard mono-dimensional one. We use two different reactor geometries: the first geometry does not include the control rod area which is included in the other geometry. This code has been used, under the assumption of weakly correlated assemblies, for a preliminary assessment of an open square lattice with three fuel radial zones at different levels of enrichment. With the first geometry we study four cases of open and closed assembly models with and without the control assembly region.

Chapter 1

CFD reactor modeling

1.1 Reactor upper and lower plenum region model

The purpose of the code discussed in this report is to compute the velocity, pressure and temperature distributions in different regions of the reactor. In 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 tools. In this section, we briefly present the equations implemented in the code that are available in the modeling of the lower and upper plenum. In this region we can solve the three-dimensional equations of conservation of mass, momentum and energy equation coupled with turbulence models. In particular we may use turbulent models such as $\kappa-\epsilon$, $\kappa-\omega$ and LES. Briefly in the next sections we summarize the equation solvers in the finite element implementations. The parameters of the different turbulence models and of the Navier-Stokes system and energy equation can be controlled through appropriate configuration variables as discussed in Chapter 3.

1.1.1 Navier-Stokes system and its finite element approximation

Navier-Stokes system

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

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\, \mathbf{v}) = 0\,, \tag{1.1}$$

b) **Momentum equation**

$$\frac{\partial \rho\, \mathbf{v}}{\partial t} + \nabla \cdot (\rho \overline{\mathbf{v}\mathbf{v}}) = -\nabla p + \nabla \cdot \bar{\tau} + \rho \mathbf{g}\,, \tag{1.2}$$

c) **Energy equation**

$$\frac{\partial \rho e_0}{\partial t} + \nabla \cdot (\rho \mathbf{v} e_0) = \Phi + \nabla \cdot \mathbf{q} + \dot{Q}\,. \tag{1.3}$$

For our purposes the system can be considered incompressible, while the density is assumed only to be slightly variable as a function of the temperature, with $\rho = \rho(T)$ given. The tensor

$\bar{\tau}$ is defined by

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

with $i, j = x, y, z$. In a similar way the tensor $\overline{\mathbf{vv}}$ is defined as $\overline{\mathbf{vv}}_{ij} = v_i\, v_j$. 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 number are defined by

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

In order to close these equations it is also necessary to specify some state equation. We assume

$$\rho = a + \gamma T, \quad e_0 \equiv C_v T + \frac{\mathbf{v}^2}{2}\,, \tag{1.7}$$

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 k the heat conductivity. $\dot{Q}$ is the volume heat source and Φ the dissipative heat term. Equations are supplemented with the constant data and appropriate boundary conditions and they form a closed set of partial differential equations.

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. In the following, we refer to "filtering" as the convolution of a function with a filtering kernel G

$$\mathbf{v}(\vec{x}) = \int G(\vec{x} - \vec{\xi})\mathbf{v}(\vec{\xi})d\vec{\xi}, \tag{1.8}$$

resulting in

$$\mathbf{v} = \bar{\mathbf{v}} + \bar{\mathbf{v}}'\,, \tag{1.9}$$

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 field $\bar{\mathbf{v}}$ and $\bar{p}$ as

$$\frac{\partial\rho\,\bar{\mathbf{v}}}{\partial t} + \nabla\cdot(\rho\overline{\bar{\mathbf{v}}\bar{\mathbf{v}}}) = -\nabla\bar{p} + \nabla\cdot\bar{\tau} + \rho\bar{\mathbf{g}}\,. \tag{1.10}$$

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, due to the fact that

$$\overline{v_j\frac{\partial v_i}{\partial x_j}} \neq \bar{v}_j\frac{\partial\bar{v}_i}{\partial x_j} \tag{1.11}$$

and hence

$$\tau_{ij} = \bar{v}_i\bar{v}_j - \overline{v_i v_j}\,. \tag{1.12}$$

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

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\mu_t \bar{S}_{ij}\,, \tag{1.13}$$

where $\bar{S}_{ij}$ is the rate-of-strain tensor for the resolved scale defined by

$$\bar{S} = \frac{1}{2}\left(\nabla \mathbf{v} + \nabla \mathbf{v}^T\right) \tag{1.14}$$

and μ_t is the subgrid-scale turbulent viscosity. Substituting into the filtered Navier-Stokes equations, we then have

$$\frac{\partial \rho\,\mathbf{v}}{\partial t} + \nabla \cdot (\rho \overline{\mathbf{v}\mathbf{v}}) = -\nabla p + \nabla \cdot [(\nu + \nu_t)\nabla \mathbf{v}] + \rho \mathbf{g}\,, \tag{1.15}$$

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 the paper 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

$$\bar{\tau} = 2(\mu + \rho\nu_t)\bar{D}(\bar{u})\,, \tag{1.16}$$

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

$$\mathbf{q} =\equiv -C_p\left(\frac{\mu}{Pr} + \frac{\rho\nu_t}{Pr_t}\right)\nabla T\,. \tag{1.17}$$

The functions ν_t and Pr_t are called turbulent viscosity and turbulent Prandtl numbers and must be computed by solving other transport equations. For many fluids Pr_t is assumed to be constant.

In order to complete the system (1.1-1.3) we need to specify the physical constants. We focused only on liquid lead as coolant. It is well known that lead-bismuth eutectic (LBE) is sometimes preferred between lead and bismuth because of its better properties like cross section, radiation damage, activation and in particular because of the fact that it has a lower melting point and it is liquid in a wider range of temperatures, which is an obvious advantage for heat removal and safety. On the other hand, lead cooled fast reactors with nitride fuel assemblies are currently being studied in the world because of the lower price of the coolant. It is also to be said that, from the point of view of density and viscosity, which are our concern presently, there is no remarkable difference between lead and lead-bismuth eutectic [4]. Below we report the state equations that are implemented in the code for lead liquid metal coolant. These state equations can be modified easily in the configuration files of the code (see Chapter 3).

Density The lead density is assumed to be a function of temperature as

$$\rho = (11367 - 1.1944 \times T)\,\frac{Kg}{m^3} \tag{1.18}$$

for lead in the range $600K < T < 1700K$.

Dynamic Viscosity The following correlation has been used for the viscosity μ

$$\mu = 4.55 \times 10^{-04}\, e^{(1069/T)}\, Pa \cdot s\,, \tag{1.19}$$

for lead in the range $600K < T < 1500K$.

Thermal Expansion For the mean coefficient of thermal expansion (AISI 316L) we assume

$$\alpha_v = 14.77 \times 10^{-6} + 12.20^{-9}(T - 273.16) + 12.18^{-12}(T - 273.16)^2\, \frac{m^3}{K}\,. \tag{1.20}$$

Thermal conductivity The lead thermal conductivity κ is

$$\kappa = 15.8 + 108 \times 10^{-4}\,(T - 600.4)\, \frac{W}{m \cdot K}\,. \tag{1.21}$$

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

$$C_p = 147.3\, \frac{J}{Kg \cdot K}\,. \tag{1.22}$$

Variational form of the Navier-Stokes equations

Now we consider the variational form of the Navier-Stokes system. In the rest of the paper we denote the spaces of all possible solutions in pressure, velocity and temperature with $P(\Omega)$, $\mathbf{V}(\Omega)$ and $H(\Omega)$, respectively.

a) **Incompressibility constraint.** By multiplying (1.1) by a scalar test function ψ in the space $P(\Omega)$ and integrating over the domain Ω we have the following variational form of the incompressibility constraint

$$\int_\Omega \left(\psi\, \frac{\partial \rho}{\partial t} + \psi\, \nabla \cdot\, (\rho\, \mathbf{v}) \right) d\mathbf{x} = 0 \quad \forall \psi \in P(\Omega)\,. \tag{1.23}$$

b) **Momentum equation.** If one multiplies (1.2) for the three-dimensional test vector function ϕ in the space $\mathbf{V}(\Omega)$ and integrates over the domain Ω one has, after integration by parts, the following variational momentum equation

$$\begin{aligned} &\int_\Omega \frac{\partial\, \rho\, \mathbf{v}}{\partial t} \cdot \phi\, d\mathbf{x} + \int_\Omega (\nabla \cdot\, \rho \overline{\mathbf{v}\mathbf{v}}) \cdot \phi\, d\mathbf{x} = \\ &\qquad \int_\Omega p\, \nabla \cdot \phi\, d\mathbf{x} - \int_\Omega \bar{\tau} : \overline{\nabla \phi}\, d\mathbf{x} + \int_\Omega \rho \mathbf{g} \cdot \phi\, d\mathbf{x} - \qquad \forall \phi \in \mathbf{V}(\mathbf{\Omega}) \\ &\qquad \int_\Gamma (p\vec{n} - \bar{\tau} \cdot \vec{n}) \cdot \phi\, d\mathbf{s}\,. \end{aligned} \tag{1.24}$$

The surface integrals must be computed by using the boundary conditions and they are zero if appropriate boundary conditions are imposed. We remark that if we set $\phi = \delta\mathbf{v}$ where $\delta\mathbf{v}$ is a variation of the velocity field $\mathbf{v}$ then (1.24) is the evolution equation for the rate of virtual work.

c) **Energy equation.** Finally for the energy equation, if we multiply for the scalar test function φ in the space $H(\Omega)$ and integrate over the domain Ω we have, after integration by parts, the following variational energy equation

$$\int_\Omega \frac{\partial\, \rho\, C_p\, T}{\partial t}\, \varphi\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\, \mathbf{v} C_p\, T)\, \varphi\, d\mathbf{x} = \int_\Omega \Phi\, \varphi\, d\mathbf{x} - \int_\Omega k\, \nabla T\cdot\nabla\varphi\, d\mathbf{x} + \int_\Omega \dot{Q}\varphi\, d\mathbf{x} + \qquad \forall\varphi\in\mathbf{H}(\Omega) \tag{1.25}$$
$$\int_\Gamma (k\nabla T\cdot\vec{n})\, \varphi\, d\mathbf{s}\,.$$

Again, the surface term must be computed by imposing the appropriate boundary conditions.

Finite element Navier-stokes system

The pressure space $P(\Omega)$, the velocity space $\mathbf{V}(\Omega)$ and the energy space $H(\Omega)$ in (1.23-1.25) 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 domain Ω is always discretized by Lagrangian finite element families with parameter h.

The finite element Navier-Stokes system becomes

a) **FEM incompressibility constraint**

$$\int_\Omega \left(\psi_h \frac{\partial\rho}{\partial t} + \psi_h\, \nabla\cdot\,(\rho\, \mathbf{v}_h)\right) d\mathbf{x} = 0 \quad \forall\psi_h\in P_h(\Omega)\,, \tag{1.26}$$

b) **FEM momentum equation**

$$\int_\Omega \frac{\partial\, \rho\, \mathbf{v}_h}{\partial t}\cdot\phi_h\, d\mathbf{x} + \int_\Omega (\nabla\cdot\rho\,\overline{\mathbf{v}_h\mathbf{v}_h})\cdot\phi_h\, d\mathbf{x} = \int_\Omega p_h\, \nabla\cdot\phi_h\, d\mathbf{x} - \int_\Omega \bar{\tau}_h : \overline{\nabla\phi_h}\, d\mathbf{x} + \int_\Omega \rho\mathbf{g}\cdot\phi_h\, d\mathbf{x} - \qquad \forall\phi_h\in\mathbf{V}_h(\Omega) \tag{1.27}$$
$$\int_\Gamma (p_h\vec{n} - \bar{\tau}_h\cdot\vec{n})\cdot\phi_{\mathbf{h}}\, d\mathbf{s}\,,$$

c) **FEM energy equation**

$$\int_\Omega \frac{\partial\rho\, C_p\, T_h}{\partial t}\, \varphi_h\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\mathbf{v}_h C_p\, T_h)\, \varphi_h\, d\mathbf{x} = \qquad \forall\varphi_h\in H_h(\Omega) \tag{1.28}$$
$$\int_\Omega \Phi_h\, \varphi_h\, d\mathbf{x} - \int_\Omega \mathbf{q}_h\cdot\nabla\varphi_h\, d\mathbf{x} + \int_\Omega \dot{Q}_h\, \varphi_h\, d\mathbf{x} + \int_\Gamma k\ \nabla T_h\cdot\vec{n}\,\varphi_h\, d\mathbf{x}\,.$$

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 (1.26-1.28) yields a system of equations which has one equation for each FEM basis element.

For a Newtonian fluid the viscous stress is given by

$$\tau_{hij} = 2(\mu + \rho\,\nu_t)\,S_{hij}\,, \tag{1.29}$$

where the viscous strain-rate tensor is defined by

$$S_{hij} \equiv \frac{1}{2}\left(\frac{\partial v_{hi}}{\partial x_j} + \frac{\partial v_{hj}}{\partial x_i}\right) - \frac{1}{3}\frac{\partial v_{hk}}{\partial x_k}\delta_{ij}\,. \tag{1.30}$$

The heat flux becomes

$$\mathbf{q}_h = \equiv -C_p(\frac{\mu}{Pr} + \frac{\rho\nu_t}{Pr_t})\nabla T_h\,. \tag{1.31}$$

Finite element class for Navier-stokes and energy equation

The Navier-Stokes solver is implemented in the class `MGSolverNS` which consists of the declaration file `MGSolverNS.h` and implementation files `MGSolverNS.C`, `MGSolverNS3D.C`. The parameters can be set in the file `/config/MGSNSconf.h`. The energy solver is implemented in the class `MGSolverT` which is defined in the files `MGSolverT.C`, `MGSolverT3D.C` and `MGSolverT.h`. The parameters can be set in the file `/config/MGSTconf.h`.

1.1.2 $\kappa - \epsilon$ turbulence model

Standard $\kappa - \epsilon$ turbulence model

In the standard $\kappa - \epsilon$ turbulence model the *turbulent viscosity* is modelled as

$$\mu_t = \rho\nu_t\rho C_\mu \frac{k^2}{\epsilon}\,. \tag{1.32}$$

The *turbulent kinetic energy* k satisfies the following equation

$$\frac{\partial}{\partial t}\,k + \nabla\cdot(k\,\mathbf{v}) = \nabla\cdot\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\nabla k\right] + P_k + P_b - \epsilon\,, \tag{1.33}$$

where P_k is the production term of k and P_b the buoyancy term. For the *turbulent dissipation energy* ϵ we have

$$\frac{\partial}{\partial t}\epsilon + \nabla\cdot(\epsilon\,\mathbf{v}) = \nabla\cdot\left[\left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right)\nabla\epsilon\right] + C_{1\epsilon}\frac{\epsilon}{k}\,(P_k + C_{3\epsilon}P_b) - C_{2\epsilon}\frac{\epsilon^2}{k} \tag{1.34}$$

where the model constants are

$$C_{1\epsilon} = 1.44,\;\; C_{2\epsilon} = 1.92,\;\; C_\mu = 0.09,\;\; \sigma_k = 1.0,\;\; \sigma_\epsilon = 1.3\,. \tag{1.35}$$

The production P_k of k is defined as

$$P_k = -\overline{v_i' v_j'}\frac{\partial v_j}{\partial x_i} = \nu_t S^2\,, \tag{1.36}$$

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

$$S \equiv \sqrt{2S_{ij}S_{ij}} = \frac{1}{2}\|\nabla\mathbf{v} + \nabla\mathbf{v}^T\|\,. \tag{1.37}$$

The effect of buoyancy term P_b is given by

$$P_b = \alpha_t \frac{\mu_t}{Pr_t} \mathbf{g} \cdot \nabla T \,, \tag{1.38}$$

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 of thermal expansion, α_t, is defined as

$$\alpha_t = -\frac{1}{\rho} \left(\frac{\partial \rho}{\partial T} \right)_p \,. \tag{1.39}$$

Finite element $\kappa - \epsilon$ 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.

d) **Turbulent kinetic energy equation**

$$\begin{aligned} \int_\Omega \frac{\partial\, \kappa_h}{\partial t}\, \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot (\mathbf{v}_h\, \kappa_h)\, \varphi_h \, d\mathbf{x} + \int_\Omega \left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla \kappa_h \cdot \nabla \varphi_h \, d\mathbf{x} = \\ \int_\Omega (P_{kh} + P_{bh})\, \varphi_h \, d\mathbf{x} - \int_\Omega \epsilon_h\, \varphi_h \, d\mathbf{x} \qquad \forall \varphi_h \in K_h(\Omega) \,. \end{aligned} \tag{1.40}$$

e) **Turbulent dissipation energy equation**

$$\begin{aligned} \int_\Omega \frac{\partial\, \epsilon_h}{\partial t}\, \varphi_h \, d\mathbf{x} + \int_\Omega \nabla \cdot (\mathbf{v}_h\, \epsilon_h)\, \varphi_h \, d\mathbf{x} + \int_\Omega \left(\nu + \frac{\nu_t}{\sigma_\epsilon} \right) \nabla \epsilon_h \cdot \nabla \varphi_h \, d\mathbf{x} = \\ \int_\Omega C_{1\epsilon} \frac{\epsilon}{k} (P_{kh} + C_{3\epsilon} P_{bh})\, \varphi_h \, d\mathbf{x} - \int_\Omega C_{2\epsilon}\, \frac{\epsilon^2}{k}\, \varphi_h \, d\mathbf{x} \qquad \forall \varphi_h \in E_h(\Omega) \,, \end{aligned} \tag{1.41}$$

with all the constants defined as above.

Finite element class for the $\kappa - \epsilon$ system

The $\kappa - \epsilon$ turbulence solver is implemented in the class `MGSolverKE` which consists of the declaration file `MGSolverKE.h` and the implementation files `MGSolverKE.C`, `MGSolverKE3D.C`. The parameters can be set in the file `/config/MGSKEconf.h`.

1.1.3 $\kappa - \omega$ turbulence models

Wilcox's $\kappa - \omega$ turbulence model

The $\kappa - \omega$ model is one of the most common turbulence models. It still consists of two transport equations to represent the turbulent properties of the flow. This two equation model takes into account effects like convection and diffusion of turbulent energy. As in the $\kappa - \epsilon$ model the first transported variable is the turbulent kinetic energy k. The second

transported variable is the *specific dissipation rate* ω which is the variable that determines the scale of the turbulence. There are many $\kappa-\omega$ turbulence models. The most used are: Wilcox's $\kappa-\omega$ model, Wilcox's modified $\kappa-\omega$ model and SST $\kappa-\omega$ model. In the code we have implemented the standard $\kappa-\omega$ model (Wilcox's $\kappa-\omega$ model). In the standard $\kappa-\omega$ turbulence model the turbulent viscosity is modelled as

$$\mu_t = \rho\frac{k}{\omega}\,. \tag{1.42}$$

The turbulent kinetic energy k satisfies the following equation [6]

$$\frac{\partial k}{\partial t} + \nabla\cdot(k\mathbf{v}) = P_k - \beta^* k\omega + \nabla\cdot\left[(\nu+\sigma^*\nu_T)\,\nabla k\right] \tag{1.43}$$

and the specific dissipation rate ω satisfies

$$\frac{\partial \omega}{\partial t} + \nabla\cdot(\omega\mathbf{v}) = \alpha\frac{\omega}{k}\,P_k - \beta\omega^2 + \nabla\cdot\left[(\nu+\sigma\nu_T)\,\nabla\omega\right] \tag{1.44}$$

where we have the closure coefficients and auxiliary relations defined by

$$\alpha = \frac{5}{9}\quad \beta = \frac{3}{40}\quad \beta^* = \frac{9}{100}\quad \sigma = \frac{1}{2}\quad \sigma^* = \frac{1}{2}\quad \varepsilon = \beta^*\omega k\,. \tag{1.45}$$

Finite element $\kappa-\omega$ turbulence model

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.

d) **Turbulent kinetic energy equation**

$$\begin{aligned}\int_\Omega \frac{\partial\,\kappa_h}{\partial t}\,\varphi_h\,d\mathbf{x} + \int_\Omega \nabla\cdot(\mathbf{v}_h\,\kappa_h)\,\varphi_h\,d\mathbf{x} + \int_\Omega (\mu+\nu_t*\sigma^*)\,\nabla\kappa_h\cdot\nabla\varphi_h\,d\mathbf{x} = \\ \int_\Omega P_{kh}\,\varphi_h\,d\mathbf{x} - \int_\Omega \beta^*\kappa_h\,\omega_h\,\varphi_h\,d\mathbf{x} \qquad \forall\varphi_h\in K_h(\Omega)\,.\end{aligned} \tag{1.46}$$

e) **Turbulent specific dissipation rate equation**

$$\begin{aligned}\int_\Omega \frac{\partial\,\omega_h}{\partial t}\,\varphi_h\,d\mathbf{x} + \int_\Omega \nabla\cdot(\mathbf{v}_h\,\omega_h)\,\varphi_h\,d\mathbf{x} + \int_\Omega (\nu+\nu_t\sigma)\,\nabla\omega_h\cdot\nabla\varphi_h\,d\mathbf{x} = \\ \int_\Omega \alpha\frac{\omega}{\kappa}P_{kh}\,\varphi_h\,d\mathbf{x} - \int_\Omega \beta\omega^2\,\varphi_h\,d\mathbf{x} \qquad \forall\varphi_h\in W_h(\Omega)\,.\end{aligned} \tag{1.47}$$

with all the constants defined as above.

Finite element class for the $\kappa-\omega$ system

The $\kappa-\omega$ turbulence model solver is implemented in the class `MGSolverKW` which consists of the declaration file `MGSolverKW.h` and the implementation files `MGSolverKW.C`, `MGSolverKW3D.C`. The parameters can be set in the file `/config/MGSKWconf.h`.

1.1.4 LES

Large eddy simulation (LES) is a popular technique for simulating turbulent flows. An implication of Kolmogorov's theory of self similarity (1941) is that the large eddies of the flow are dependent on the geometry while the smaller eddies are not. 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).

Smagorinsky-Lilly model

The Smagorinsky model could be summarized as:

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\left(C_s\Delta\right)^2 \left|\bar{S}\right| S_{ij}\,. \tag{1.48}$$

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

$$\mu_{sgs} = \rho\left(C_s\Delta\right)^2 \left|\bar{S}\right|\,, \tag{1.49}$$

where the filter width is usually taken to be

$$\Delta = (\text{Volume})^{\frac{1}{3}} \tag{1.50}$$

and

$$\bar{S} = \sqrt{2S_{ij}S_{ij}}\,. \tag{1.51}$$

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.

Implementation of the *LES* turbulence model

The *LES* turbulence model is implemented in the class `MGSolverNS` in the file `MGSolverNS.C` The parameters can be set in the file `/config/MGSNSconf.h`.

1.2 Reactor core region model

In the core region the geometry is so detailed that a direct simulation with the purpose of computing the velocity, pressure and temperature distributions is not possible and an approximation is necessary. The approximation is presented in [3] and in this section we briefly recall the equations.

1.2.1 Two-level finite element Navier-Stokes system

General model

Let us consider a two level solution scheme where a fine level and a coarse level solution can be defined. 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.

a) **Incompressibility constraint.** 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 and therefore

$$\int_\Omega (\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}}))\, \psi_h(k)\, d\mathbf{x} = 0\,. \tag{1.52}$$

Now let $(\hat{p}_h, \widehat{\mathbf{v}}_h)$ be the solution 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 (1.1) we have [3]

$$\int_\Omega (\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}}))\, \widehat{\psi}_h(k)\, d\mathbf{x} = \int_\Omega P^c_{ef}(\widehat{\mathbf{v}}_h - \mathbf{v}_h)\, d\mathbf{x}\,, \tag{1.53}$$

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

$$R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \int_\Omega \nabla \cdot \rho(\widehat{\mathbf{v}}_h - \mathbf{v})\, \widehat{\psi}_h(k)\, d\mathbf{x}\,. \tag{1.54}$$

a) **Momentum equation.** 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 equation

$$\begin{aligned} &\int_\Omega \frac{\partial\, \rho\, \mathbf{v}_h}{\partial t} \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h}) \cdot \phi_h(i)\, d\mathbf{x} - \\ &\quad \int_\Gamma (\bar{\tau}_h \cdot \vec{n} - p_h \vec{n}) \cdot \phi_{\mathbf{h}}(i)\, d\mathbf{s} - \\ &\quad\quad \int_\Omega p_h\, \nabla \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega \bar{\tau}_h : \overline{\nabla \phi_h}(i)\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \phi_h(i)\, d\mathbf{x} = 0 \end{aligned} \tag{1.55}$$

for all the elements of the 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 (1.55) we have [3]

$$\begin{aligned} &\int_\Omega \frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \\ &\quad \int_\Omega \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega \bar{\bar{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)}\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \\ &\quad\quad \int_\Omega R^m_{cf}(p_h, \mathbf{v}_h, \widehat{p}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x}\,, \end{aligned} \tag{1.56}$$

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

$$\int_\Omega R^m_{cf}(p_h, \mathbf{v}_h, \widehat{p}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \tag{1.57}$$
$$\int_\Omega P^m_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} +$$
$$\int_\Omega S^m_{cf}(p_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega K^m_{cf}(\mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x}\,.$$

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 [3] and the turbulent transfer operator $T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h)$ by

$$T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = -\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h} + \nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h} - \nabla \cdot \rho\, \overline{(\widehat{\mathbf{v}}_h - \mathbf{v}_h)(\widehat{\mathbf{v}}_h - \mathbf{v}_h)}\,. \tag{1.58}$$

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

$$\int_\Omega S^m_{cf}(p_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = -\int_\Gamma p_h\, \vec{n} \cdot \widehat{\phi}_h(k)\, d\mathbf{s} \tag{1.59}$$

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

$$\int_\Omega K^m_{cf}(\mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \int_\Gamma (\bar{\tau}_h \cdot \vec{n}) \cdot \widehat{\phi}_h(k)\, d\mathbf{s}\,. \tag{1.60}$$

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 operator.

b) **Energy equation.** We can apply the same procedure for the energy equation [3]. 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 obtained by solving

$$\int_\Omega \frac{\partial \rho\, C_p\, T_h}{\partial t}\, \varphi_h(i)\, d\mathbf{x} + \int_\Omega \nabla \cdot (\rho\, \mathbf{v}_h C_p\, T_h)\, \varphi_h(i)\, d\mathbf{x} - \tag{1.61}$$
$$\int_\Gamma k\, (\nabla T_h \cdot \vec{n})\, \varphi_h(i)\, d\mathbf{s} - \int_\Omega \Phi_h\, \varphi_h(i)\, d\mathbf{x} + \int_\Omega k\, \nabla T_h \cdot \nabla \varphi_h(i)\, d\mathbf{x} - \int_\Omega Q_h\, \varphi_h(i)\, d\mathbf{x} = 0$$

for all basis element $\varphi_h(i)$ in $\mathbf{H}_h(\Omega)$.

If we introduce the coarse level solution $\widehat{T}_h$ in (1.61) we have [3]

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega \nabla \cdot (\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} -$$
$$\int_\Omega \Phi_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega k\, \nabla \widehat{T}_h \cdot \nabla \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega Q_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} = \tag{1.62}$$
$$\int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,,$$

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

$$\int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\,d\mathbf{x} = \int_\Omega S^e_{cf}(T_h)\,\widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)\ \widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h)\ \widehat{\varphi}_h(k)\,d\mathbf{x}\,. \tag{1.63}$$

where

$$P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.64}$$

is the energy fine-coarse transfer operator [3] and

$$T^e_{cf}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla\cdot(\rho\, C_p\, \widehat{\mathbf{v}}_h\, \widehat{T}_h) - \nabla\cdot(\rho\, C_p\, \mathbf{v}_h\, T_h) - \nabla\cdot(\rho\, C_p\, (\widehat{\mathbf{v}}_h - \mathbf{v}_h)\, (\widehat{T}_h - T_h))\,. \tag{1.65}$$

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

$$\int_\Omega S^e_{cf}(T_h)\,\widehat{\varphi}_h(k)\,d\mathbf{x} = \int_\Gamma k\,(\nabla T_h\cdot\vec{n})\,\widehat{\varphi}_h(k)\,d\mathbf{x}\,. \tag{1.66}$$

Reactor transfer operators

In order to complete the equations in Section 1.1 we must define the reactor transfer operators in working conditions.

a) **Incompressibility transfer operators**

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

  $$P^c_{ef} = 0\,. \tag{1.67}$$

  The assumption is exact since

  $$P^c_{ef}(\widehat{\mathbf{v}}_h - \mathbf{v}_h) = \nabla\cdot\rho(\widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.68}$$

  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.

b) **Momentum transfer operators**

- 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

  $$T_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = \nabla\cdot\bar{\widehat{\tau}}^\tau_h \tag{1.69}$$

  where the turbulent tensor $\bar{\widehat{\tau}}^\tau_h$ is defined as

  $$\bar{\widehat{\tau}}^\tau_h = 2\mu_\tau D(\widehat{\mathbf{v}}_h) \tag{1.70}$$

  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 in a similar way by

$$\begin{aligned} &P_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) = \\ &\zeta(\mathbf{x})\Big(\frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k) \; + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \\ &\qquad \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \bar{\bar{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)} \; - \; \rho \mathbf{g} \cdot \widehat{\phi}_h(k) - \nabla \cdot \bar{\bar{\tau}}^{eff}\Big) \end{aligned} \tag{1.71}$$

  where $\zeta(\mathbf{x})$ is the fraction of fuel and structural material in the total volume. The tensor $\bar{\bar{\tau}}^{eff}$ is defined as

$$\bar{\bar{\tau}}^{eff}_h = 2\mu^{eff} D(\widehat{\mathbf{v}}_h)\,. \tag{1.72}$$

  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

$$S^m_{cf}(p_h) = 0\,. \tag{1.73}$$

- $K^m_{cf}(v_h)$. 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 channel and that the loss of pressure in this channel can be compute with classical engineering formulas. In working conditions for forced motion in equivalent channels we may set

$$K^m_{cf}(\mathbf{v}_h) = \zeta(\mathbf{x}) \frac{\rho 2 \widehat{\mathbf{v}}_h |\widehat{\mathbf{v}}_h|}{D_{eq}} \lambda \tag{1.74}$$

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

c) **Energy transfer operators**

- 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

$$T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla \cdot (\frac{\mu_t}{Pr_t} \nabla \widehat{T}_h)\,, \tag{1.75}$$

  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

$$\begin{aligned} &P^e_{cf}(T_h - \widehat{T}_h) = \\ &\zeta(\mathbf{x})\Big(\frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t} \; + \nabla \cdot (\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h) - \Phi_h \; - \; Q_h - \nabla \cdot (k^{eff} \nabla \widehat{T}_h)\Big) \end{aligned} \tag{1.76}$$

  where $\zeta(\mathbf{x})$ is the fraction of fuel and structural material in the volume. The values of k^{eff} depends 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

$$S^e_{cf}(p_h) = W_0 \cos\left(\frac{\pi(z-(H_{in}+H_{out})/2)}{H_{out}-H_{in}}\right). \tag{1.77}$$

where $H_{in}$and H_{out} are the heights where the heat generation starts and ends. The quantity W_0 is assumed to be a known function of space which is defined by the power distribution factor (see Section 3.2.2)

1.2.2 Core model equations

Model Equations

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

$$\rho(T,P) = \rho_0(T)\exp(\beta p) \tag{1.78}$$

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

a) **FEM incompressibility equation**

$$\int_\Omega \left(\beta\,\frac{\partial p_h}{\partial t} + (\nabla\cdot\rho\,\widehat{\mathbf{v}}_h)\right)\widehat{\psi}_h(k)\,d\mathbf{x} = 0\,. \tag{1.79}$$

b) **FEM momentum equation**

$$\begin{aligned}\int_\Omega \frac{\partial\,\rho\,\widehat{\mathbf{v}}_h}{\partial t}\cdot\widehat{\phi}_h(k)\,d\mathbf{x} + \int_\Omega (\nabla\cdot\rho\,\overline{\widehat{\mathbf{v}}_h\widehat{\mathbf{v}}_h})\cdot\widehat{\phi}_h(k)\,d\mathbf{x} - \\ \int_\Omega \widehat{p}_h\,\nabla\cdot\widehat{\phi}_h(k)\,d\mathbf{x} + \int_\Omega (\bar{\bar{\tau}}_h + \bar{\bar{\tau}}^{\tau}_h + \bar{\bar{\tau}}^{eff}_h) : \overline{\nabla\widehat{\phi}_h(k)}\,d\mathbf{x} + \\ \int_\Omega \frac{2\rho\,\widehat{\mathbf{v}}_h|\widehat{\mathbf{v}}_h|}{D_{eq}}\lambda\cdot\widehat{\phi}_h(k)\,d\mathbf{x} - \int_\Omega \rho\mathbf{g}\cdot\widehat{\phi}_h(k)\,d\mathbf{x} = 0\end{aligned} \tag{1.80}$$

b) **FEM energy equation**

$$\begin{aligned}\int_\Omega \frac{\partial\,\rho\,C_p\,\widehat{T}_h}{\partial t}\,\widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\,C_p\widehat{\mathbf{v}}_h\,\widehat{T}_h)\,\widehat{\varphi}_h(k)\,d\mathbf{x} - \\ \int_\Omega \Phi_h\,\widehat{\varphi}_h(k)\,d\mathbf{x} + \int_\Omega (k + k^{eff} + \frac{\mu_t}{Pr_t})\,\nabla\widehat{T}_h\cdot\nabla\widehat{\varphi}_h(k)\,d\mathbf{x} - \\ \int_\Omega Q_h\,\widehat{\varphi}_h(k)\,d\mathbf{x} - \int_\Omega W_{max}\,r(\mathbf{x})\,\cos\left(\frac{\pi(z-(H_{in}+H_{out})/2)}{H_{out}-H_{in}}\right)\,\widehat{\varphi}_h(k)\,d\mathbf{x} = 0\,.\end{aligned} \tag{1.81}$$

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.

Equations in strong form The variational FEM system (1.79-1.81) is equivalent to

$$\beta \frac{\partial p_h}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}}_h) = 0 \tag{1.82}$$

$$\frac{\partial \rho \widehat{\mathbf{v}}_h}{\partial t} + (\nabla \cdot \rho \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) = -\nabla \widehat{p}_h + \nabla \cdot (\bar{\bar{\tau}}_h + \bar{\bar{\tau}}_h^{\tau} + \bar{\bar{\tau}}_h^{eff}) - \frac{2\rho \widehat{\mathbf{v}}_h |\mathbf{v}_h|}{D_{eq}} \lambda + \rho \mathbf{g} \tag{1.83}$$

$$\frac{\partial \rho C_p \widehat{T}_h}{\partial t} + \nabla \cdot (\rho C_p \widehat{\mathbf{v}}_h \widehat{T}_h) = \Phi_h + \nabla \cdot (k + k^{eff} + \frac{\mu_t}{Pr_t}) \nabla \widehat{T}_h + \tag{1.84}$$

$$Q_h \widehat{\varphi}_h(k) + W_0 \, r \, \cos\left(\frac{\pi(z - (H_{in} + H_{out})/2)}{H_{out} - H_{in}}\right) .$$

The equations (1.82-1.84) must be completed with the appropriate boundary conditions.

Chapter 2

CFD sub-assembly fuel channel modeling

2.1 Heat exchange modeling for sub-assembly fuel channels

2.1.1 Introduction

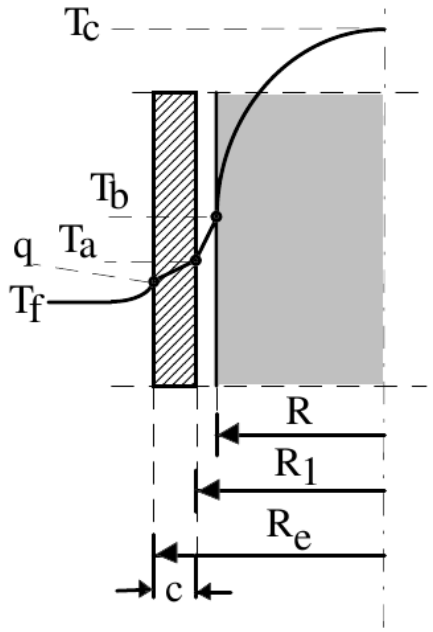

Figure 2.1: Temperature distribution in an assembly channel

The simulation of the core introduced in the previous chapter of this report takes into account average quantities over the assemblies and computes average coolant temperatures. When the coolant average temperatures are known then temperature profiles inside the fuel rod and the cladding can be computed by using standard assumptions and standard heat transfer correlations. In the liquid metal case and in three-dimensional configurations the heat exchange coefficient may not be constant along the vertical coordinate and standard heat exchange models cannot be appropriate. In this chapter we investigate the heat transfer

model between the liquid metal coolant and the fuel rod with CFD three-dimensional codes. In order to assess the validity of the standard heat transfer computational model we consider a very simple test which consists of the heat transfer flow around a rod in a single channel. For these CFD computations we use commercial and open source software available on the ENEA-GRID platform on CRESCO [2]. The computational results are then compared with experimental results from the KALLA facilities reported in [5].

The core computations previously proposed are able to define only average assembly temperatures. For temperatures inside the fuel rod we can use the average assembly coolant temperature T_f and standard analytical formulas. Let T_c be the temperature on the fuel rod axis and T_d the cladding temperature. We assume

$$T_c = T_f + \Delta T_1 + \Delta T_2 + \Delta T_3 + \Delta_4 = T_f + (T_c - T_b) + (T_b - T_a) + (T_a - T_d) + (T_d - T_f)\,. \quad (2.1)$$

Let q_l be the constant linear heat flux of the fuel rod. We have that

$$\Delta T_1 = (T_c - T_b) = \frac{q_l}{4\pi \bar{K}_f} \quad (2.2)$$

where $\bar{K}_f = \int_{T_b}^{T_c} K_f dT/(T_c - T_b)$,

$$\Delta T_2 = (T_b - T_a) = \frac{q_l\, R_c}{2\pi R_i} \quad (2.3)$$

where R_c and R_i are the fuel and the internal cladding radius,

$$\Delta T_3 = (T_a - T_d) = \frac{q_l\, s}{2\pi K_c} \quad (2.4)$$

with s the cladding thickness, K_c the conductivity of the cladding material and

$$\Delta T_4 = (T_d - T_f) = \frac{q_l}{2\pi R_e h_{df}} \quad (2.5)$$

with R_e the external cladding radius. The physical quantities K_f and K_c are well known but h_{df} must be determined. The heat transfer coefficient h_{df} is usually defined through the Nusselt number as

$$h_{df} = \frac{Nu\, k}{D_e q} = \frac{q''}{(T_d - T_f)}\,. \quad (2.6)$$

Close to the channel inlet the ratio h_{df} between the wall heat flux and $T_d - T_f$ may not be uniform and a brief analysis by using three-dimensional CFD tools is proposed in next section. The computation of h_{df} should be done in an appropriate bundle channel but, in order to simplify the computations, we proceed to obtain information on h_{df} by the single rod experiment. The computational method remains valid but we plan in the future to use the correct bundle geometry. With this in mind we compute the temperature jump ΔT_1, ΔT_2, ΔT_3 with data from literature and use the experimental and computational values of h_{df} for ΔT_4.

2.1.2 Single rod experiment with LBE coolant

In [5] three experiments are presented: a turbulent lead bismuth flow in a circular tube and two experiments in a 19-pin hexagonal rod bundle assembly in turbulent water. In the

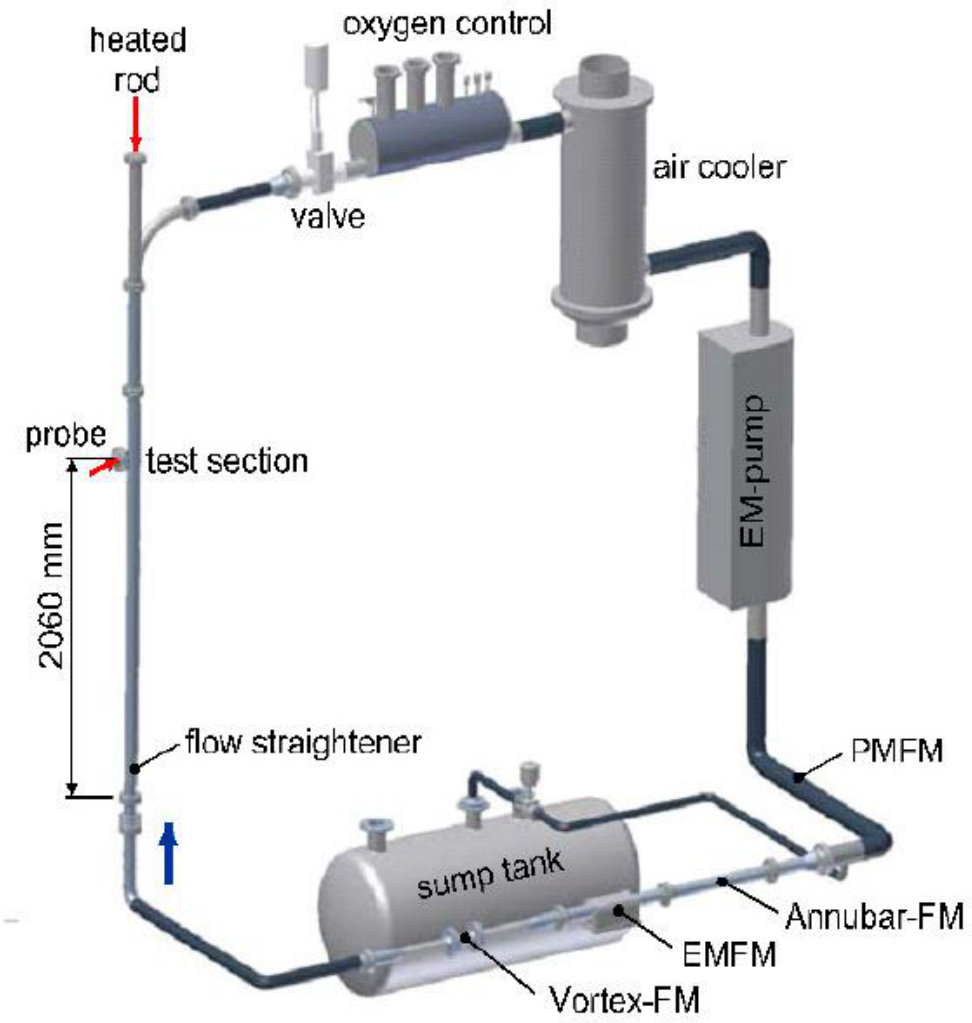

Figure 2.2: Scheme of the experiment

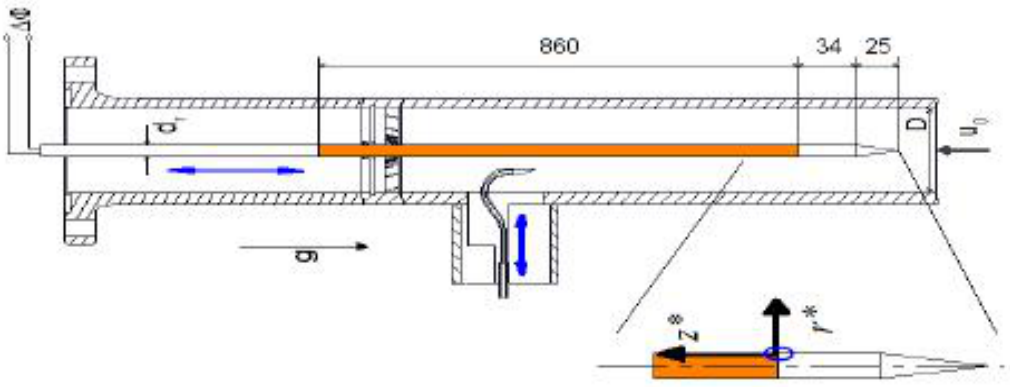

Figure 2.3: Long rod section of the experiment

Parameter	Experiment
Total power	$8\,kW$
Number of rods	1
Rod diameter	$8.2 \times 10^{-3}\,m$
Rod length	$1.200\,m$
Heated length	$0.860\,m$
Mean velocity	$0 - 1.6\,m/s$

Table 2.1: Data for the single rod experiment

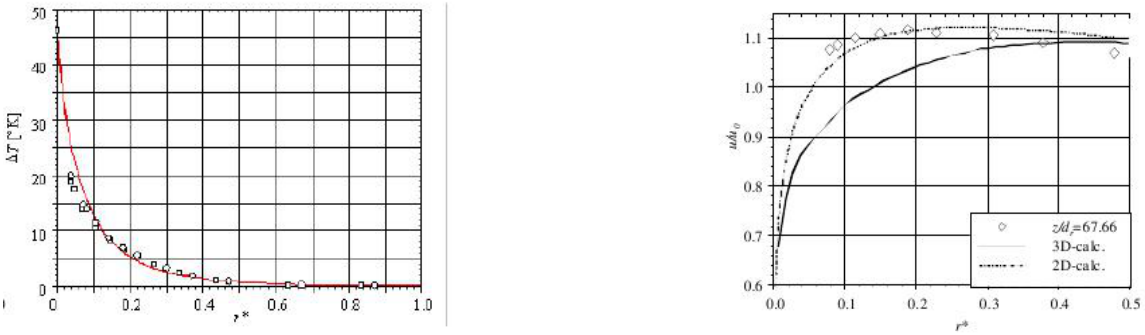

Figure 2.4: Experimental temperature (left) and velocity (right) profiles in the radial direction

first case the lead bismuth coolant flows vertically, heated by a rod placed concentrically in an circular pipe. This experiment essentially describes the thermal development of the temperature boundary layer on the fuel rod surface. The flow is studied experimentally by means of rakes consisting of several thermocouples combined with velocity sensors based on a Pitot tube. The attainable heat flux is $0.01\,W/m^2$ with Reynolds numbers ranging from 6×10^4 to 6×10^5.

The single rod experimental setup is shown in Figure 2.2. A pump pushes a liquid metal flow through the test section which consists of an electrically heated cylindrical rod placed vertically in a circular tube. A detailed view of the test section setup is given in Figure 2.3. The axial location of the rod is fixed by three spacers placed at equidistant positions of $0.370\,m$. The lower two spacers contain three wings and the upper spacer is built of four wings that are equipped with several calibrated thermocouples. The thermocouples are located to minimize the heat transfer through the wing distorting the temperature measurements. The wing tip thickness is in flow facing direction and spreads up to house all the thermocouples and wiring. The developing length of the tube flow is about 30 hydraulic diameters which is sufficient to obtain a hydrodynamic fully developed flow.

The heated rod test section is illustrated in Figure 2.3. In the experiment the temperature is measured by thermocouples mounted on the spacer and on the Pitot tube. The velocity is also measured by means of a Pitot tube. Measurements and numerical calculations are shown for a configuration with a mean velocity $\bar{u}$ of $0.77\,m/s$, an inlet temperature of 300 ℃and a heating power of $8\,kW$ generating a heat flux of $0.0040\,W/m^2$ over the inner cylinder. This corresponds to a Reynolds number of $Re = 2.7 \times 10^5$ based on the annular tube diameter and a corresponding Prandtl number of $Pr = 0.021$. The physical and geometrical data for the experiment are summarized in Figure 2.1. For this case the axial temperature profiles are as

the one reported on the left of Figure 2.4. In Figure 2.4 on the right there is a velocity profile as a function of the radius r^* at $z^* = 67.6$. The non-dimensional axial position z^* is set as $z^* = z/d_r$ with the heated rod diameter $d_r = 8.2 \times 10^{-3}\, m$ and the non-dimensional radial position r^* is defined as $r^* = y - (d_r/2)/(D/2) - (d_r/2)$ with the diameter of test section $D = 60.5 \times 10^{-3}\, m$. In Figure 2.4 on the left there is an experimental temperature profile as a function of the radius r^* at $z^* = 67.6$.

2.2 CFD results for the single rod configuration

2.2.1 Mesh and data setting

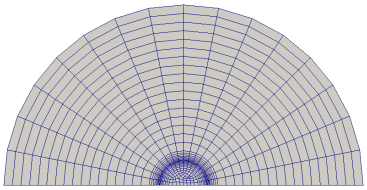

Figure 2.5: Coarse mesh: top view.

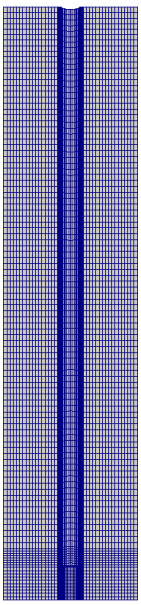

Figure 2.6: Coarse mesh: longitudinal section view.

The geometry used for the numerical simulations is given in Figures 2.5 and 2.6. First we generate a coarse mesh and then we refine it in order to obtain an accurate solution.

Concerning the mesh, the priority has been given to hexahedral elements since they have the advantage of little or no distortion with parallel segments that are perpendicular to the inlet plane. The procedure to construct hexahedral elements is long and tedious but the results are much more accurate. The choice of the ideal number of cells is done by iteration. We start with a sufficiently fine mesh and add cells until the precision of the results does not change significantly. The number of cells is a problem which interests the CPU time and memory storage and it can be considered optimal around few hundreds of thousands of cells. In order to limit the number of cells the mesh has been refined only in the boundary layer to obtain an optimal value of the non-dimensional distance from the wall y^+ for the turbulence models. This adaptive refinement can be obtained by using the appropriate function in SALOME or the boundary layer refinement tool in GAMBIT-FLUENT. This mesh has a fine boundary layer on the rod surface which gives a y^+ value of 12. This mesh is refined several times. This mesh is used by FLUENT, but also by SATURNE via a converter tool from FLUENT5/6 mesh format to the MED format. The MED format can be used in the SALOME platform and also in SATURNE [9].

2.2.2 Numerical results

Simulation of the single rod experiment

density ρ	$10340\,kg/m^3$
viscosity μ	$0.0018\,Pa\,s$
heat capacity Cp	$145\,J/(kg\,K)$
thermal conductivity λ	$11.7\,W/(m\,K)$

Table 2.2: Data for the experiment [4]

The thermophysical properties of the LBE used in all simulations are reported in Table 2.2. In this case we have an LBE, turbulent, steady flow with heat transfer in a circular tube with a length of $1.076\,m$ and a diameter of $0.605\,m$. The average speed of the flow at the inlet is $0.773\,m/s$ corresponding to a mass flow of $8\,m^3/h$ for average Reynolds number of 36411 with temperature of 300 ℃. The thermal power is $9\,kW$ corresponding to a heat flux of $4.064\,MW/m^2$ over the surface of the inner cylinder with a diameter of $4.1 \times 10^{-3}\,m$ and a length of $0.86\,m$. Actually the domain is not a simple annulus but contains various instrumentation and spacers needed to hold the heated rod. After various tests on the influence of the spacers we neglect them and consider only the simple geometry described before. By using the coarse mesh and $\kappa - \omega$ turbulence model the solution is reported in Figures 2.7-2.8. In Figure 2.7 we plot the temperature profile over a horizontal (top) and a vertical plane (bottom). In Figure 2.8 we show the pressure and the turbulent viscosity over the same vertical plane.

Simulations with different codes

We refine the mesh shown in Figure 2.5-2.6 to obtain a mesh with y^+ of the order of 6 and solve the state fields with both SATURNE and FLUENT codes [8, 10, 7]. We set the parameters as in Table 2.2 and use the $\kappa-\omega$ turbulence model. In Figure 2.9 temperature and velocity profiles for these computations are reported. On the left one can see the temperature

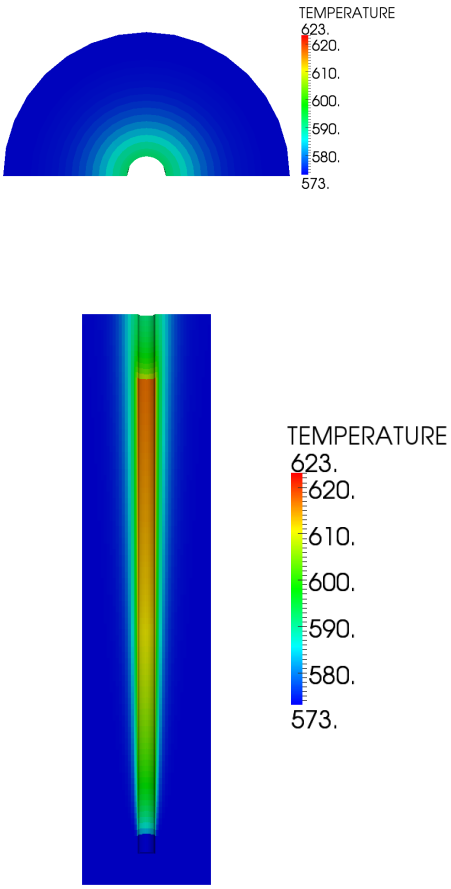

Figure 2.7: Temperature top (left) and section view (right)

solution along the radius at $z = 0.592$ for SATURNE (A) and FLUENT (B) together with the experimental data (Exp). The results match the experiment very closely and both SATURNE (A) and FLUENT give comparable results. These codes are Finite Volume Codes and the solution is computed at the center of the cell. The extrapolation from the center values to the cell side can give some important differences at the point closest to the wall. On the right we can see the velocity profile. Both SATURNE (A) and FLUENT (B) give comparable results which match the experimental results in the range where these are available.

Simulations for different turbulence models

In this paragraph we use different turbulence models to understand the influence of the model approximation. In Figure 2.10 we use $\kappa - \epsilon$ (A), $\kappa - \omega$ (B), LES (C) turbulence models and compare these results with the experimental data (Exp). With standard parameters and standard wall functions the $\kappa - \omega$ turbulence model results in the more accurate model while the LES is the less accurate one. However all these models give very promising results. On the left of the Figure 2.10 we have the temperature distribution and on the right the velocity

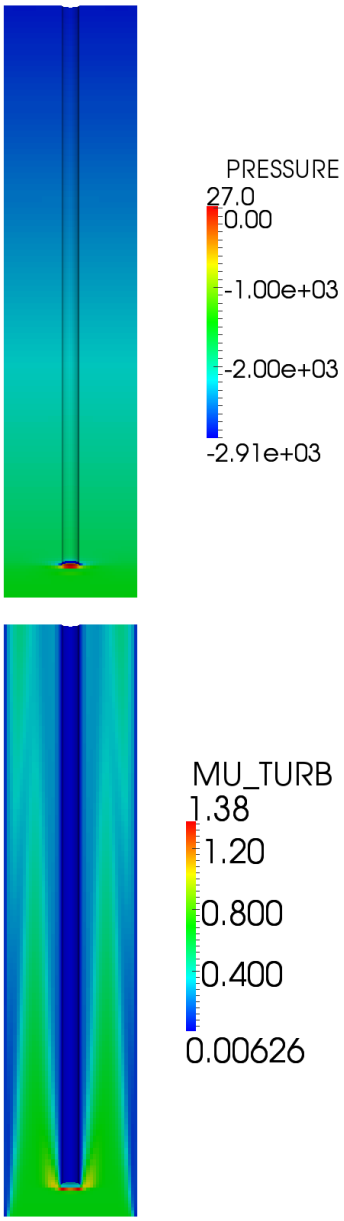

Figure 2.8: Pressure (left) and turbulent viscosity (right)

profile (at $z = 0.592\,m$).

Mesh and y^+

One of the most important parameters in CFD computations is the mesh size. For this reason we compute the temperature and velocity profiles for different meshes. Since the mesh is fine enough in the center of the domain and the temperature gradient is high close to the heated surface we refine only the boundary layer previously constructed to obtain smaller values of y^+. In Figures 2.11-2.12 we plot different profiles for the different values of $y^+ = 12$ (A), 6

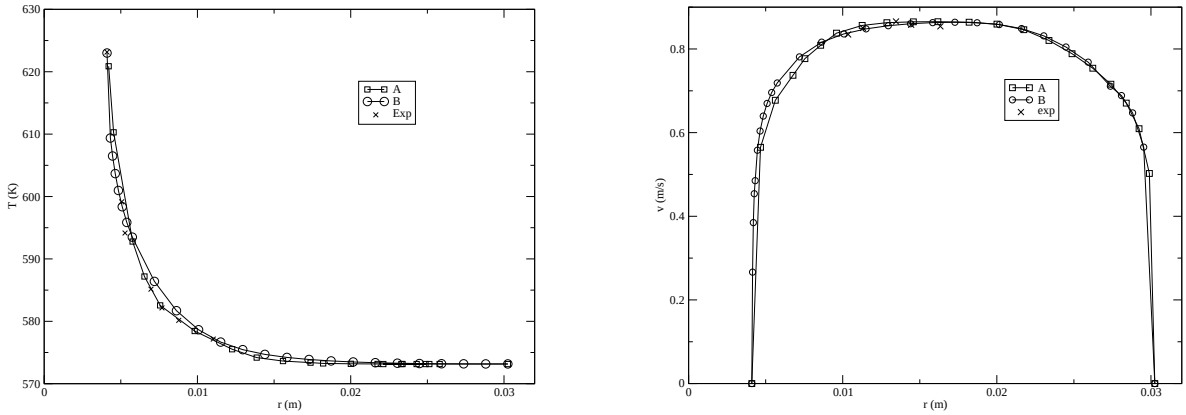

Figure 2.9: Temperature and velocity profile at $z = 0.592$ for (A) SATURNE, (B) FLUENT and (Exp) experimental data

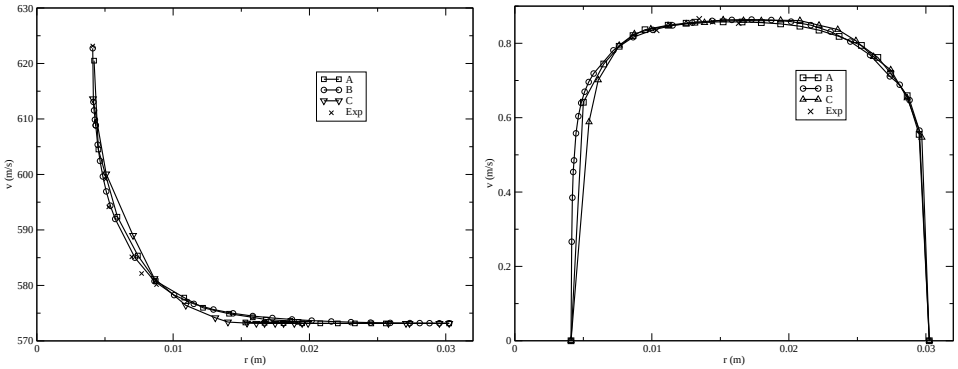

Figure 2.10: Temperature and velocity radial profiles at $z = 0.592\,m$ for different turbulent models: $\kappa - \epsilon$ (A), $\kappa - \omega$ (B), LES (C) and experimental results (Exp)

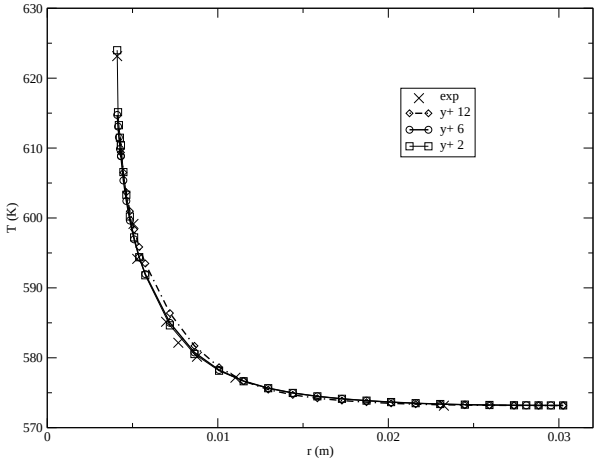

Figure 2.11: Temperature radial profile at $z = 0.592\,m$ for different y^+ values.

(B) and 2 (C) over the radius from the inner cylinder to the external wall. In Figure 2.11 we report the temperature profile while in Figure 2.12 we have the velocity along the radius

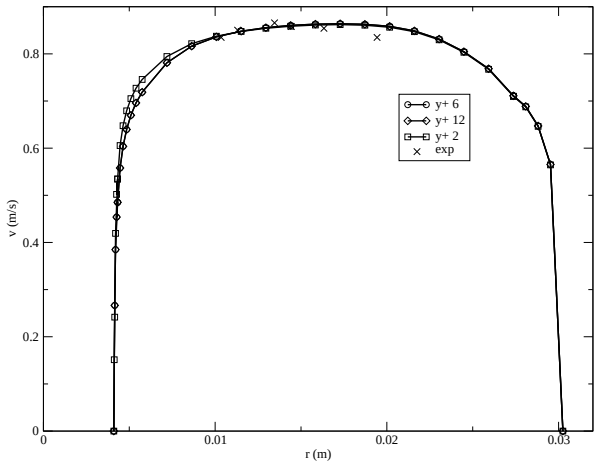

Figure 2.12: Velocity radial profile at $z = 0.592\,m$ for different y^+

over the plane $z = 0.592\,m$. The solution becomes better and better with more points on the boundary layer, i.e. when y^+ becomes smaller.

Computation of the heat transfer coefficient h_{df}

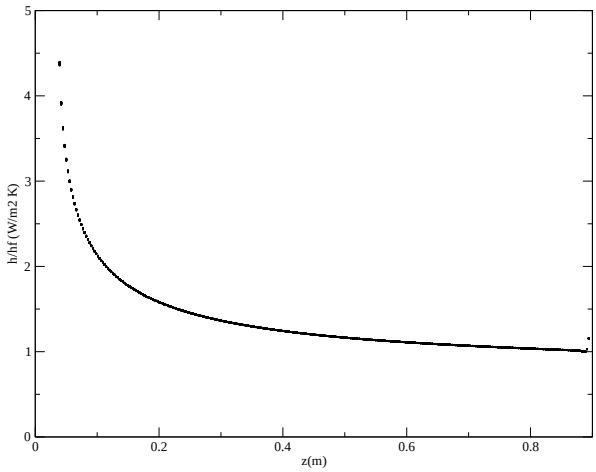

Figure 2.13: h_{df}/h_{teo} ratio value as a function of z along the heated wall.

Now we can compute the heat exchange coefficient h_{df} along the z-axis by the ratio

$$h_{df} = \frac{q_l}{\Delta T(z)}\,. \tag{2.7}$$

where $\Delta T(z) = T_w(z) - T_f(z)$ is the difference between the wall temperature and the fluid bulk temperature at the plane z. Since the linear heat flux q_l is constant and ΔT is not uniform over the z-axis we expect a variation of h_{cd} along the heated surface. If we suppose that h_{teo} is the value reached at the end of the channel then we can compute the ratio h_{df}/h_{teo} between the coefficient and the end value. The behaviour of h_{df}/h_{teo} is reported in Figure

2.13. For sufficiently long rod the value h_{teo} should approach the value taken from liquid metal standard correlations.

2.3 The ratio h_{df}/h_{teo} for lead coolant.

If we substitute the LBE coolant with lead coolant we have the properties in Table 2.3.

density ρ	10562 kg/m^3
viscosity μ	0.0022 $Pa\,s$
heat capacity Cp	147.3 $J/(kg\,K)$
thermal conductivity λ	16.58 $W/(m\,K)$

Table 2.3: Data for lead

By following the procedure discussed in the previous section we can obtain the ratio h_{df}/h_{teo} for the lead along the channel. Figure 2.14 shows the results for two different inlet velocities equal to 0.773 m/s (A) and 1. m/s (B). In this case the ratio h_{df}/h_{teo} is independent of the inlet velocity.

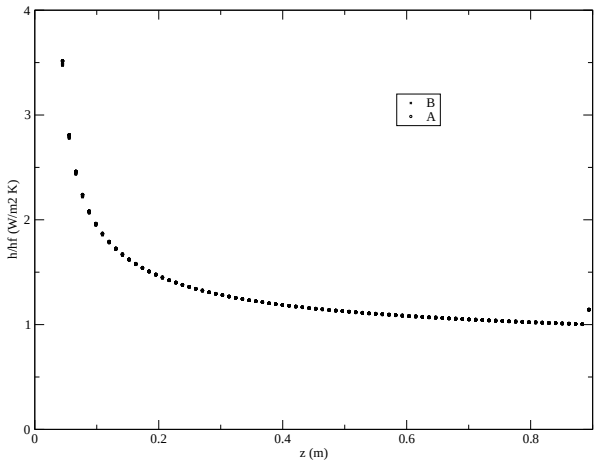

Figure 2.14: h_{df}/h_{teo} ratio value as a function of z in the heated wall.

In the computation proposed at the beginning of the chapter we can use the ratio h_{df}/h_{teo} to evaluate h_{df} along the channel assuming h_{teo} as the value obtained from standard correlations (from $Nu = Nu(Pe, Pr_t)$). One should use the ratio h_{df}/h_{teo} with extreme caution since different simulations may lead to different h_{df}/h_{teo}.

Chapter 3

Program user guide

3.1 Introduction

3.1.1 The version 2.0 of the reactor module code

This reactor code is a module of a more complex and wider CFD finite element code developed at the Laboratory of Montecuccolino at the University of Bologna. This code, called LIBMESH-unibo, is based on the open source LIBMESH library and therefore an extensive documentation on the basic classes can be found at `http://www.libmesh.org` [12]. In this User Guide we introduce a general and brief guide to some aspects which are relevant to this module. This program consists of the basic three-dimensional modules for solving the Navier-Stokes, the energy and the turbulent model equations. A special treatment of the Navier-Stokes equations is considered in the core region where some additional terms are considered. When the program is installed then its execution 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 (UG) refers to the version 2.0 of the code. The version 1.0 is discussed briefly in the document [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 runs on this version 2.0 and vice versa.

Installation.

There are two different installations of the code: the installation that includes the LIBMESH library and a simpler one where the LIBMESH library is not installed. The LIBMESH library is used to generate the multigrid mesh starting from the basic mesh generated by a mesh generator and therefore in the simple version, since the LIBMESH library is not installed, the multigrid mesh must be given as input file together with all restriction and prolongation matrices. Those can be easily generated by the code which has a complete installation or from any other machine with the same operating system. These mesh files can be found in the `data_in` directory after any execution of the code with the same geometry. In the simple

installation it is possible to change physical properties, boundary and initial conditions but not the geometry.

In order to install the package without the LIBMESH library one can use the package RMCFD-2.0.tar.gz. In this case one must choose a directory and run the following commands

- uncompress: `tar xzvf RMCFD-2.0.tar.gz`;
- go to the directory: `cd RMCFD-2.0`;
- run the configuration script: `./configure`;
- compile the program: `make ex13`.

In order to install the package with the LIBMESH library (and eventually the PETSC library for parallel computations) one must first install the LIBMESH library. The instruction for the LIBMESH installation can be found at `http://libmesh.org`. Once the LIBMESH is installed one can proceed to the installation of the reactor code from the package RMCFD-2.0.tar.gz as described above.

Step 1. Preprocessing and mesh generation

The mesh at the coarse level is generated by a mesh generator program such as GAMBIT (see for example http://www.FLUENT.com) or SALOME (see http://www.salome.org). Since the core is made of rectangular assemblies only HEX27 elements should be used. The multigrid mesh is generated starting from the coarse level mesh by the LIBMESH library. Some meshes at different levels are enclosed with this package. The basic coarse mesh from the GAMBIT mesh generator must be saved in `generic` format which is basically a text file. The basic coarse mesh from the SALOME mesh generator must be saved in `MED` format. The multilevel meshes, generated by the LIBMESH library, are in HDF5 storage format. Detailed information on MED and HDF5 formats can be found at http://www.salome-platform.org/ and http://www.HDF5.org/.

Step 2. Running and compiling

Make. Before running the code the program must be compiled. The code can be compiled by using the `make` command. The `make` command executes a chain of commands which are written inside the file `Makefile`. The following options are available

- `make ex13`: compile the program;
- `make clean`: remove the object files for the specified `METHOD`;
- `make clobber`: remove all object and executable files;
- `make distclean`: remove all object, executable files and dynamic libraries;
- `make contrib`: generate dynamic libraries needed for execution.

Run. In order to run the program one must execute the following commands

```
make ex13
ex13
```

We remark that the make command is necessary after any change in the configuration files (`config.h`) and in the source files (`*.h` and `*.C` with the exception of the file `parameters.in`). After changes in the `parameters.in` file the code does not need to be compiled and can run with the command `ex13`.

Restart the code. In order to restart the program from the iteration n at the time t it is necessary so set the following parameters in the `config.h` file

```
#define RESTART n
#define RESTARTIME t
```

and then execute

```
make ex13
ex13
```

Method. It is possible to run the code in two different modes by specifying the shell parameter `METHOD` before compilation. You can set this to optimized (`opt`) or to debug mode (`dbg`) using the command

```
export METHOD=opt
```

for optimized mode and similarly for debugging mode. The default mode is set to `opt`.

Step 3. Postprocessing and visualization.

The output is in XDMF format with the use of HDF5 storage format and detailed information can be found at http://www.xdmf.org/ and http://www.hdf5.org/. Both the XDMF files (with extension `.xmf`) and HDF5 files (with extension `h5`) are saved in the `output` directory. The open source PARAVIEW software is used to see the output files [13]. For tutorials and manuals one can see http://www.paraview.org and http://www.vtk.org/.

3.1.2 Code structure

Main directory and its structure

Once one opens the main directory it appears to be divided into some subdirectories. The program consists of a main directory where the source code is stored and seven additional subdirectories. The seven subdirectories are: `src`, `include`, `config`, `data_in`, `contrib`, `fem` and `output`.

The code is written in C++. The main file is `ex13.C` which can be found at the main directory. In the main directory there are also the files `ex13.h`, `gmsh.C` and `gmsh.h`.

The file `ex13.C` calls the mesh reader and initializes all the classes. By using the class functions the file `ex13.C` reads the core power distribution, solves the equations and prints all the results. The file `ex13.h` reads the mesh and generates all the necessary files for

different multigrid levels. This file is the only interface to the LIBMESH library. In the simple installation this file is not used and therefore the LIBMESH library is not used.

The files `gmsh.C` and `gmsh.h` are necessary for reading GAMBIT generic and SALOME MED files. They contain a table to read different elements created by the mesh generator and translate them into a different format.

The source code subdirectories `src` and `include`

The subdirectory `src` is the source directory where all the source files are stored. Here is the list of the main classes:

- the class `MGCase`, written in the files `MGCase.C` and `MGCase.h`, defines input and output data flow;
- the class `MGGauss`, written in the files `MGGauss.C` and `MGGauss.h`, defines Gaussian integration;
- the class `MGMesh`, written in the files `MGMesh.C` and `MGMesh.h`, defines the geometrical mesh;
- the class `MGSolBase`, written in the files `MGSolverBase.C` and `MGSolverBase.h`, defines the basic equation solver;
- the class `MGSolDA`, written in the files `MGSolverDA.C`, `MGSolverDA2D.C`, `MGSolverDA3D.C` and `MGSolverDA.h`, defines the basic equation solver on linear and quadratic elements;
- the class `MGSol`, written in the files `MGSolverNS3D.C` and `MGSolver.h`, defines the Navier-Stokes equation solver;
- the class `MGSolKE`, written in the files `MGSolverKE3D.C` and `MGSolverKE.h`, defines the $\kappa - \epsilon$ turbulence equation solver;
- the class `MGSolKW`, written in the files `MGSolverKW3D.C` and `MGSolverKW.h`, defines the $\kappa - \omega$ turbulence equation solver;
- the class `MGSolT`, written in the files `MGSolverT.C` `MGSolverT3D.C` and `MGSolverT.h`, defines the energy equation solver.

In order to set the initial and boundary condition functions one needs to open the corresponding source file. The initial and boundary condition functions for Navier-Stokes equations are the first two functions `ic_read` and `bc_read` in the file `MGSolNS3D.C`. In a similar way the initial and boundary condition functions for the energy equation are the first two functions in the file `MGSolverT3D.C`. The parameters of each class are defined in the `/config/class` directory in the corresponding configuration file. The turbulence model and classes are under development and in some versions the $\kappa - \epsilon$ and $\kappa - \omega$ can be found in a unique file. The LES model is implemented inside the Navier-Stokes class.

The directory `include` is the source directory where all the include files are stored. For example the Navier-Stokes solver class are defined in the file `MGSolverNS.h`, the energy equation in `MGSolverT.h`, the $\kappa - \epsilon$ turbulence model in `MGSolverKE.h`, etc... This file contains the constructor and destructor definitions and all the prototype functions of the corresponding class.

The subdirectory `config`

The subdirectory `config` consists of a directory `class`, and two configuration files: `config.h` with the configuration variables (solution of Navier-Stokes system, solution of energy system, boundary integration, etc ...) and `parameters.in` with constant material properties and numerical parameters. The subdirectory `class` contains all the local configuration files relative to each class, The use of the energy equation or the use of the turbulence model is configured by `config.h`. However all the parameters relative to the particular system must be changed in the `/config/class` and not in the main configuration file `config.h`. We note that the `data.h` file does not exist anymore and all the values in it are now in the file `parameters.in` that is read at runtime.

The data directories `data_in` and `fem`

The subdirectory `data_in` is the data directory. All the data necessary for the simulation must be stored here. The mesh, the matrices and all the prolongation and restriction operators are stored in this directory.

The partial differential equations governing the system must be discretized in matrices and vectors. The structures of these matrices and vectors are stored in files in the `data_in` directory. Each geometry requires a new set of files. The LIBMESH library is needed for this generation. In this directory you should find the files as

- `mesh.msh`. The mesh file from the mesh generator
- `mesh.h5`. The mesh file in HDF5 storage format
- `(S)matrix(l).in`. The matrix file relative to the system of type (S) at level (l);
- `(S)prol(l).in`. The prolongation operator relative to the system of type (S) at level (l);
- `(S)rest(l).in`. The restriction operator relative to the system of type (S) at level (l);
- `mesh.data`. The power distribution and pressure loss factor

If the `mesh.data` is not in this directory it will be generated from the program (with all values 1).

The subdirectory `fem` is the finite element directory where the finite element Gaussian point values for integration are stored. Only the finite element HEX27, HEX8, QUAD9, QUAD4, EDGE3 and EDGE2 are in this package. A program is designed to generate these files by using the LIBMESH library. Only HEX27 elements are considered for a reactor core and therefore the program is not included. This directory `fem` consists of several files with Gaussian integration point values. The name of the files have the form

- `shape(N)D_(G)(F)0302.in`

where (N) is the geometrical dimension (1,2 o 3), (G) is the order of the Gaussian integration (09 or 27) and (F) is the type of the finite element (02 for Linear EDGE3 linear finite element, 04 for QUAD4 linear finite element, 09 for QUAD9 quadratic finite element, 08 for HEX8 linear finite element and 27 for HEX27 quadratic finite element)

The subdirectory `output`

The subdirectory `output` is the directory where the results are stored. Meshes, boundaries and field values can be found there for each time step. The data are stored in XDMF and HDF5 formats which can be read using the open-source software PARAVIEW. The file in HDF5 format can be read by the `hdfview` software in its table form. In this directory one can find different types of files: case, mesh and solution files. The case file stores boundary and initial conditions and its name has the form

- `case.(I).F`

where (I) is the initial time-step number and (F) the extension (`.xmf` for XDMF format and `.h5` for HDF5 data format). The mesh file contains boundary and volume elements and it is named

- `meshxmf.F`

where (F) is the extension (`.xmf` for XDMF format and `.h5` for HDF5 data format). The solution files store all field values for each time-step and their names have the form

- `sol.(N).F`

where (N) is the time-step number and (F) is the extension (`.xmf` for XDMF format and `.h5` for HDF5 data format). Finally the time sequence collection file is named

- `time.(I).F`

where (I) is the initial time-step number and (F) is the extension (`.xmf` for XDMF format and `.h5` for HDF5 data format).

The subdirectory `contrib`

The subdirectory `contrib` is the contribution directory. This directory stores the LASPACK library, the `datagen` executable and the `matrix` directory. The package LASPACK is a linear algebra library for solving large sparse linear systems. For details see

http://www.mgnet.org/mgnet/Codes

The `datagen` directory contains the files to define the core heat generation and the pressure drop factors. For details see Section 3.2.2 The `matrix` directory contains the interface implementation of the linear algebra functions. The interface can be connected with different linear algebra packages such as LASPACK and PETSC library [17, 11]. The interface for the PETSC library is not implemented in this version.

3.2 Step 1: Mesh generation and core factor files

3.2.1 Mesh generation

The first step in the computation of the velocity, pressure and temperature distributions inside the reactor is the generation of the mesh geometry. A typical reactor core geometry is shown in Figure 3.1. The core consists of several assemblies with rectangular shape. In

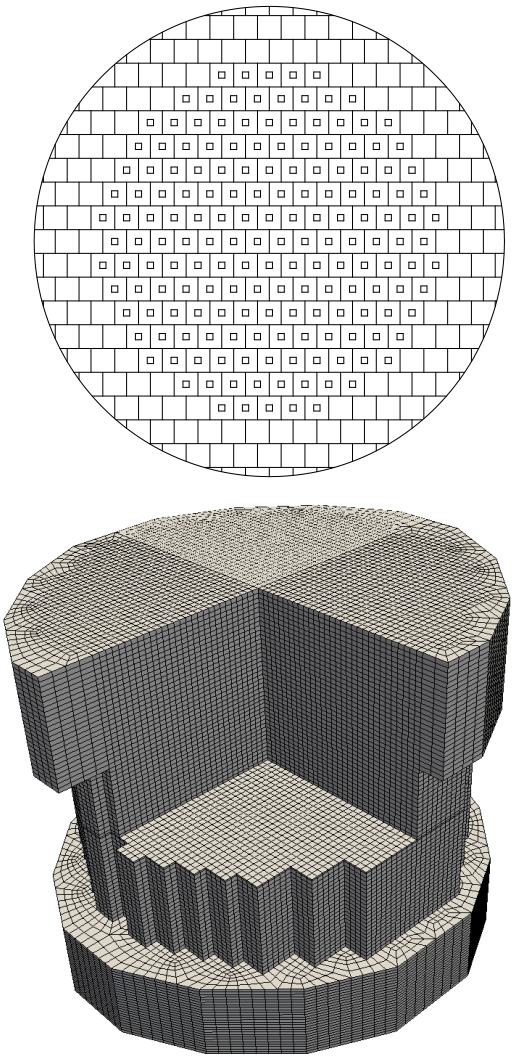

Figure 3.1: Core design and reactor mesh

order to impose correctly the heat source inside the assembly we must use Cartesian finite elements. The finite element for the reactor simulation must be HEX8, HEX20 and HEX27 the hexahedral finite element with 8,20 or 27 points. Once the coarse mesh is known the code can generate multilevel mesh with HEX27 finite element or with HEX20 finite elements.

For cartesian geometries the code has an internal mesh generator which needs only very simple settings. For more complex geometries one must convert the file to a compatible format. The code can handle a limited number of mesh formats: the generic mesh GAMBIT format and the SALOME mesh MED format. 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.

The final mesh for the code is stored using HDF5 storage and XDMF format. For details about these formats see the post-processing Section 3.4.

Internal mesh generator

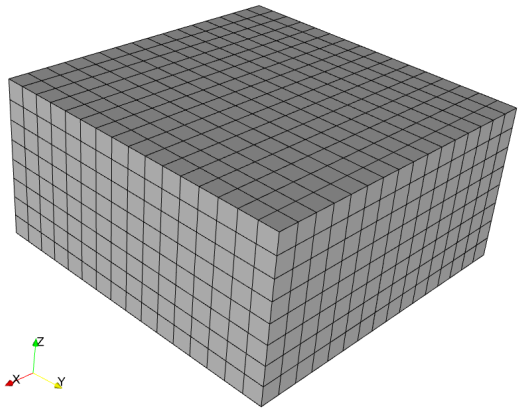

Figure 3.2: Mesh from the internal generator

Inside the code there is an internal mesh generator with HEX27 finite elements. This generator is limited to regular cube and cartesian geometries. To obtain a reactor of dimension $[a, b] \times [c, d] \times [e, f]$ with $N_x \times N_y \times N_z$ one must set the function inside the file `ex13.h`. The command is written as

```
MeshTools::Generation::build_cube
  (*msh[0],nintervx,nintervy,nintervz,0,1,0,1,0,1,HEX27);
```

in the form

```
MeshTools::Generation::build_cube
  (*msh[0],N_x,N_y,N_z,a,b,c,d,e,f,HEX27);
```

An example of mesh is shown in Figure 3.2. The mesh is generated together with all the multilevel meshes ready for multigrid solvers.

GAMBIT generic mesh format

The GAMBIT mesh generator is a commercial software. Tutorial and user guide can be found at http://www.fluent.us. The GAMBIT software must be set to `generic`. The `FLUENT5` option instead of `generic` does 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. In order to set the heat power generated in the reactor correctly one must mesh the assembly of the core exactly and reproduce all the assemplies which are not regularly distributed. Finally the mesh must be exported with the `Export --> Mesh` command in the `File` menu.

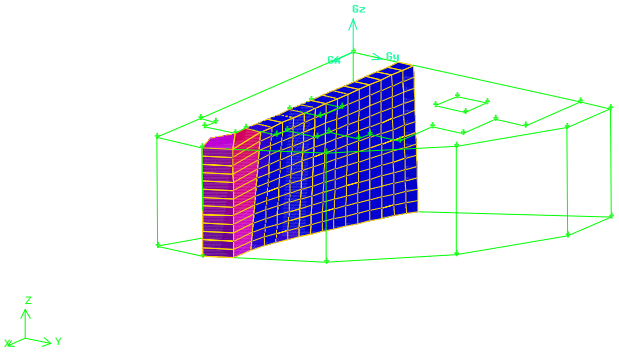

Figure 3.3: GAMBIT mesh

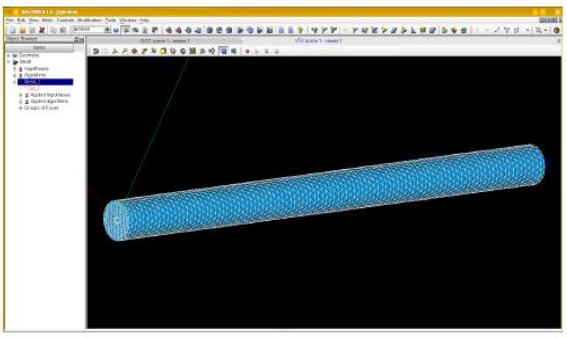

Figure 3.4: SALOME mesh

SALOME mesh and MED Format

SALOME is a free software that provides a generic platform for pre and post processing for numerical simulation [9]. 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 http://www.salome.org. Universal binaries are also provided for 32 and 64 bits Linux distributions as tgz archive files. SALOME has the following main modules:
a) KERNEL: a module for distributed component management, study management and general services;
b) GUI: a module for graphical user interface;
c) GEOM: a module for creating, editing, importing and exporting CAD models;
d) MESH: a CAD program that uses a standard meshing algorithm or any external mesher;
e) MED: a module for MED data file management;
f) POST: a dedicated viewer to analyse the results of solver computations;

g) YACS: a module involving multi-solver coupling codes;

For mesh generation one must set the module GEOM, see Figure 3.4, and use its mesh capability. SALOME meshing plug-in modules are used to generate 2D/3D meshes in SALOME Mesh modules. Inside SALOME the following meshing plug-in modules are available:
a) Free: NETGEN
b) Commercial: GHS3D, GHS3D parallel, BLSURF, Hexotic.
In addition, SALOME distribution allows to export the mesh MED and UNV formats.

The reactor mesh can be generated by the SALOME MESH module. The MESH module must use Hexahedron mesh capability 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.

3.2.2 Core power and pressure loss distribution

Core power distribution input file

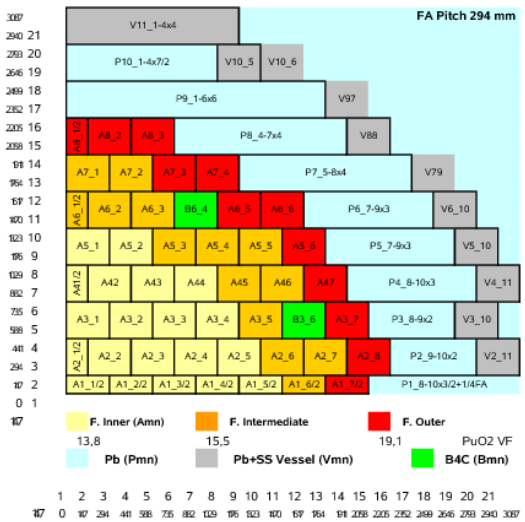

Figure 3.5: Typical core power distribution

The power distribution and the pressure loss distribution are set in the `data_in/mesh.data` file. A tipical power distribution is reported in Figure 3.5. In order to have a consistent power distribution inside the assembly core the finite element discretization cannot cut the assembly geometry and therefore the coarse mesh must contain the geometrical pattern defined by the assemblies. 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
........
........
```

The file shows for each element: 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 loss factor. Since the direct editing is not very long then the file can be generated by a program that is enclosed in the directory `contrib/datagen`. The file should be completed for all mesh levels.

Program datagen for automatic power distribution

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

- delete the file `data_in/mesh.data`
- run the code with no `data_in/mesh.data` file. Then a new `data_in/mesh.data` is generated and the code stops.
- copy the file `data_in/mesh.data` in `contrib/datagen/data.in`
- edit `contrib/datagen/datagen.H` inserting power distribution factors as in Figure 3.5.
- run `datagen` program in the directory `contrib/datagen`.
- copy the file `contrib/datagen/data.in` in `data_in/mesh.data`

The power distribution in Figure 3.5 is enclosed with the provided package. The pressure loss distribution enclosed in the code is constant and equal to 1.

The file `contrib/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

```
// level
#define NLEV 1

// number of elements in the core section
#define NZC 9
```

```
// Geometry
// half fuel assembly length
#define HR   0.147
#define HIN  0.95
#define HOUT 1.85
```

The power distribution of Figure 3.5 over a plane must be reported in the double-indexed array `mat_pf` for a reactor that has assemblies only in the square 8×8. The area is divided in `INNER`, `INTER` and `OUTER` zone. The zone with no fuel `CTRLR` and the reflector area with `DUMMY` are written in the matrix `mat_zone`.

```
 // ---------------------------
// zone
// ---------------------------

#define INNER 0
#define INTER 1
#define OUTER 2
#define CTRLR 3
#define DUMMY 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 power factor over a plane 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 power factor over the z axis can be assumed to have a cosine 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 power distribution for the pressure loss must be reported in the matrix `axlpf` and vector `axllf`.

3.3 Step 2: Configuration, data and parameter setting

3.3.1 Configuration setting

Global configuration

In order to run a complete computation the following steps are necessary :
a) generate a hexahedral mesh with GAMBIT in the file `data_in/mesh.msh`;
b) generate the core power factor in the file `data_in/mesh.data` ;
c) Set the configuration file `config/config.h` and, if necessary, the additional class parameters for the active classes (for energy and Navier-Stokes the additional class parameters are in the files `config/class/MGSNSconf.h` and `config/class/MGSTconf.h`);
d) Set the physical and numerical properties in `config/parameters.in`;
e) Set the boundary and initial conditions in the files of the active classes (for energy and Navier-Stokes the files are `src/MGSolverNS3D.C` and `src/MGSolverT3D.C`).

Standard configuration is provided with the code and only few changes are necessary. In order to rerun a computation (mesh is provided) with different parameters only the steps c), d) and e) are necessary. Explanation of steps a) and b) can be found in the previous section.

The configuration file is `config/config.h`. The standard configuration for solving energy and Navier-Stokes equations is

```
// ============================
// BASIC SETTINGS
// ============================
// DIM2=TWO-DIM (UNDEF 3D) =====
//#define DIM2  1

// ============================
// EQUATIONS:        NS_EQUATIONS
//        T_EQUATIONS  TURBULENCE
```

```
//        TWO_PHASE TWO_EQUATIONS
// ==============================
// NAVIER-STOKES  EQUATIONS =====
#define NS_EQUATIONS   1
//#define KE_EQUATIONS   1
//#define KW_EQUATIONS   1
 ..........
// ==============================
// ENERGY EQUATION ==============
//#define T_EQUATIONS 1
 ...........
// ==============================
// RESTART GENERATION PRINT
// ==============================
//  RESTART OPTIONS =============
//#define RESTARTIME 0.1
//#define RESTART 100
// MESH GENERATION OPTIONS ======
  #define GENCASE 1
  #define LIBMESHF 1
  #define PRINT_INFO 1
// PRINT OPTIONS ================
//#define OUTGMV 1
  #define XDMF 1
//#define Vtk 1
  #define HDF5
 ............
```

Options are set via the `#define` command in C or C++. The option is not active when the corresponding `#define` is commented. For example if one sees in the file `config/config.h` a definition as

`# define NS_EQUATIONS 1`

then the Navier Stokes solver is active. If the `#define` option is with a comment (`//` is the comment operator in C++) as

`// # define NS_EQUATIONS 1`

then the Navier Stokes solver is not active and the Navier Stokes will not be solved. The number after the `#define` is necessary and sometimes such a number is used in the program for further options.

Many of the options in the file `config/config.h` are not important for our problems. They remain in the configuration file for compatibility with the CFD multipurpose code. The following options are important for this code:

- `DIM2`. With this option one can set the spatial dimension of the problem. For reactor problems one can use always 3D option, so the 2D option should be commented, i.e. `// # define DIM2 1`
- `NS_EQUATIONS, T_EQUATIONS`. If the system has the $(p, \mathbf{v})$ state then you must solve only the Navier-Stokes equations. If the system has the $(p, \mathbf{v}, T)$ state then you must

solve the Navier-Stokes equations and the energy equation. For the reactor we must set
```
# define NS_EQUATIONS 1
# define T_EQUATIONS 1
```

- `RESTARTIME, RESTART`. In order to restart your program from the solution at $t = 1.1$ written in the file `Out.10.vtu` one must set the restart option and the restart time, i.e.
```
#define RESTARTIME 1.1
#define RESTART 10
```
In order to start from zero ($t = 0.$) one must set
```
#define RESTARTIME 0.
// #define RESTART 10
```

- `GENCASE, LIBMESHF`. In order to avoid the use of the LIBMESH library functions all the necessary files are given to you. In this case you must set
```
// #define GENCASE 1
// #define LIBMESHF 1 .
```
In order to generate a new geometry and therefore a new case of files one must set
```
#define GENCASE 1
#define LIBMESHF 1 .
```
This option needs the LIBMESH library (see http://libmesh.sourceforge.net for details).

- `GMV, XDMF, VTK, HDF5`. This sets all the output files in the desired format [15, 14, 16, 13]. For GMV format see http://laws.lanl.gov/XCM/gmv/GMVHome.html, for XDMF format see http://www.xdmf.org, for VTK format see http://www.vtk.org and for HDF5 format see http://www.hdfgroup.org/HDF5/. For visualization with PARAVIEW one must set only the XDMF and the HDF5 options
```
//#define OUTGMV 1
#define XDMF 1
//#define Vtk 1
#define HDF5
```

- `PRINT_INFO`. With this option the program prints a message from each routine. It is useful for debugging. Let set
```
#define PRINT_INFO 1
```
for printing only messages from routines.

Class parameter configuration

In the directory `config/class` there is a configuration file for each class (Navier-Stokes equation, energy equation, $\kappa - \epsilon$ turbulence equation). All the parameters that are needed in the class model can be changed in these files. The most important ones are the dependence of the physical properties on the temperature, the turbulence parameters and the numerical solver algorithm.

Dependence of the physical properties on the temperature. The code can run with lead properties that can be considered as a function of temperature. These functions are

defined directly in the files where they are used. If the property law must be modified it is necessary to change the inline functions at the top of the followings:
- `config/class/MGSNSconf.h` for the momentum equations; here the functions $\rho = \rho(T)$ and $\nu = \nu(T)$ are defined.
- `config/class/MGSTconf.C` for the energy equation; here the functions $\rho = \rho(T)$, $\kappa = \kappa(T)$ and $C_p = C_p(T)$ are defined.
Furthermore, you have to set
`// #define CONST 1`
in the same files in order to activate the temperature dependence.

Turbulence parameters. The parameters for the $\kappa-\epsilon$ and $\kappa-\omega$ turbulence model 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.

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
8.  GMRES(10) = GMRESIte
9.  BiCG = BiCGIter
10.  QMR = QMRIter
11.  CGS = CGSIter
12.  Bi-CGSTAB = BiCGSTABIter
13.  Test = TestIter .
```

Among the preconditioner matrices we can have

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

Standard configuration uses GMRES and ILU preconditioner.

3.3.2 Parameter setting

The parameter file is `config/parameters.in`. An example is

```
 # NonDimensionalTimeStep
dt        .2
itime      0.0
nsteps     100
printstep  1
```

```
# Reference ValueForNon-dimensionalization
Uref      1.
Lref      1.
Tref      1.

# FluidProperties
rho0      10562.99
mu0       0.0022
kappa0    16.58
cp0       148.77
komp0     0.

# HeatSource
qheat 1.14591e+8

# Gravity
dirgx  0.
dirgy  0.
dirgz  0.
```

The data values are defined by setting the value on the right of each item. For example if we see in the file `config/parameters.in` a definition such as
`rho0 10000`
then the quantity `rho0` takes the value of 10000. The units are always in agreement with the International System.

parameter	value	description
`dt`	0.2	Time step Δt
`itime`	.0	Initial time t
`nsteps`	100	Number of time steps
`printstep`	5	print every 5 time steps
`Uref`	1.	Reference velocity
`Lref`	1.	Reference length
`Tref`	1.	Reference temperature
`rho0`	10562.99	density
`mu0`	.0022	viscosity
`kappa0`	16.58	conductivity
`cp0`	147.3	heat capacity
`komp0`	0.	compressibility
`qheat`	1.14591e+8	heat volume density source
`dirgx`	0.	gravity x-direction
`dirgy`	0.	gravity y-direction
`dirgz`	0.	gravity z-direction

Table 3.1: Parameters in `config/parameters.in` file

The parameters to set from this file are as follows:

- `dt,itime,nsteps,printstep`. These are the values of the time step, the initial time, the number of time steps and the print timestep interval. In order to start at $t = 0$, perform 100 time steps of length $\Delta t = 0.01$ and print every time step one must set
  `dt 0.01`
  `itime 0.0`
  `nsteps 100`
  `printstep 1.`

- `Uref,Lref,Tref`. These are the reference values of velocity, length and temperature.

- `rho0,mu0,kappa0,cp0`. Those are the reference values of the lead density, viscosity, thermal conductivity and of the constant-pressure specific heat, respectively. In our case we set
  `rho0 10562.99`
  `mu0 .0022`
  `kappa0 15.8`
  `cp0 147.3`
  The temperature dependence is set inside the single files: `config/MGSTconf.h` (energy equation) and `config/MGSolNSconf.h` (Navier-Stokes system). For details see Section 3.3.1.

- `qheat`. The average volumetric heat source for the reactor is set by this parameter as
  `qheat 1.15e+8`.
  The sinusoidal shape and other source factors are set directly inside the file `MGSTconf.h`. The power factor distribution of the assemblies is in the file `mesh_data` in the directory `data_in`. See Section 3.2.2.

- `dirgx, dirgy, dirgz`. Gravity can be set by using these parameters. If one wants gravity ($\rho \mathbf{g}$) along the z-axis one must set
  `dirgx 0.`
  `dirgy 0.`
  `dirgz -1.`
  All the value are in $g = 9.81\, m/s^2$ units.

3.3.3 Boundary and initial conditions

Boundary conditions

The boundary conditions are already set for the reactor. In order to change the boundary conditions, it is necessary to edit the user part in the appropriate function. For pressure and velocity boundary conditions one must edit the function
`void MGSolNS::bc_read(double xp[],double normal [],int bc_flag[])`
in the file `src/MGSolverNS3D.C`. If you open the mentioned file you may find

```
void MGSolNS::bc_read(double xp[],double normal [],int bc_flag[]) {
#ifdef DIM2
#else
```

```
// 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;
 }
#endif
  return;
}
```

This function sets Dirichlet boundary conditions over a box of dimension $[0,1] \times [0,1] \times [0,1]$ The node coordinates `(xp[0],xp[1],xp[2])` can be used to set all the necessary boundary conditions. The value `bc_flag[0]` is the flag for the boundary condition on the x-component of the velocity field. If the flag is not set the point has Neumann conditions by default. By setting `bc_flag[0]=0` one imposes a Dirichlet boundary condition. The `bc_flag[1]` and `bc_flag[2]` are the flags for the y and z-component 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[],double normal [],int bc_flag[])`
in the file `src/MGSolverT3D.C`. If you open the mentioned file you may find

```
 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 points are provided also for this equation.

Initial conditions

Initial pressure velocity solution. If one wants to set the initial solution in pressure and velocity it is necessary to edit the function
`void MGSol::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 over the z-axis. Therefore in the appropriate part of the function you have

```
void MGSolNS::ic_read(double xp[],double u_value[]) {
#ifdef DIM2
#else
  // 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].;
```

Initial Energy solution. 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`. In the user area of the file `MGSolverT3D.C` you will find

```
void MGSolNT::ic_read(double xp[],double u_value[]) {
#ifdef DIM2
#else
  // xp[]=(xp,yp,zp) u_value[]=(u,v,w,p)
  u_value[0] =400.;
```

In this case we set uniform temperature to 400 for $t = 0$.

3.4 Step 3: Analysis of the CFD solution

The results can be analyzed by using the PARAVIEW software. The solution at each time step is read by using the XDMF format and the data are stored in HDF5 format.

3.4.1 Output format in HDF5 and XDMF format

The HDF5 format

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. For detailed information one can see http://www.hdfgroup.org/HDF5/. HDF5 format supports all types of data stored digitally, regardless of origin or size. The HDF5 technology is designed to organize, store, discover, access, analyze, share, and preserve diverse, complex data in continuously evolving heterogeneous computing and storage

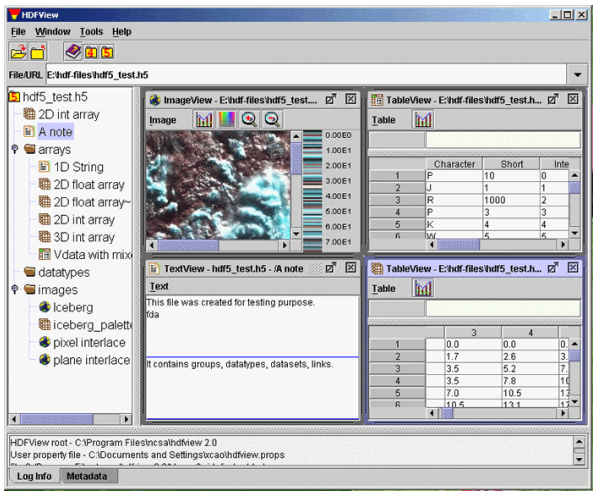

Figure 3.6: HDF5View for raw HDF5 data

environments. The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing and analyzing data in the HDF5 format.

In our code we have the following HDF5 files:

- `mesh.h5` and `meshxmf.h5`, with mesh points and connectivity;
- `sol.*.h5`, with field data over points and cells;
- `case.*.h5`, with boundary and initial conditions.

Inside these files the data are organized in directories or datasets. The `mesh.h5` file has the following datasets:

- CONNl the connectivity of the mesh at the level l for quadratic finite elements (e.g. HEX27);
- X the x-coordinates of the mesh;
- Y the y-coordinates of the mesh;
- Z the z-coordinates of the mesh;

The `meshxmf.h5` file has the following datasets:

- CONN the connectivity of the mesh at the top level for elements that can be visualized by PARAVIEW;

The connectivity for parabolic finite elements cannot be visualized directly inside PARAVIEW and therefore two connectivities must be used. For example the HEX27 cannot be seen in PARAVIEW and therefore must be converted to HEX8. The `sol.*.h5` file for the reactor code has the following datasets:

- `u1` the x-component of the velocity field;;
- `u2` the y-component of the velocity field;

- `u3` the z-component of the velocity field;
- `u4` the pressure field;
- `T` temperature field;

If other fields are solved these variables appear in the `sol.*.h5` files. The `case.*.h5` file for the reactor code has the following datasets:

- `u1` initial condition for the x-component of the velocity field;;
- `u2` initial condition for the y-component of the velocity field;
- `u3` initial condition for the z-component of the velocity field;
- `u4` initial condition for the pressure field;
- `T` initial condition for the temperature field;
- `u1bd` boundary condition for the x-component of the velocity field;;
- `u2bd` boundary condition for the y-component of the velocity field;
- `u3bd` boundary condition for the z-component of the velocity field;
- `u4bd` boundary condition for the pressure field;
- `Tbd` boundary condition for temperature field;

The raw data in HDF5 format can be seen by the HDFView software (see Figure 3.6). HDFView can be downloaded free of charge from
http://www.hdfgroup.org/HDF5/.
The command `hdf5view` starts the GUI of the viewer. The user guide can be found at
http://www.hdfgroup.org/hdf-java-html/hdfview/UsersGuide/.

The eXtensible Data Model (XDMF) Format

The need for a standardized method to exchange scientific data between High Performance Computing codes and tools lead to the development of the eXtensible Data Model and Format (XDMF). The XDMF format categorizes data by two main attributes: size and function. Data can be Light (typically less than about a thousand values) or Heavy (megabytes, terabytes, etc.). In addition to raw values, data can refer to Format (rank and dimensions of an array) or Model (how that data is to be used, i.e. XYZ coordinates vs. Vector components). XDMF uses XML to store Light data and to choose the Model. HDF5 is used to store Heavy data. The data Format is stored redundantly in both XML and HDF5. This allows tools to parse XML to determine the resources that will be required to access the Heavy data.

The eXtensible Markup Language (XML) format is widely used for many purposes and is well documented in many sites. See for example http://www.xdmf.org. There are numerous open source parsers available for XML. The XDMF API takes advantage of the `libxml2` parser to provide the necessary functionality. Without going into too much detail, XDMF views XML as a "personalized HTML" with some special rules. It it case sensitive and is made of three major components : elements, entities, and processing information. In the XDMF format we are primarily concerned with the elements. These elements follow the basic form:

Attribute (XdmfAttribute)	
Name	(no default)
AttributeType	Scalar, Vector, Tensor, Tensor6, Matrix, GlobalID
Center	Node, Cell, Grid, Face, Edge
DataItem (XdmfDataItem)	
Name	(no default)
ItemType	Uniform, Collection, tree, HyperSlab, coordinates — Function
Dimensions	(no default) in KJI Order
NumberType	Float, Int, UInt, Char, UChar
Precision	1, 4, 8
Format	XML, HDF
Domain (XdmfDomain)	
Name	(no default)
Geometry (XdmfGeometry)	
GeometryType	XYZ, XY, X_Y_Z, VxVyVz, Origin_DxDyDz
Grid (XdmfGrid)	
Name	(no default)
GridType	Uniform, Collection, Tree, Subset
CollectionType	Spatial, Temporal (if GridType="Collection")
Section	DataItem, All (if GridType="Subset")
Topology (XdmfTopology)	
Name	(no default)
TopologyType	Polyvertex Polyline , Polygon , Triangle , Quadrilateral , Tetrahedron, Pyramid, Wedge, Edge_3, Triagle_6, Quadrilateral_8, Tetrahedron_10, Wedge_15, Hexahedron_20, Hexahedron, Pyramid_13, Mixed, 2DSMesh, 2DRectMesh, 2DCoRectMesh, 3DSMesh, 3DRectMesh
NodesPerElement	(no default)
NumberOfElement	(no default)
OR	
Dimensions	(no default)
Order	each cell type has its own default
BaseOffset	0, #
Time	
TimeType	Single, HyperSlab, List, Range
Value	(no default, Only valid for TimeType="Single")

Table 3.2: XML Element (Xdmf ClassName) and Default XML Attributes

```
<ElementTag
 AttributeName="AttributeValue"
 AttributeName="AttributeValue"
 ... >
 CData
</ElementTag>
```

Each element begins with a `<tag>` and ends with a `</tag>`. Optionally there can be several `"Name=Value"` pairs which convey additional information. Between the `<tag>` and the `</tag>` there can be other `<tag></tag>` pairs and/or character data (CData). CData is typically where the values are stored; like the actual text in an HTML document. The XML parser in the XDMF API parses the XML file and builds a tree structure in memory to describe its contents. This tree can be queried, modified, and then "serialized" back into XML. The organization of XDMF begins with the XDMF element. So that parsers can distinguish from previous versions of XDMF , there exists a Version attribute (currently at 2.0). Any element in XDMF can have a Name attribute or have a Reference attribute. The Name attribute becomes important for grids while the Reference attribute is used to take advantage of the XPath facility (more detail on this later).

Some of the most important XDMF elements are the following:

- Domain. A Domain can have one or more Grid elements. Each Grid contains a Topology, Geometry, and zero or more Attribute elements. Topology specifies the connectivity of the grid while Geometry specifies the location of the grid nodes. Attribute elements are used to specify values such as scalars and vectors that are located at the node, edge, face, cell center, or grid center. To specify actual values for connectivity, geometry or attributes, XDMF defines a DataItem element. A DataItem can provide the actual values or provide the physical storage (which is typically an HDF5 file).

- Grid. The DataItem element is used to define the data format portion of XDMF. It is sufficient to specify fairly complex data structures in a portable manner. The data model portion of XDMF begins with the Grid element. A Grid is a container for information related to 2D and 3D points, structured or unstructured connectivity, and assigned values.

- Topology. The Topology element describes the general organization of the data. This is the part of the computational grid that is invariant with rotation, translation, and scale. For structured grids, the connectivity is implicit. For unstructured grids, if the connectivity differs from the standard, an Order may be specified.

- Geometry. The Geometry element describes the XYZ values of the mesh. The important attribute here is the organization of the points. The default is XYZ; an X,Y, and Z for each point starting at parametric index 0. Possible organizations are: interlaced arrays (XYZ) or separate arrays (X_Y_Z).

- Attribute. The Attribute element defines values associated with the mesh. Currently the supported types of values are: Scalar, Vector and Tensor (9 values expected). These values can be centered on Node, Edge, Face or Cell.

A more detailed explanation of the XDMF elements can be found in Table 3.2. In the output files we have the following XDMF files:

- `mesh.xmf` for reading mesh points and connectivity;
- `sol.*.xmf` for reading field data over points and cells;
- `case.*.xmf` for reading initial and boundary conditions.

The heavy data for the XDMF files are the files in HDF5 format discussed in the previous section.

3.4.2 PARAVIEW

Visualization with PARAVIEW

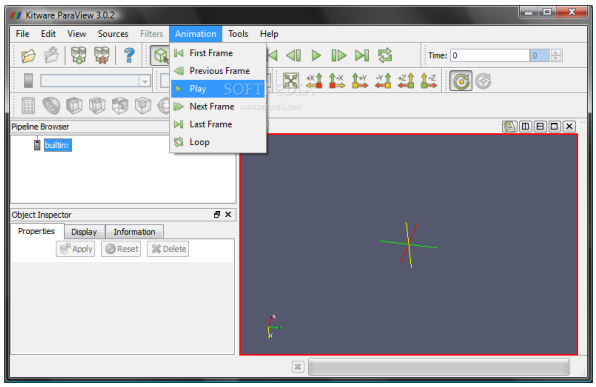

Figure 3.7: PARAVIEW

PARAVIEW is an open-source, multi-platform data analysis and visualization application. The PARAVIEW software can be downloaded from `http://www.paraview.org`. PARAVIEW can build visualizations to analyze data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using PARAVIEW batch processing capabilities. The PARAVIEW code base is designed in such a way that all of its components can be reused to quickly develop other applications.

The program starts with the command `paraview` as in Figure 3.7. PARAVIEW can read two types of files produced by the reactor code

- files with extension `xmf` that contain single time solution;
- files with extention `pvd` that contain long time solution.

Data post-processing

PARAVIEW can manage the results through functions that use the stored data. Very useful are the calculator function, the integration and the differentiation functions.

Using the calculator. In order to compute quantities such as temperature on the fuel cladding we can use analytical expressions. The function calculator can be found in the menu path
`Filters -> Alphabetical -> Calculator`.
The analytical expression can be written in the line as shown in Figure 3.8 by using the

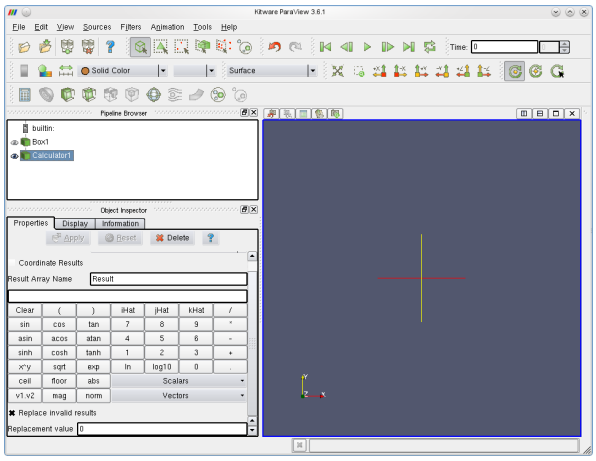

Figure 3.8: PARAVIEW calculator

Vectors and Scalars read from the data files.

Integration and differentiation of post-processing data. Integration and differentiation of of all scalar and vector solution variables can be done by following the menu paths `Filters -> Alphabetical -> Compute Derivatives` and `Filters -> Alphabetical -> Integrate Variables` respectively.

Chapter 4

CFD reactor simulation

4.1 Introduction

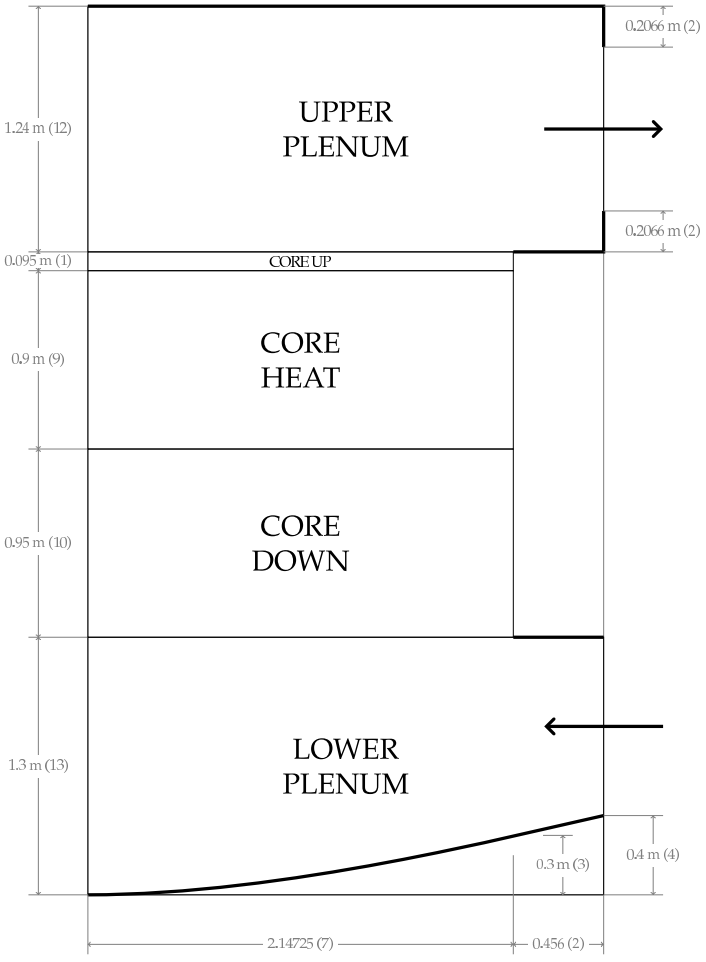

Figure 4.1: Vertical section of the reactor.

The design of the reactor starts from the horizontal quarter section of the core shown in Figure 4.2. Each fuel assembly consists of a 17×17 pin lattice. The distribution of the

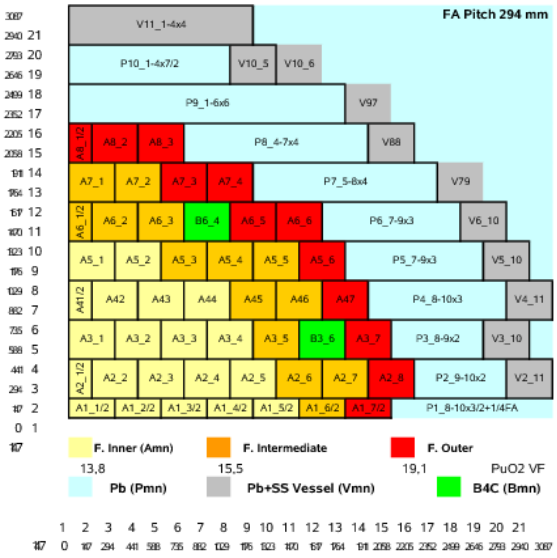

Figure 4.2: Horizontal section of the reactor with fuel power factors.

170 assemblies results in an approximately circular arrangement fitting the required core area. 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. The side of a square assembly (LFA) is $0.294\,m$ in working conditions at the temperature of $673.15\,K$ (400 ℃). We are interested in the active (upper) and non-active (lower) core sections and in the upper and lower plena. In our computational description we consider the core region from $0\,m$ to $1.85\,m$ where the active core (upper core) 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 outlet. The heat generation zone or the active core zone for the reactor starts at $H_{in} = 0.95\,m$ and ends at $H_{out} = 1.85\,m$. The reactor is cooled by lead that enters at the temperature of 400 ℃. Since our model describes the reactor at the assembly level the sub-assembly composition is seen as a homogeneous medium. Data about this model composition are reported in Table 4.1. In particular we note that the coolant/assembly ratio is 0.5408. Each assembly has a square section with the side length of $L = 0.294\,m$, as shown in Table 4.2 and this completely defines the horizontal core structure. For the vertical geometry we refer to Figure 4.1. In Table 4.3 we report the physical properties (at 400 ℃) inserted in the code.

	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 4.1: Coolant assembly area ratio data

		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 4.2: Core characteristic values at working temperature

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 4.3: Lead properties at T=400 °C

Let us consider the core region only. In steady working conditions with constant properties (see Table 4.3) and constant velocity the (1.79-1.81) become

a) **Incompressibility constraint**

$$\frac{\partial \widehat{w}_h}{\partial z} = 0 \tag{4.1}$$

b) **Momentum equation**

$$\frac{\partial \widehat{p}_h}{\partial z} = \frac{2\rho \widehat{w}^2}{D_{eq}} \lambda \tag{4.2}$$

b) **Energy equation**

$$\rho \, C_p \widehat{w}_h \, \frac{\partial \widehat{T}_h}{\partial z} = \kappa \, \frac{\partial^2 \widehat{T}_h}{\partial z^2} + \dot{q}^{\,\prime\prime\prime} \, r \, . \tag{4.3}$$

These equations can be solved and therefore this solution can be used to check the order of magnitude of the numerical solution.

The problem (4.1-4.3) can be further simplified for some velocity ranges for which the thermal conductivity κ can be neglected. If we assume $\kappa = 0$ the system, after integration

along the z-axis, yields

$$\widehat{w}_h = const \tag{4.4}$$

$$\widehat{p}_{in} - \widehat{p}_{out} = \frac{2\rho\widehat{w}^2}{D_{eq}}\lambda(\widehat{w}_h)\, H \tag{4.5}$$

$$\rho\, C_p \widehat{w}_h\, (\widehat{T}_{out} - \widehat{T}_{in}) = \dot{Q}\, r\, (H_{out} - H_{in})\,, \tag{4.6}$$

where H is the total height of the reactor, H_{in} and H_{out} are the starting and ending point of the heat generation. In order to compute $(\widehat{w}_h, \widehat{p}_h, \widehat{T}_h)$ we set each property to be constant (see Table 4.3).

Velocity. The velocity can be solved as

$$\widehat{w}_h = \frac{\dot{m}}{\rho\, r\, A\, N} = \frac{124539}{10562 \times 0.5408 \times 0.294 \times 0.294 \times 170} = 1.484\, m/s\,. \tag{4.7}$$

The volumetric flow rate for assembly

$$\dot{V} = \frac{\dot{m}}{\rho\, N} = \frac{124539}{10562 \times 170} = 0.069360\, m^3/s\,. \tag{4.8}$$

The volumetric flow rate for assembly per unit surface is

$$\dot{V} = \frac{\dot{m}}{\rho\, A\, N} = \frac{124539}{10562 \times 0.294 \times 0.294 \times 170} = 0.8024\, m/s\,. \tag{4.9}$$

The Reynolds number is

$$Re = \frac{\rho\, \widehat{w}_h\, D_{eq}}{\mu} = 10562 \times 1.484 \times 0.0129/0.0022 = 91906 \tag{4.10}$$

Pressure. The friction coefficient is

$$\lambda = \frac{0.079}{Re^{0.25}} = \frac{0.079}{91906^{0.25}} = 0.004537\,. \tag{4.11}$$

Then the pressure jump can be computed as

$$\widehat{p}_{hin} - \widehat{p}_{hout} = \frac{2\rho\widehat{w}^2\, H\, \lambda(\widehat{w}_h)}{D_{eq}} = \frac{2 \times 10562 \times 1.484 \times 1.484 \times 1.945 \times 0.004537}{0.0129} = 32000Pa$$

Temperature. The temperature jump can be computed as

$$\widehat{T}_{hout} - \widehat{T}_{hin} = \frac{\dot{Q}}{\dot{m}\, c_P\, (H_{out} - H_{in})} = \frac{1.482 \times 10^9}{124539 \times 124539 \times 147.3 \times 0.9} = 89.7\, K$$

At the top of the reactor we obtain a constant temperature distribution with value

$$\widehat{T}_{hout} = \widehat{T}_{hin} + 89.7 = 762.85\, K$$

These values obtained with many assumptions can be considered as first benchmarks for the computational results. The simulation of the reactor follows step by step the User Guide in 3. The process consists of three steps:

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

4.2 Step 1. Preprocessing: mesh and data generation

Mesh generation

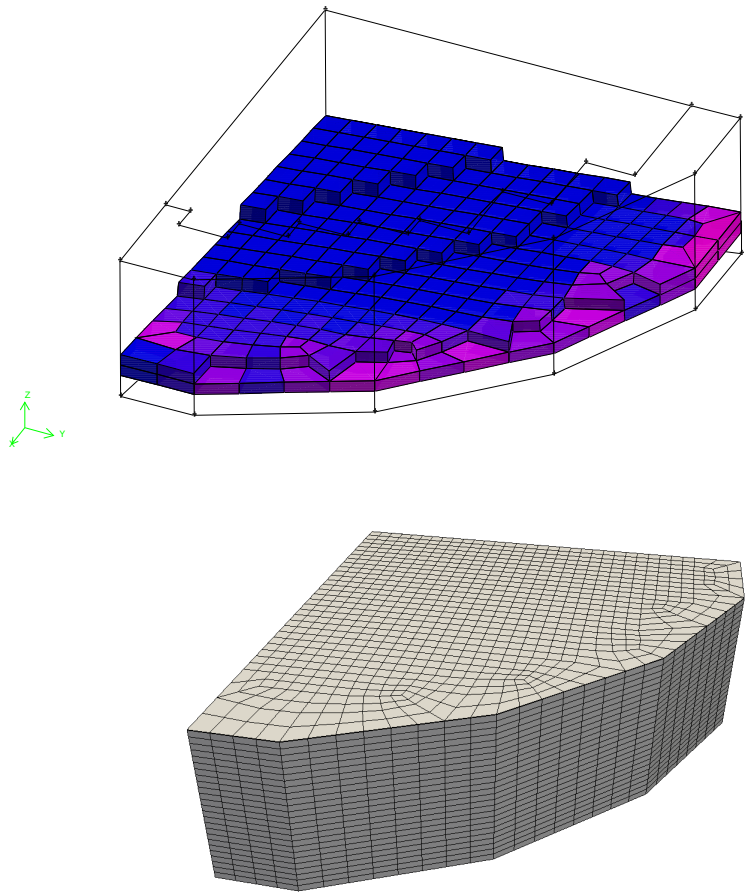

Figure 4.3: Reactor lower plenum

The mesh is generated by using the GAMBIT mesh generator with geometrical dimensions taken by Figures4.1 and 4.2. The mesh should match the assembly geometry in the core. In order to obtain this, we first project the core assembly structure on the top of the reactor lower plenum obtaining Figure 4.3. The top of the lower plenum is set over the plane $z = 0$. The inlet is distributed over all the perimeter of the reactor with a height of $0.9m$. The lower point is set at $1.2m$ below the top of the lower plenum. A simple projection over the core assembly structure gives Figure 4.4. The axial length of the core is $1.8m$. The core consists of two parts with equal length: the top and bottom parts. The top part is the nuclear core where the heat is generated. The upper plenum and the complete reactor are shown in Figures 4.5-4.6. The GAMBIT general option should be set to `Generic` and the mesh export should be

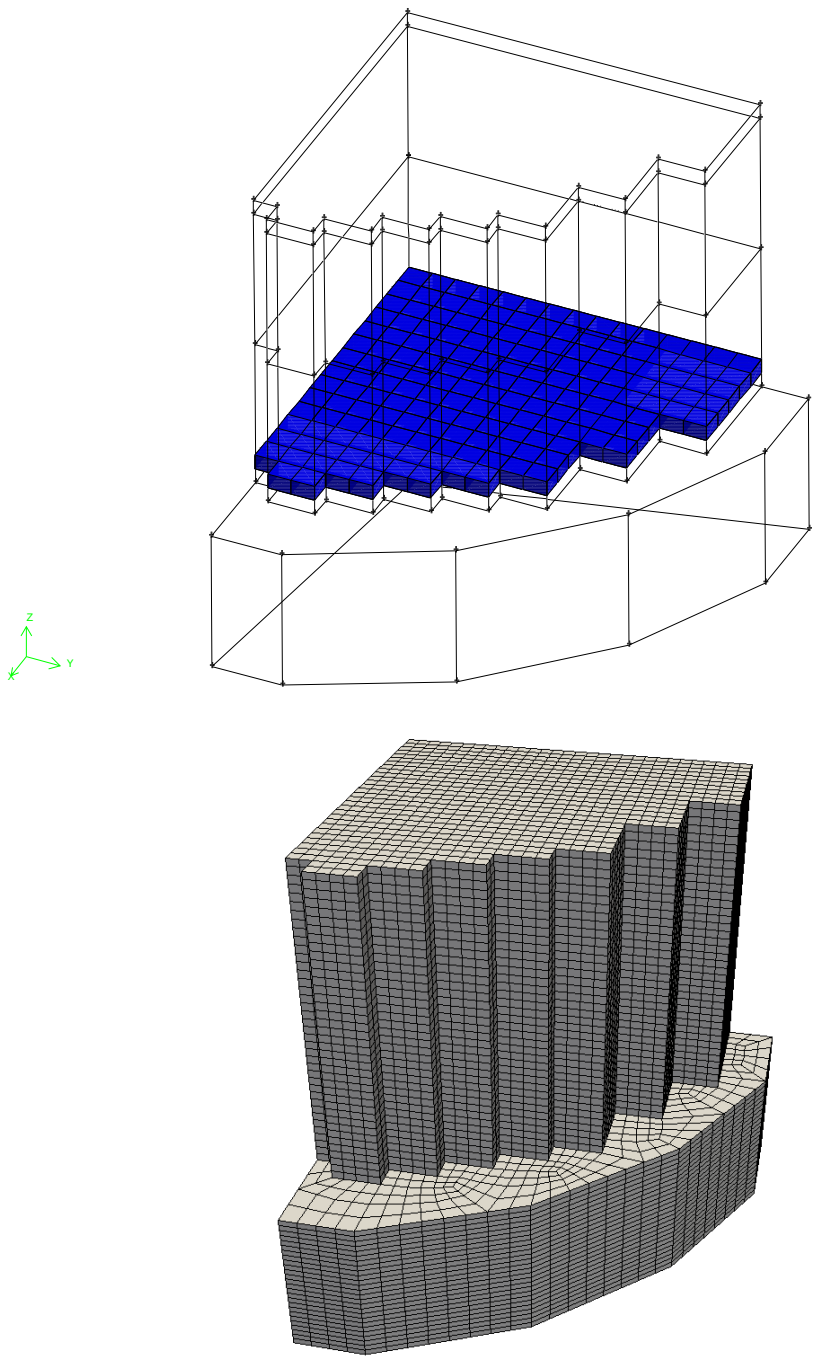

Figure 4.4: Reactor core and lower plenum

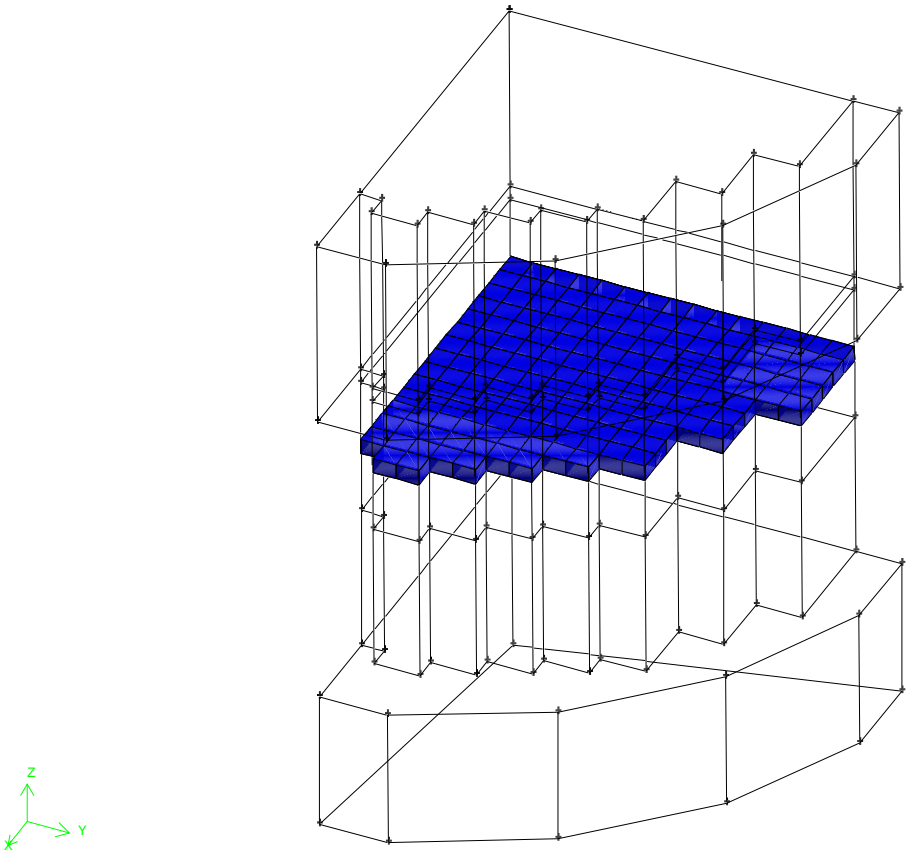

Figure 4.5: Reactor design

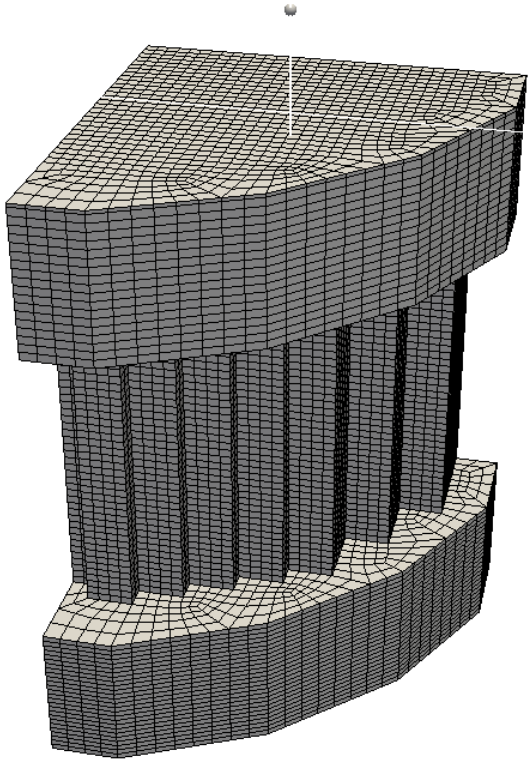

Figure 4.6: Reactor mesh

performed under the `Mesh` option. We rename this file `mesh.msh` and copy it in the directory `data_in`.

Data generation

In this case we consider the heat power in the form

$$\dot{Q}(x, y, z) = W_0(x, y) \cos\left(\frac{2\pi(z - H_{in})}{H_{out} - H_{in}}\right) \tag{4.12}$$

where $W_0(x, y)$ is the two-dimensional distribution at $(H_{out} - H_{in})/2$. In the numerical model we assume constant value within each element. We assume a (x, y)-distribution of the type shown on the top of Figure 4.7. The procedure needed to load this power distribution in the code is as follows:

- delete the file `data_in/mesh.data`
- run the code with no `data_in/mesh.data` file. Then a new `data_in/mesh.data` is generated and the code stops.

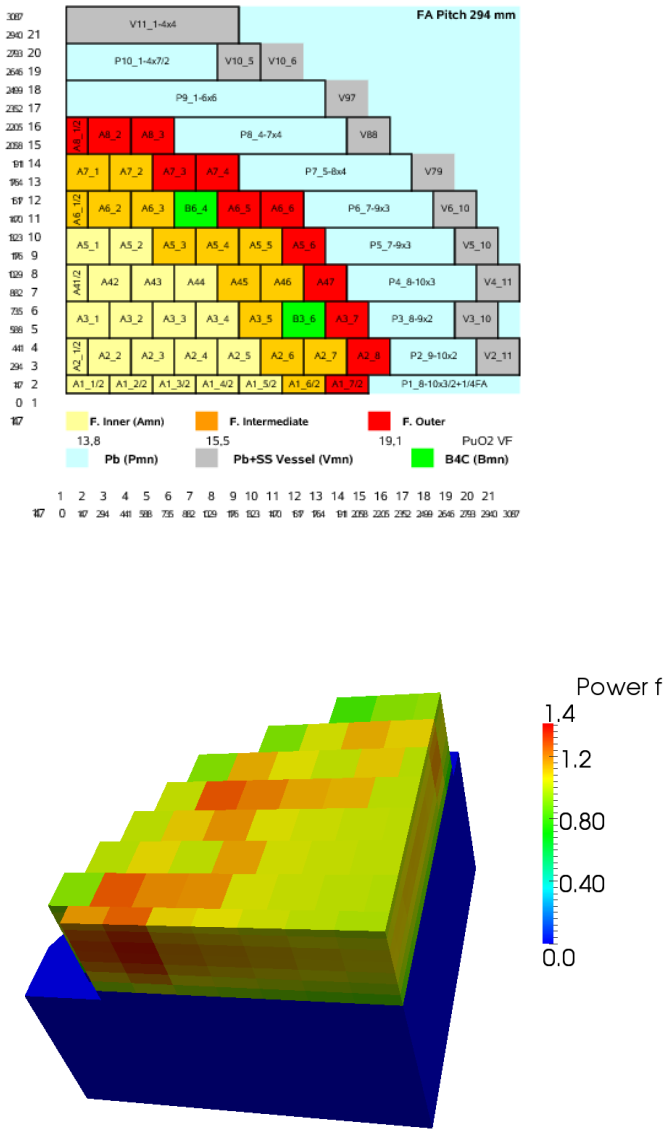

Figure 4.7: Horizontal power generation profile: code model (bottom) and desired (top)

- copy the file `data_in/mesh.data` in `contrib/datagen/data.in`
- edit `contrib/datagen/datagen.H` inserting power distribution factors as in Figure 3.5.
- run the `datagen` executable in the directory `contrib/datagen`.
- copy the file `data_in/data.in` in `data_in/mesh.data`

The pressure loss distribution is set to a constant value of 1.

The file `contrib/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 (see Section 3.2.2)

```
// level
#define NLEV 1

// number of elements in the core section
#define NZC 9

// Geometry
// half fuel assembly length
#define HR   0.147
#define HIN  0.95
#define HOUT 1.85
```

The power distribution of Figure 3.5 over a plane must be reported in the double-indexed array `mat_pf` for a reactor that has assemblies only in the square 8×8. The area is divided into `INNER`, `INTER` and `OUTER` zone. The zone with no fuel `CTRLR` and the reflector area with `DUMMY` are written in the matrix `mat_zone`.

```
 // --------------------------
// zone
// --------------------------

#define INNER 0
#define INTER 1
#define OUTER 2
#define CTRLR 3
#define DUMMY 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 power factor over a plane is reported in `mat_pf`

```
#define CTL_R 1.
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
```

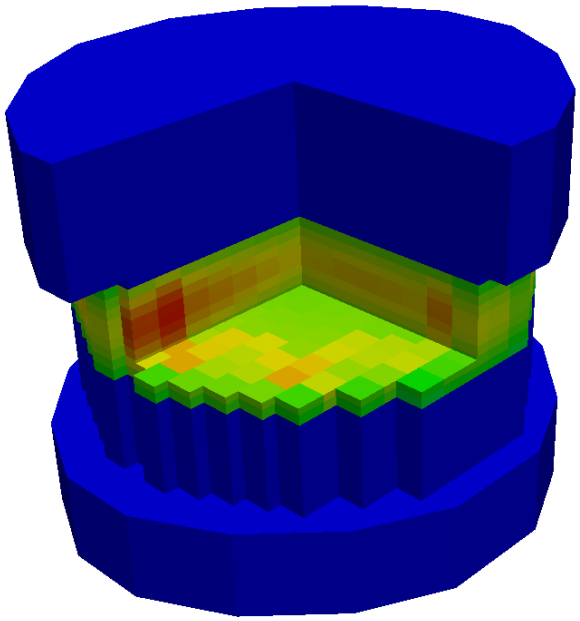

Figure 4.8: Vertical power generation profile in the code model

```
{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
};
```

We note that in this case the `CTRLR` is set to 1 and not to zero as in Figure 4.7. The power factor over the z- axis can be assumed to have a cosine 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}
};
```

The power distribution can be seen from the file `case0.xmf` (once the code is run) by using PARAVIEW. Once the file `case0.xmf` is open under PARAVIEW, the following path `Filters`

`-> Clip`
generates a slice and the selection
`Select data1`
shows the bottom of Figure 4.7. The Figure 4.7 shows the horizontal profile and Figure 4.8 the vertical profile of the reactor where the cosine approximation can be seen.

4.3 Step 2: Configuration, data and parameter setting

4.3.1 Configuration setting

Global configuration

Standard configuration is provided with the code and only few changes are necessary. We follow the steps defined in 3.3.

The configuration file is `config/config.h`. The standard configuration for solving energy and Navier-Stokes equations is

```
// ============================
// BASIC SETTINGS
// ============================
// DIM2=TWO-DIM (UNDEF 3D) =====
//#define DIM2  1

// ============================
// EQUATIONS:        NS_EQUATIONS
//        T_EQUATIONS  TURBULENCE
//        TWO_PHASE TWO_EQUATIONS
// ============================
// NAVIER-STOKES  EQUATIONS =====
#define NS_EQUATIONS   1
 ..........
// ============================
// ENERGY EQUATION ==============
#define T_EQUATIONS 1
 ...........
// ============================
// RESTART GENERATION PRINT
// ============================
//  RESTART OPTIONS =============
//#define RESTARTIME 0.1
//#define RESTART 100
// MESH GENERATION OPTIONS ======
  #define GENCASE 1
  #define LIBMESHF 1
  #define PRINT_INFO 1
// PRINT OPTIONS ================
//#define OUTGMV 1
  #define XDMF 1
```

```
//#define Vtk 1
  #define HDF5
 ............
```

In this case we use the Navier-Stokes equations and the energy equation but not the $\kappa - \epsilon$ or $\kappa - \omega$ system. A simple LES turbulence model is used in this computation. The LIBMESH library is used to generate the operators and the output files are in XDMF and HDF5 formats.

Class parameter configuration

In the directory `config/class` there is a configuration file for each class (Navier-Stokes equations, energy equation, $\kappa - \epsilon$ turbulence equation). We need to consider the parameters inside the files `config/MGSNSconf.h` and `config/MGSTconf.h`.

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 [3]. We set the LES option with $\alpha = 1$ (`#define ALPHA 1`). This option sets a simple Smagorinsky LES turbulence model. Under the standard setting the system will be solved by using GMRES method with an ILU preconditioner.
MGSTconf.h. We set `#define CONST 1` in order to neglect the temperature dependence of density and viscosity. This option is acceptable for a first computation [3]. We set the LES option with $Pr_t = 0.9$ (`#define PRT 0.9`). This sets the standard heat exchange model. Under the standard setting the system will be solved by using GMRES method with an ILU preconditioner.

4.3.2 Parameter setting

The parameter file is `config/parameters.in`. We set the data file as below

```
 # NonDimensionalTimeStep
dt        .002
itime      0.0
nsteps     10000
printstep  50

# Reference ValueForNon-dimensionalization
Uref      1.
Lref      1.
Tref      1.

# FluidProperties
rho0      10562.99
mu0       0.0022
kappa0    16.58
cp0       148.77
komp0     0.

# HeatSource
```

```
qheat 1.14591e+8

# Gravity
dirgx  0.
dirgy  0.
dirgz  0.
```

The non-dimensional time step is set to .002 and the simulation stops after 10000 steps. The solution is printed every 50 time steps. Following Table 4.3, the physical quantities are set as: density $rho0 = 10562.99$, viscosity $mu0 = 0.0022$, thermal conductivity $kappa0 = 16.58$ and heat capacity $cp0 = 148.77$. The average heat source per assembly is computed as

$$q_v = \frac{\dot{Q}}{170} = \frac{1.482 \times 10^9}{170 \times 0.294 \times 0.294 \times 0.9} = 0.11206 \times 10^8 \tag{4.13}$$

and therefore we set $qheat = 0.11206 \times 10^8$. In this case we considered 170 fuel assemblies neglecting the control rod assemblies.

4.3.3 Boundary and initial conditions

Boundary conditions

The boundary conditions are rather complex due to the reactor geometry. In order to change the boundary conditions, it is necessary to edit the user part in the appropriate function. For pressure and velocity boundary conditions one must edit the function
`void MGSolNS::bc_read(double xp[],double normal [],int bc_flag[])`
in the file `src/MGSolverNS3D.C`. We write

```
#define TOL 0.0001
void MGSolNS::bc_read(double xp[],double normal [],int bc_flag[]) {
#ifdef DIM2
#else
// xp[]=(xp,yp) bc_flag[u,v,p]=0-Dirichlet 1-Neumann

 if(xp[1]<TOL) bc_flag[1]=0; // symmetry y
 if(xp[0]<TOL) bc_flag[0]=0; // symmetry x
 if(xp[2]<1.945+TOL && xp[2]>-TOL && xp[0]>TOL && xp[1]>TOL){
    bc_flag[1]=0; bc_flag[0]=0;// core
 }
 if(xp[2]<1.945+TOL && xp[2]>-TOL && xp[0]>2.058-TOL && xp[1]>-TOL){
    bc_flag[0]=0; bc_flag[1]=0;// edge
 }
 if(xp[2]<1.945+TOL && xp[2]>-TOL && xp[1]>2.205-TOL && xp[0]>-TOL){
    bc_flag[0]=0; bc_flag[1]=0;// edge
 }
 if(xp[2]<1.945+TOL && xp[2]>1.945-TOL && xp[0]<1.47+TOL  &&
    xp[0]>1.176-TOL && xp[1]<0.735+TOL && xp[1]>0.441-TOL) {
    bc_flag[0]=1; bc_flag[1]=1; bc_flag[2]=1;// control rod
```

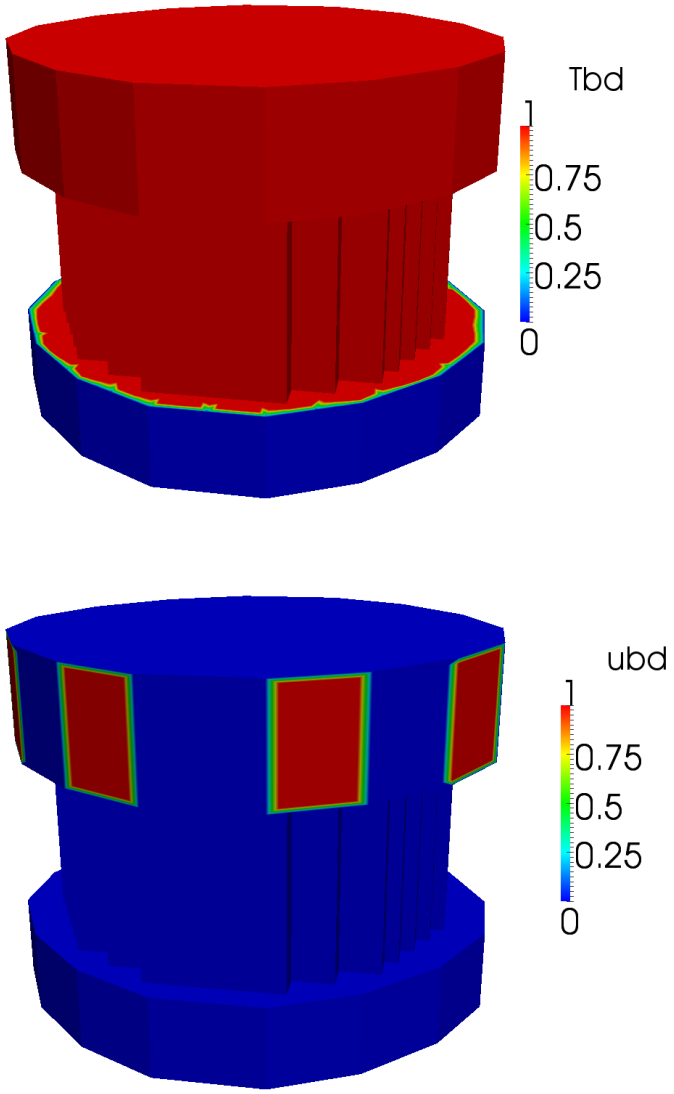

Figure 4.9: Boundary condition for the variables T and u: Dirichlet (0) and Neumann (1).

```
}
if(xp[2]<TOL && xp[2]>-TOL && xp[0]<1.47+TOL  && xp[0]>1.176-TOL
              && xp[1] < 0.735+TOL  && xp[1] >0.441-TOL) {
   bc_flag[0]=1; bc_flag[1]=1; bc_flag[2]=1;// control rod
}
if(xp[2]<1.945+TOL && xp[2]>1.945-TOL && xp[0]<1.47-TOL
   && xp[0]>1.176+TOL && xp[1]<0.735-TOL && xp[1]>0.441+TOL){
  bc_flag[2]=0;  // control rod
}
if(xp[2]<TOL && xp[2]>-TOL && xp[0]<1.47-TOL && xp[0]>1.176+TOL
```

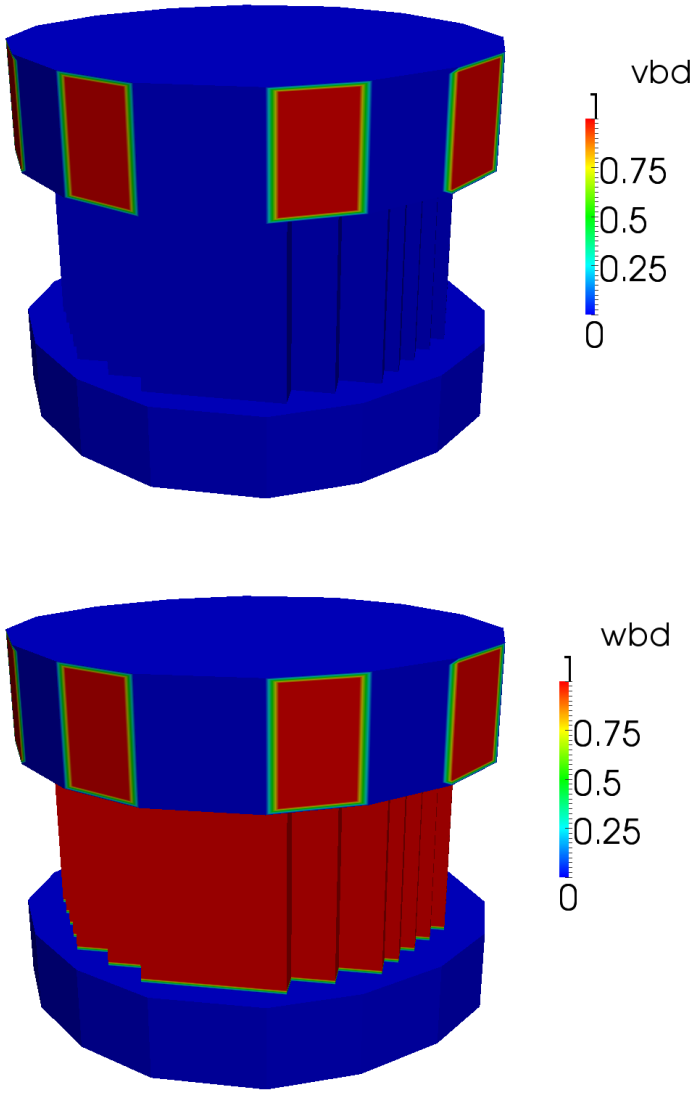

Figure 4.10: Boundary condition for the variables v and w: Dirichlet (0) and Neumann (1).

```
  && xp[1]<0.735-TOL && xp[1]>0.441+TOL) {
   bc_flag[2]=0;  bc_flag[2]=0; // control rod
}
if(xp[2]<1.945+TOL && xp[2]>1.945-TOL && xp[0]<.735+TOL
  && xp[0]>0.441-TOL && xp[1]<1.617+0.00001 && xp[1]>1.323-0.0001){
   bc_flag[0]=1;  bc_flag[1]=1;  bc_flag[2]=1; // control rod
}
if(xp[2]<TOL && xp[2]>-TOL && xp[0]<.735+TOL  && xp[0]>0.441-TOL
   && xp[1]<1.617+TOL  && xp[1]>1.323-TOL) {
   bc_flag[0]=1;  bc_flag[1]=1;  bc_flag[2]=1; // control rod
```

```
 }
 if(xp[2]<1.945+TOL && xp[2]>1.945-TOL && xp[0]<.735-TOL
    && xp[0]>0.441+TOL && xp[1]<1.617-TOL && xp[1]>1.323+TOL){
    bc_flag[2]=0; // control rod
 }
 if(xp[2]<TOL && xp[2]>-TOL && xp[0]<.735-TOL && xp[0]>0.441+TOL
    && xp[1]<1.617-TOL  && xp[1]>1.323+TOL) {
     bc_flag[2]=0;  // control rod
 }
#endif
  return;
}
```

This file sets Dirichlet boundary condition over the inlet and symmetry conditions for $x = 0$ and $y = 0$. The inlet will have an assigned velocity profile in agreement with the initial conditions. The outlet will have Neumann boundary conditions. Over the reactor walls we assume slip boundary conditions. The node point $(x, y, z) = (xp[0], xp[1], xp[2])$ is used to set all the necessary boundary conditions. The vector bc_flag is the flag for the boundary conditions on the $x - y - z$ components of the velocity field. The boundary conditions are rather complex since the reactor surface is not aligned with the axes. The boundary conditions for velocity and temperature are shown in Figures 4.10-4.9. The flag of the component u is shown on the bottom of Figure 4.9. The flags for v, w are shown on the top and bottom of Figure 4.10. In the area corresponding to a 0 value Dirichlet boundary conditions are enforced. The rest has Neumann boundary conditions.

For boundary conditions of the energy equation one must edit the function
`void MGSolT::bc_read(double xp[],double normal [],int bc_flag[])`
in the file `src/MGSolverT3D.C`. We write

```
 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;
}
```

We set Dirichlet conditions at the inlet with temperature 400 ℃. Over all the rest of the surface we set homogeneous boundary conditions, namely zero heat flux.

Initial conditions

Initial pressure and velocity solution. 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 MGSol::ic_read(double xp[],double u_value[])`
inside the file `src/MGSolverNS3D.C`. The initial condition function is

```
#define TOL 0.0001
void MGSolNS::ic_read(double xp[],double u_value[]) {
#ifdef DIM2
#else
// 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]=0.;

 double val=0.82;
 double dx=xp[0]; double dy=xp[1];
 double mm=sqrt(dx*dx+dy*dy); dx /=mm;  dy /=mm;

 if((xp[2]<0.-TOL) && (xp[2]>-0.9+TOL) && (xp[1]>-2*xp[0]+5.733-0.001)){
    u_value[0]=-dx*val; u_value[1]=-dy*val; u_value[2]=-0.05*val;
  }
 if((xp[2]<-TOL) && (xp[2]>-0.9+TOL) && (xp[1]>-0.5*xp[0]+2.793-TOL)){
    u_value[0]=-dx*val; u_value[1]=-dy*val; u_value[2]=-0.05*val;
  }
   if((xp[2]<-TOL) && (xp[2]>-0.9+TOL) && (xp[1]>-1.*xp[0]+3.528-TOL)){
    u_value[0]=-dx*val; u_value[1]=-dy*val; u_value[2]=-0.05*val;
  }
   if((xp[2]<-TOL) && (xp[2]>-0.9+TOL) && (xp[0]>2.646-TOL)){
    u_value[0]=-dx*val; u_value[1]=-dy*val; u_value[2]=-0.05*val;
  }
   if((xp[2]<-TOL) && (xp[2]>-0.9+TOL) && (xp[1] > 2.499-TOL)){
    u_value[0]=-dx*val; u_value[1]=-dy*val; u_value[2]=-0.05*val;
  }
```

Initial Energy solution. We initialize the temperature to the inlet value of 400 ℃. Inside the file `MGSolverT3D.C` in the function
`void MGSolT::ic_read(double xp[],double u_value[])`
we set

```
void MGSolNT::ic_read(double xp[],double u_value[]) {
#ifdef DIM2
#else
  // xp[]=(xp,yp,zp) u_value[]=(u,v,w,p)
  u_value[0] =400.;
```

In this case we set a uniform temperature of 400 ℃for $t = 0$.

4.4 Step 3: Analysis of the CFD solution

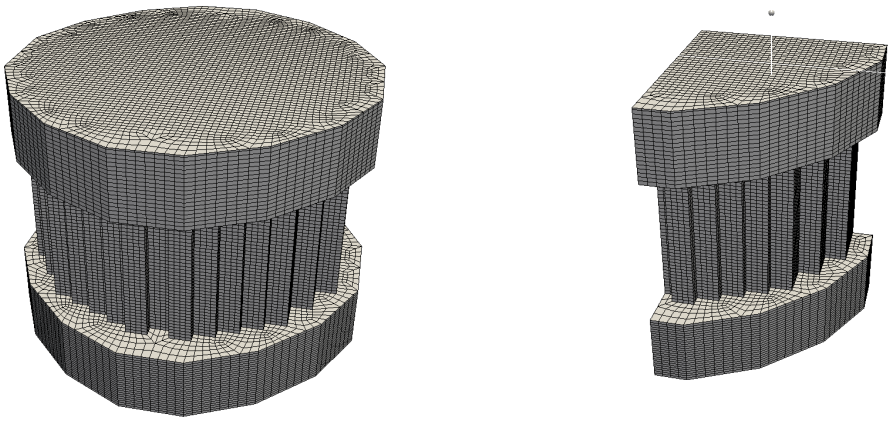

Figure 4.11: Full reactor and computational domain.

If we consider a Cartesian reference frame located at the center of the reactor the domain is not symmetric with respect to the central axis since the reactor assembly power distribution and the geometry itself are different along the x and y directions. However there is symmetry with respect to the x and y planes and therefore a quarter domain can be used. The fields over the reactor shown in 4.11 on the left are restricted to the domain shown on the right. In the rest of this section we consider four Tests:
a) Test 1: core with no assembly flow exchange (closed core model);
b) Test 2: core with upper core assembly flow exchange (partially closed assembly model)
c) Test 3: core with assembly flow exchange (open core model);
d) Test 4: core with control assemblies (open core model with control assemblies);

In Test 1 we consider closed assemblies. This is taken into account 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. In Test 2 we consider closed assemblies on the lower core and open along the upper core. As before this is taken into account setting to zero the x and y-components of the velocity fields on the lateral surface of each assembly in the core region with $z < H_{in}$ where there are no assembly cross flows. The open core model is considered in Test 3 where the velocity is imposed to be parallel to the z-axis only at the lower core inlet. Finally in Test 4 we consider an open core model with a different geometry where special control rod assemblies are inserted inside the core of the reactor.

4.4.1 Test 1. Closed core model

In this test we consider the reactor to be 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 distribution of power in the upper core is assumed as in Figures 4.12-4.13. In this test we do not reproduce the two control assemblies, eight in the

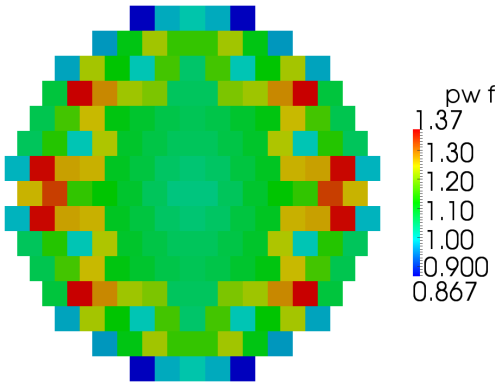

Figure 4.12: Test 1. The core horizontal power distribution factor over the 170 assemblies.

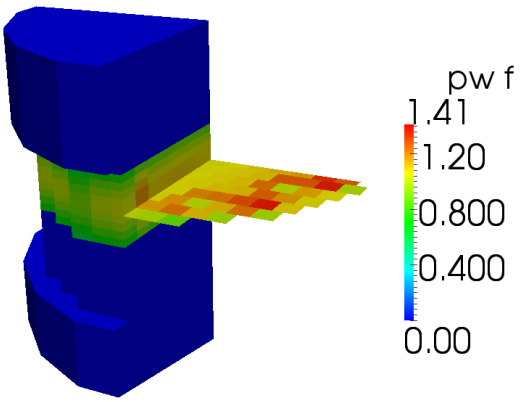

Figure 4.13: Test 1. Core view of power distribution factors.

full reactor (see Test 4 for the complete case) and for this reason the total reactor power is divided among 170 assemblies to have the average volumetric and linear assembly power of

$$\dot{q}_v = \frac{\dot{Q}}{A} = \frac{1.482 \times 10^9}{0.294 \times 0.294 \times 170} = 1.0086 \times 10^8 \, \frac{W}{m^2}$$

and

$$q_l = \frac{\dot{q}_v}{H_{out} - H_{in}} = \frac{1.0086 \times 10^8}{0.9} = 1.1206 \times 10^8 \, \frac{W}{m}. \tag{4.14}$$

In the lower and upper core we take into account the closed assembly hypothesis by imposing the velocity fields on the assembly lateral surface to be parallel to the z-direction and therefore setting the u and v components on those surfaces to zero. The reactor regions of the core and the plena must be connected with appropriate yielding conditions which can be defined

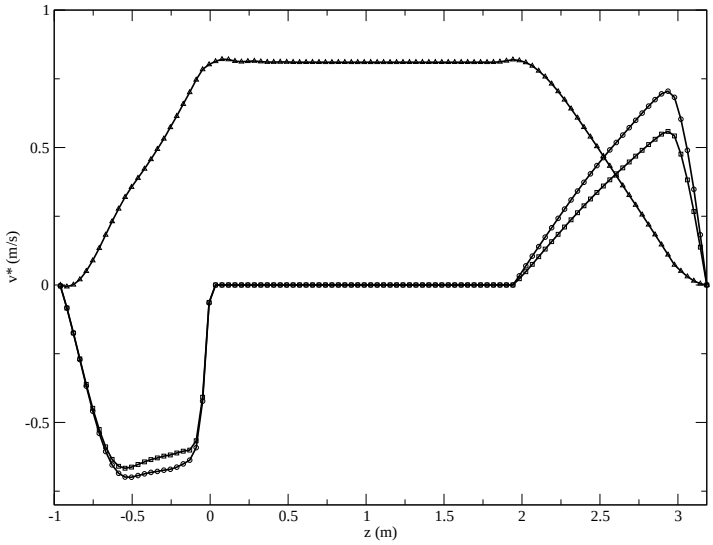

Figure 4.14: Test 1. The x (circle), y (square) and z (triangle) components of the vector $\mathbf{v}^*$ field along a z-line centered at $x = 1.323$ and $y = 1.2495$.

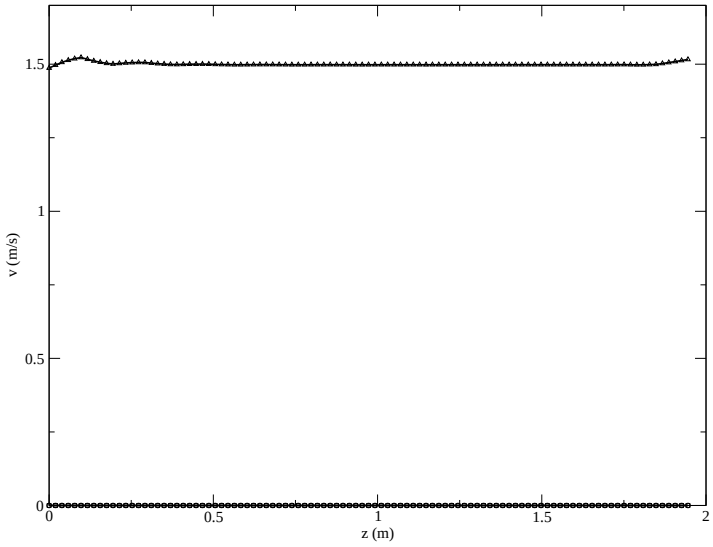

Figure 4.15: Test 1. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the core along a z-line centered at $x = 1.323$ and $y = 1.2495$.

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 but must take a value based 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

$$\mathbf{v}^* = \begin{cases} \mathbf{v} & \text{on } \Omega_{lp} \cup \Omega_{up} \\ \dfrac{\mathbf{v}}{r} & \text{on } \Omega_c \end{cases} \tag{4.15}$$

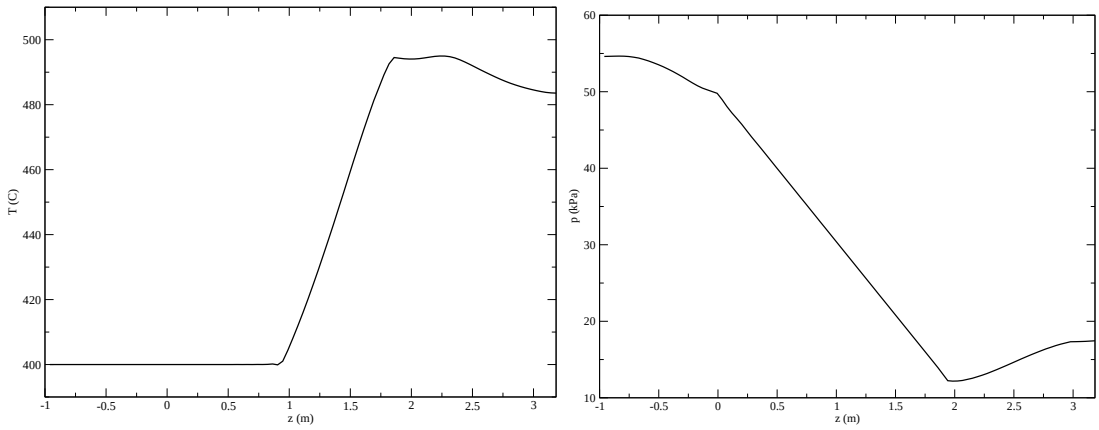

Figure 4.16: Test 1. The temperature and pressure along a z-line at $x = 1.323$ and $y = 1.2495$.

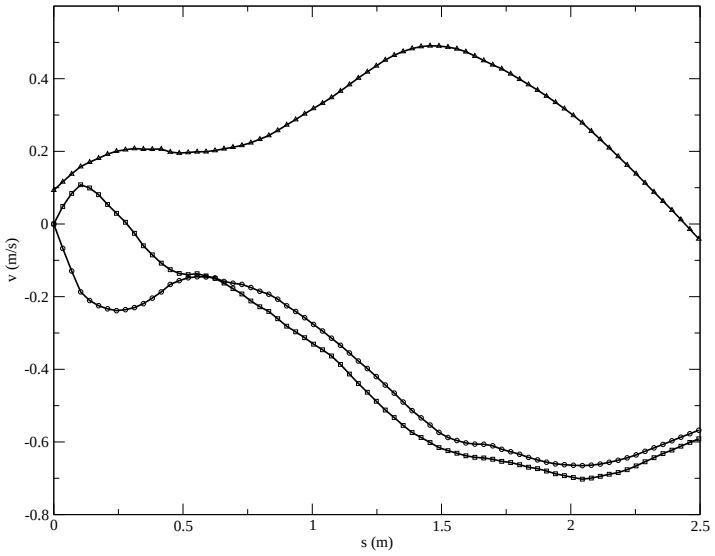

Figure 4.17: Test 1. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the lower plenum (diagonal line at $z = -0.5$).

which is continuous across all the reactor. The distribution of the temperature, pressure, vector $\mathbf{v}^*$ and velocity fields in the reactor are reported in the next paragraphs. In Figures 4.14-4.16 we see the components of the vector $\mathbf{v}^*$, the velocity, the temperature and the pressure fields along a central line parallel to the z-axis. The line is placed at coordinates $x = 1.323$ and $y = 1.2495$. In Figure 4.14 the x-component of $\mathbf{v}^*$ is shown as a line with circles and the y and z components as lines with squares and triangles respectively. The velocity field in the upper and lower plena matches the vector $\mathbf{v}^*$ but in the core the velocity is scaled by the occupation factor. The velocity field in the core is shown in Figure 4.15. Since the core assembly is closed the w component of the velocity field is constant and the u and v components are zero across all the core. The temperature and pressure on the same vertical line along all the reactor are shown in Figure 4.16 on the left and on the right respectively. Temperatures start to increase inside the upper core region where the fuel assemblies generate heat and reach the maximum value at the core outlet. In the upper plenum the coolant with

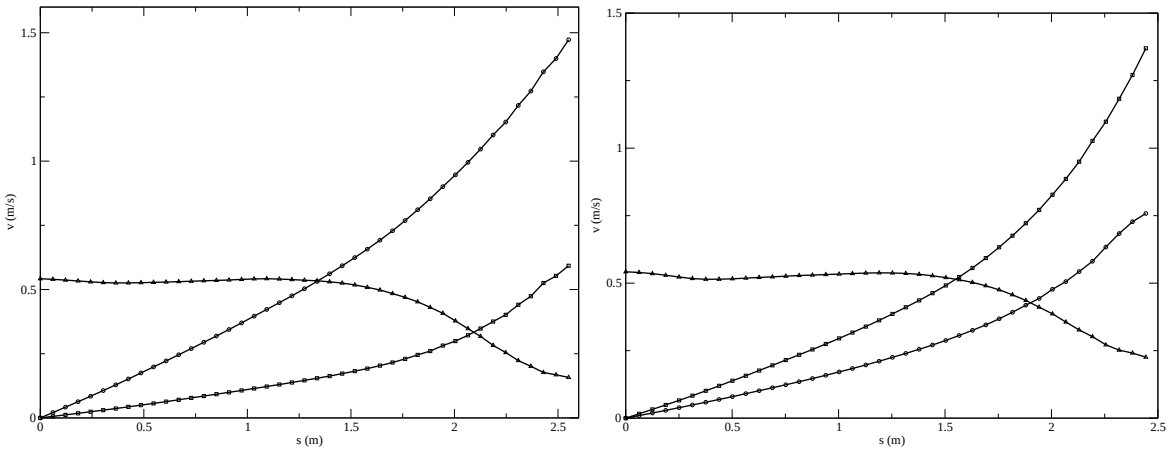

Figure 4.18: Test 1. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the upper plenum for the diagonal line on the right outlet (on the left) and on the left outlet (on the right).

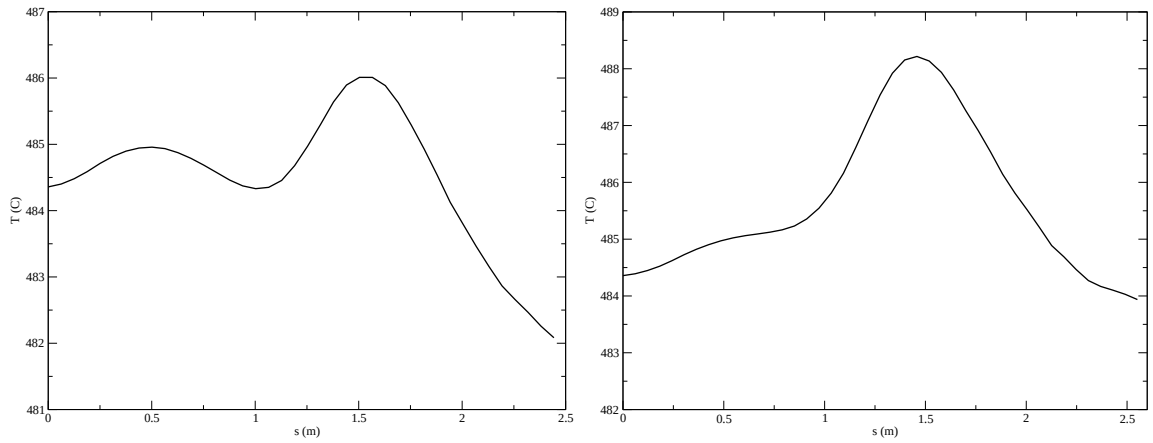

Figure 4.19: Test 1. Temperature in the upper plenum along the diagonal line on the right outlet (on the left) and on the left outlet (on the right).

different temperatures mixes before exiting the reactor. The pressure decreases mainly inside the lower and upper core regions due to the friction factor inside the sub-assemblies. The pressure drop inside the core is estimated to be approximately $0.32\,bar$ (see Section 4.1). The pressure drops due to grids and spacers are ignored but they can easily be taken into account as in [3]. In Figure 4.17 the x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the lower plenum along the diagonal line at $z = -0.5$ is reported. The u and v components decrease inside the lower plenum while the vertical component w increases. In Figures 4.18 and 4.19 the velocity and the temperature fields are shown in the upper plenum using the same symbols as before. In Figure 4.18 the velocity components are reported along two lines on the upper plenum middle plane. These lines connect the central point to the middle of the exit window close to the y-axis (left) and to the x-axis (right) respectively.

Temperature

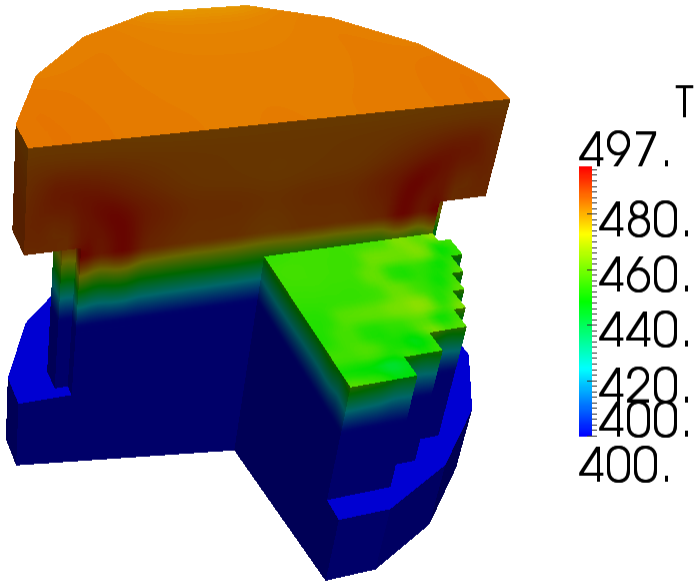

Figure 4.20: Test 1. Overview of the temperature distribution T over the reactor

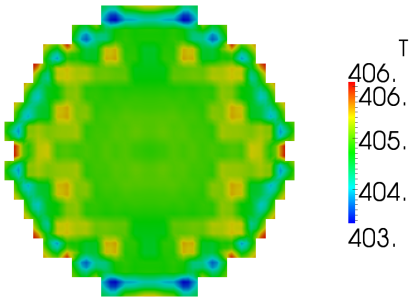

Figure 4.21: Test 1. Temperature distribution over the plane $z = 1.$ (lower core).

In Figure 4.20 there is an overview of the temperature distribution T over the reactor. In Figures 4.21-4.24 we have the temperature distribution over the planes $z = 1.$ (lower core), $z = 1.5$ (upper core), $z = 2$ (upper plenum inlet) and $z = 2.5$ (upper plenum middle plane). We note that the maximum temperature over the upper plenum inlet is higher than the

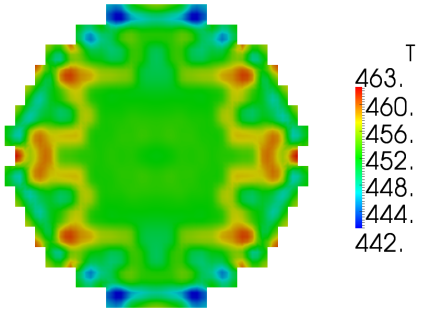

Figure 4.22: Test 1. Temperature distribution over the plane $z = 1.5$ (upper core).

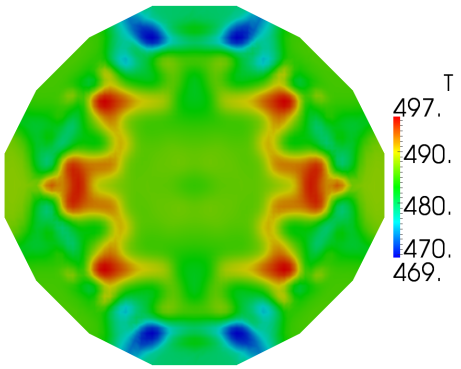

Figure 4.23: Test 1. Temperature distribution over the plane $z = 2.$ (upper plenum inlet).

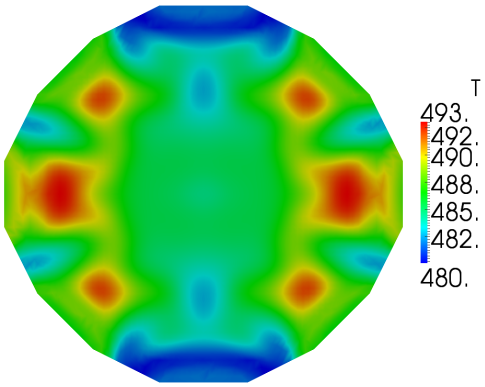

Figure 4.24: Test 1. Temperature distribution over the plane $z = 2.5$ (upper plenum middle plane).

maximum temperature over the upper plenum middle plane.

Pressure

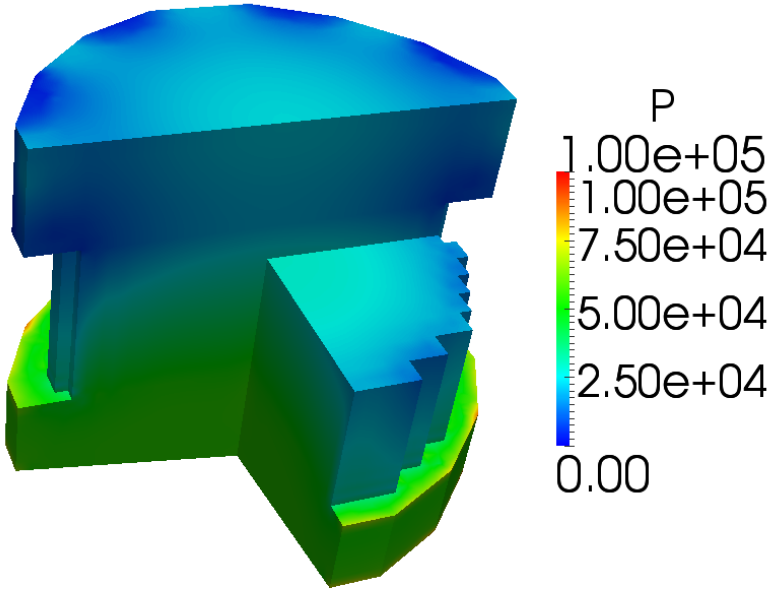

Figure 4.25: Test 1. Overview of the pressure distribution p over the reactor

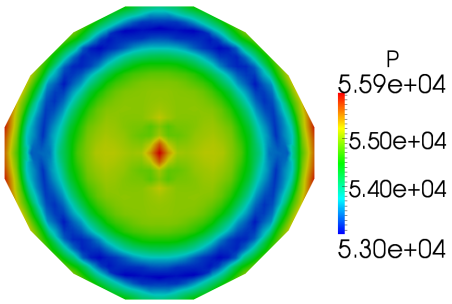

Figure 4.26: Test 1. Pressure distribution over the plane $z = -0.5$ (lower plenum).

In Figure 4.25 there is an overview of the pressure distribution p over the reactor. In Figures 4.26-4.29 we have the pressure distribution over the planes $z = -0.5$ (lower plenum), $z = 0.5$ (lower core), $z = 1.5$ (upper core) and $z = 2.5$ (upper plenum). The reference pressure is set to zero at the upper plenum outlet.

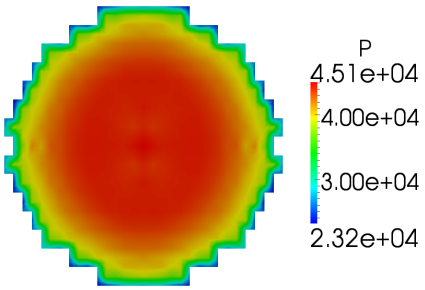

Figure 4.27: Test 1. Pressure distribution over the plane $z = 0.5$ (lower core).

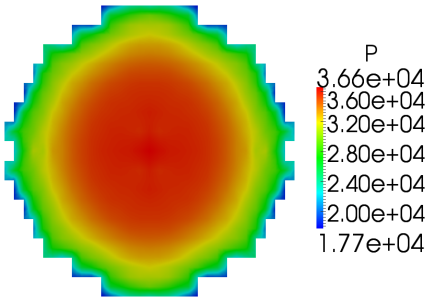

Figure 4.28: Test 1. Pressure distribution over the plane $z = 1.$ (upper core).

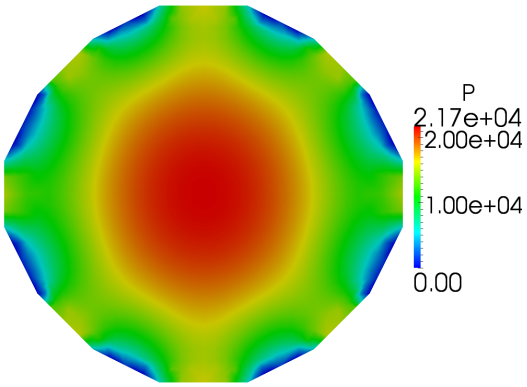

Figure 4.29: Test 1. Pressure distribution over the plane $z = 2.5$ (upper plenum).

Velocity field

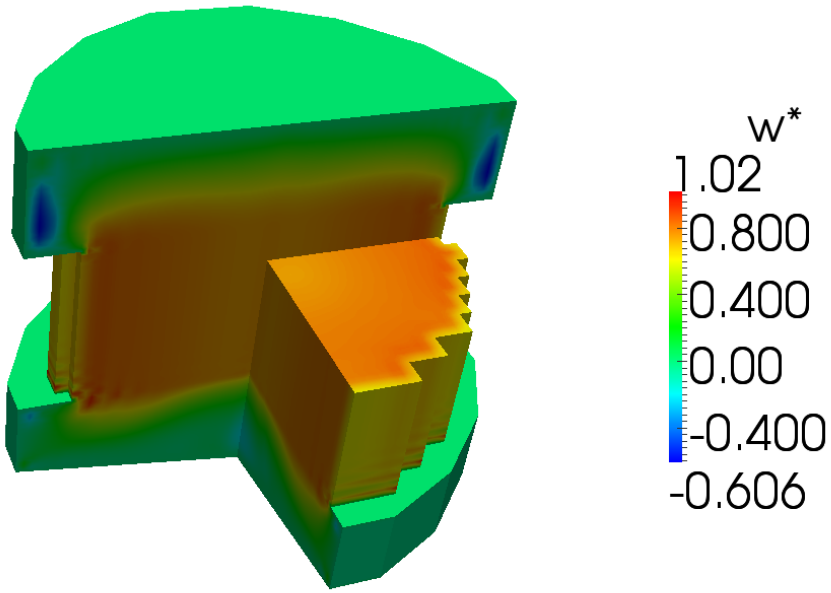

Figure 4.30: Test 1. Overview of the w^* component of the vector field $\mathbf{v}^*$ over the reactor

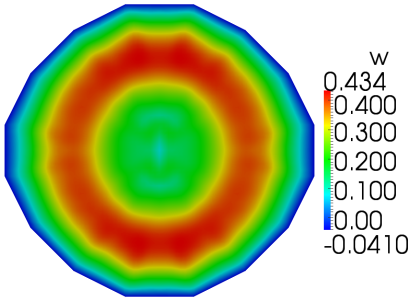

Figure 4.31: Test 1. The velocity component w over the plane $z = -0.5$ (lower plenum).

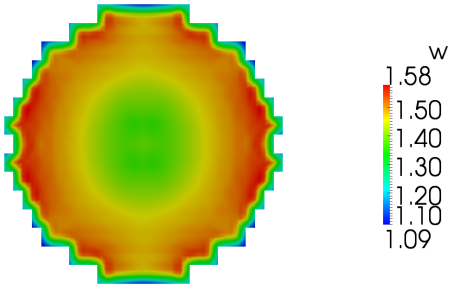

Figure 4.32: Test 1. The velocity component w over the plane $z = 0.5$ (lower core).

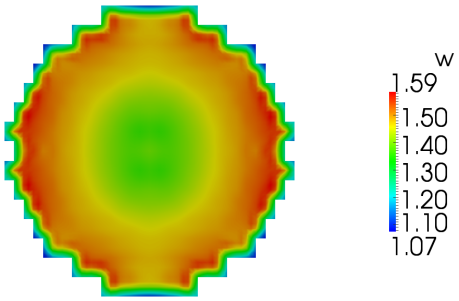

Figure 4.33: Test 1. The velocity component w over the plane $z = 1.$ (upper core).

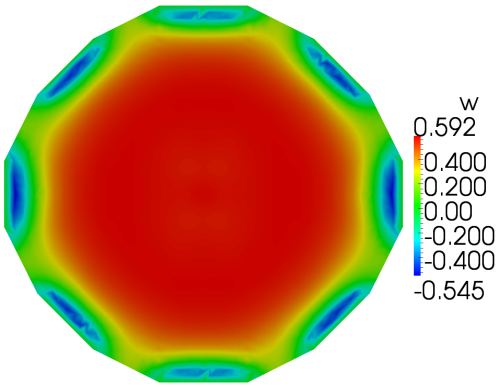

Figure 4.34: Test 1. The velocity component w over the plane $z = 2.5$ (upper plenum).

In Figure 4.30 there is an overview of the w^* component distribution of the vector field $\mathbf{v}^*$ over the reactor. In Figures 4.31-4.34 we show the w velocity component distribution over the planes $z = -0.5$ (lower plenum), $z = 0.5$ (lower core), $z = 1.5$ (upper core) and $z = 2.5$ (upper plenum). We note that the w^* component is continuous but the w component of the velocity field scales with the occupation ratio r.

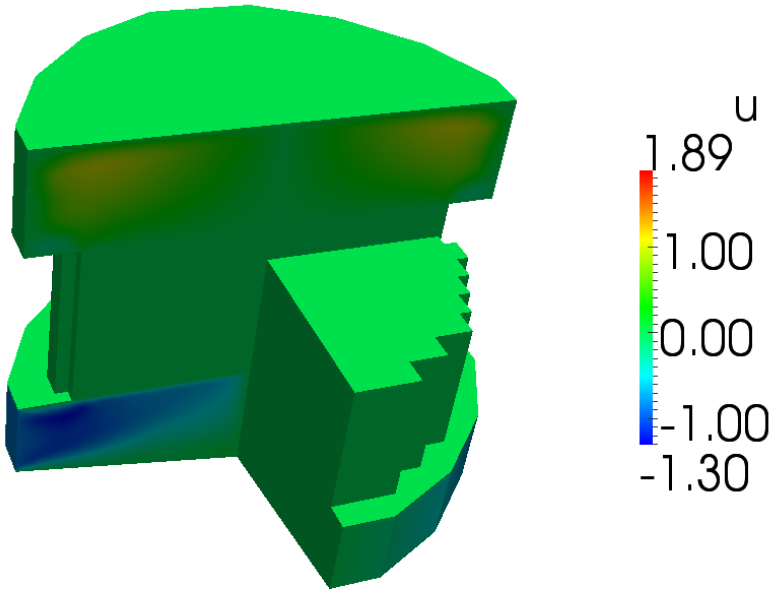

Figure 4.35: Test 1. Overview of the u^* component of the vector field $\mathbf{v}^*$ over the reactor

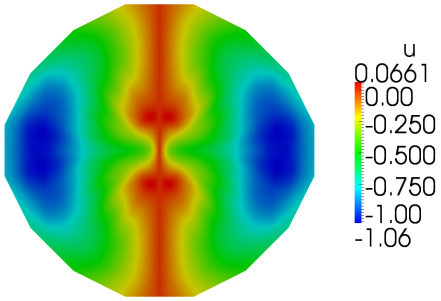

Figure 4.36: Test 1. The velocity component u over the plane $z = -0.5$ (lower plenum).

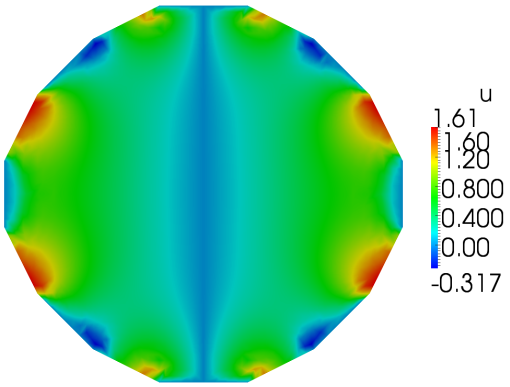

Figure 4.37: Test 1. The velocity component u over the plane $z = 2.5$ (upper plenum).

In Figure 4.35 there is an overview of the u^* component distribution of the vector field $\mathbf{v}^*$ over the reactor. In Figures 4.36-4.37 we show the u velocity component distribution over the planes $z = -0.5$ (lower plenum) and $z = 2.5$ (upper plenum). We note that the u component is zero across the core (closed assembly model).

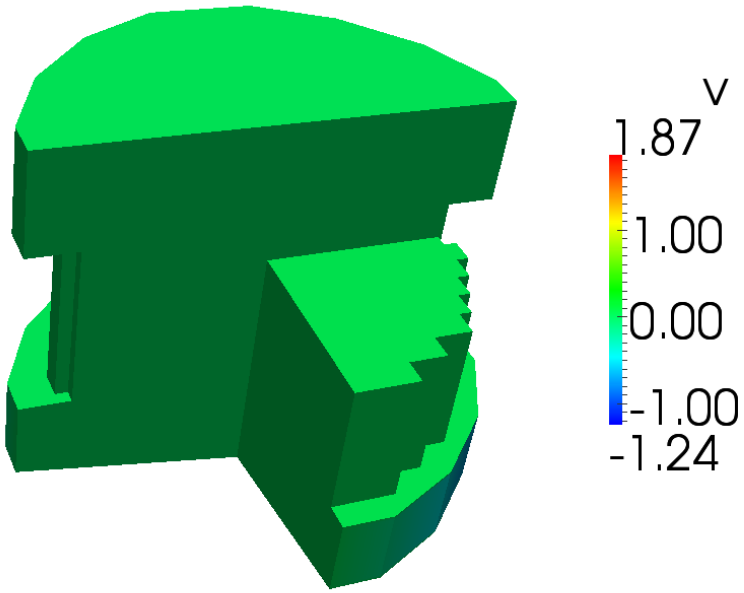

Figure 4.38: Test 1. Overview of the v^* component of the vector field $\mathbf{v}^*$ over the reactor

In Figure 4.38 there is an overview of the v^* component distribution of the vector field

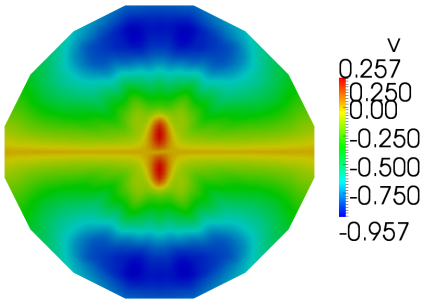

Figure 4.39: Test 1. The velocity component v over the plane $z = -0.5$ (lower plenum).

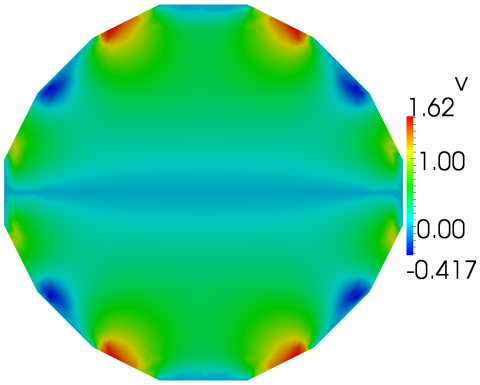

Figure 4.40: Test 1. The velocity component v over the plane $z = 2.5$ (upper plenum).

$\mathbf{v}^*$ over the reactor. In Figures 4.39-4.40 we show the v velocity component distribution over the planes $z = -0.5$ (lower plenum) and $z = 2.5$ (upper plenum). As the u component, the v component is zero across all the core.

4.4.2 Test 2. Partially closed core model

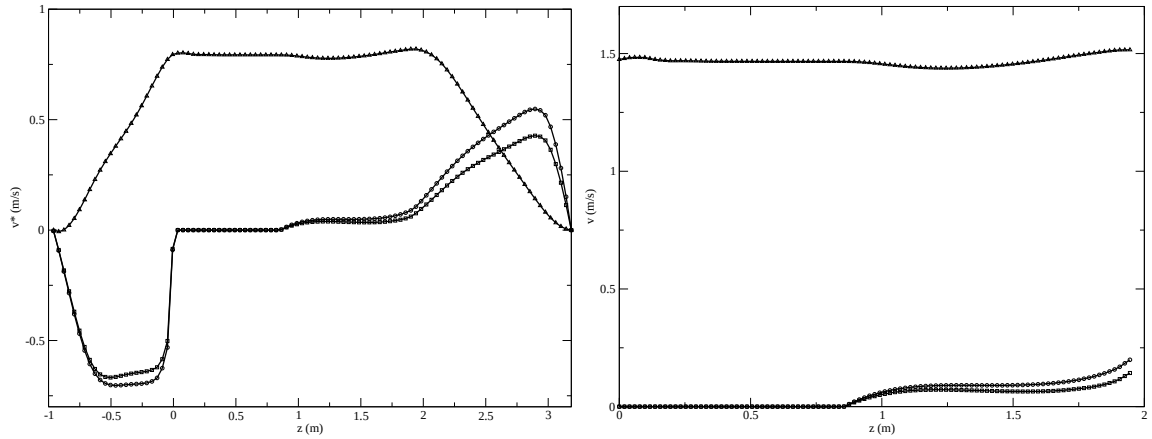

Figure 4.41: Test 2. On the left the x (circle), y (square) and z (triangle) components of the vector field $\mathbf{v}^*$ along a z-line centered at $x = 1.323$ and $y = 1.2495$ along all the reactor. On the left the same profile but for the velocity field $\mathbf{v}$ in the core region.

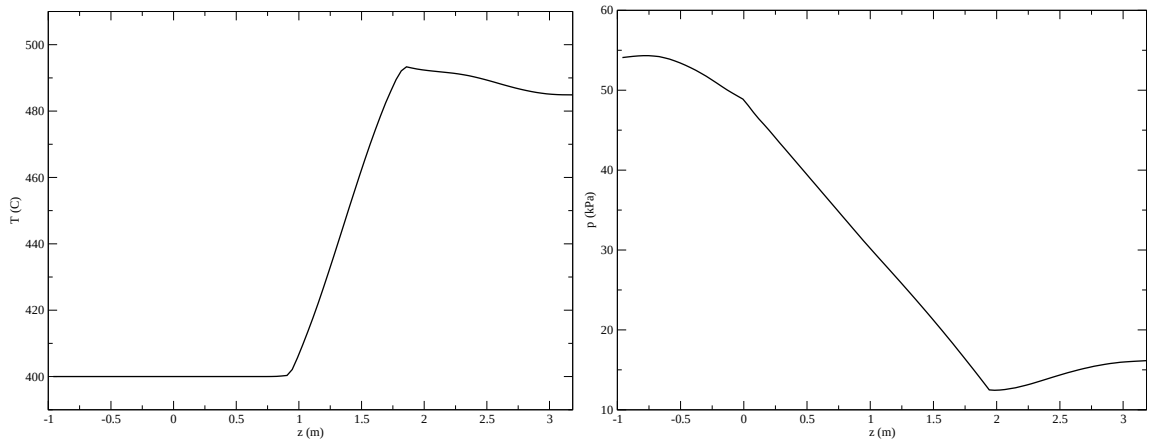

Figure 4.42: Test 2. The temperature and pressure along a z-line centered at $x = 1.323$ and $y = 1.2495$.

The reactor consists of four regions as in Test 1. In the lower and upper plenum we solve the three-dimensional Navier-Stokes and energy system with a simple turbulence LES model. In the upper core we use the same power distribution as in Test 1 but we assume open assemblies. This is obtained by imposing the velocity field on the lateral surfaces of the assemblies to be parallel to the z direction in the lower core region, while the complete three-dimensional system is solved over the top part of the core. As in Test 1 the mass flow rate at the inlet of the lower core and at the outlet of the upper core must match the lower and the upper plenum mass flow rates respectively. For this reason we can define the continuous vector field $\mathbf{v}^*$ as in 4.16. In Figures 4.41-4.42 we see the components of the vector field $\mathbf{v}^*$, the core velocity field $\mathbf{v}$, the temperature and the pressure along a central line parallel to the z-axis. The line is located at coordinates $x = 1.323$ and $y = 1.2495$. In Figures 4.41 the x,

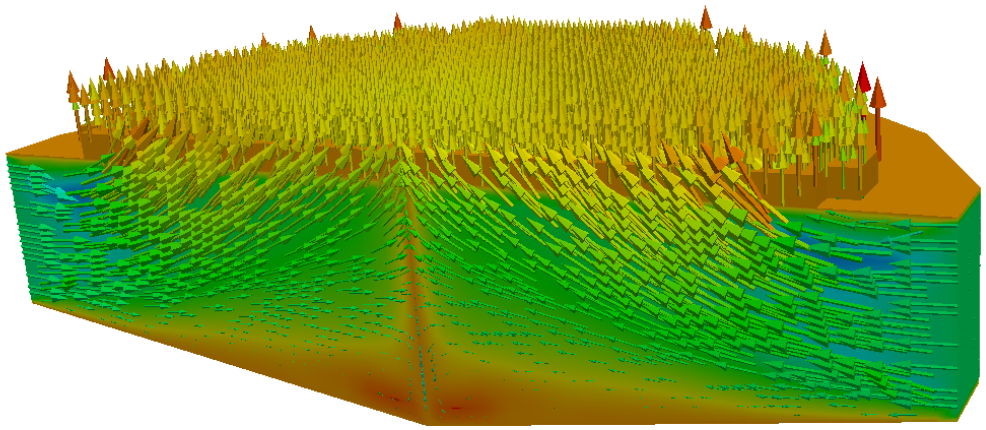

Figure 4.43: Test 2. The velocity field $\mathbf{v}$ in the lower plenum.

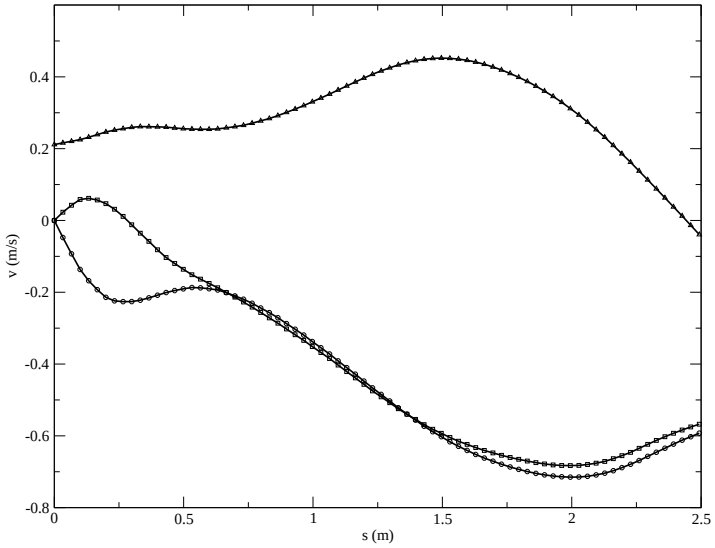

Figure 4.44: Test 2. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the lower plenum (diagonal line at $z = -0.5$).

y and z components of $\mathbf{v}^*$ are shown as lines with circles, squares and triangles, respectively. The velocity field in the upper and lower plenum matches the vector $\mathbf{v}^*$ but in the core region

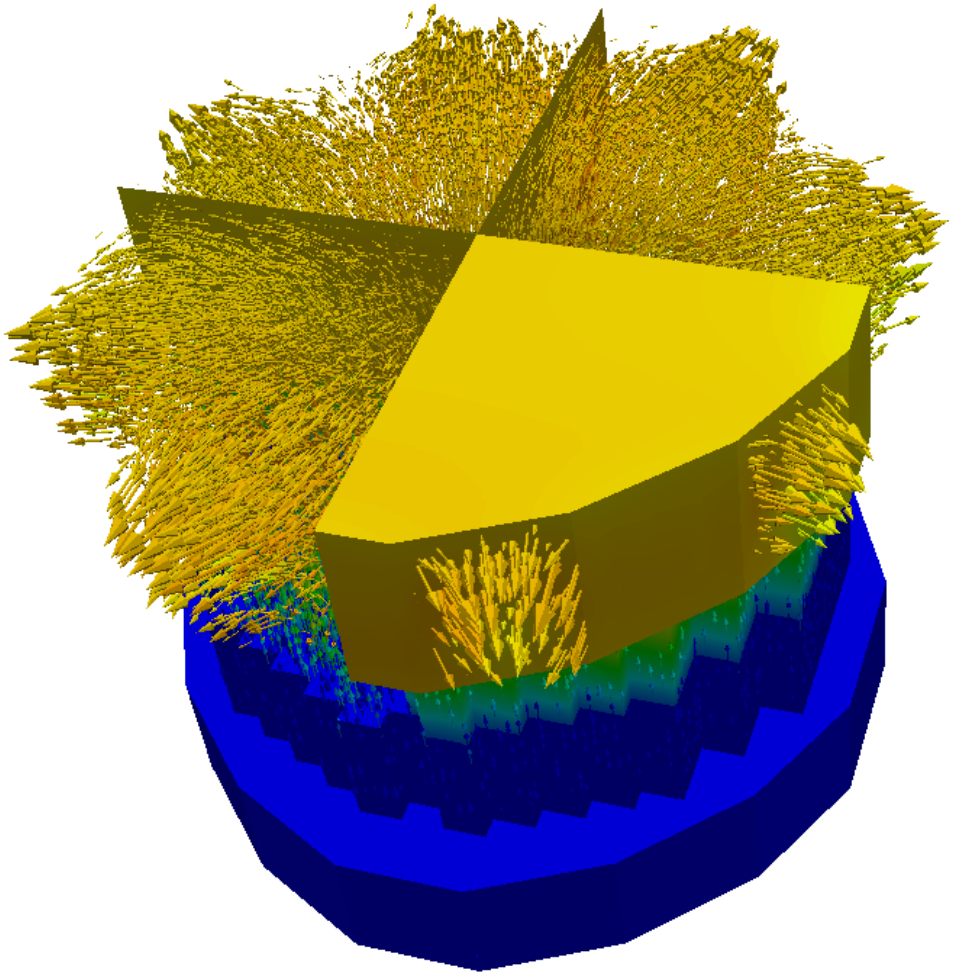

Figure 4.45: Test 2. The vector field $\mathbf{v}^*$ in the upper plenum.

the velocity is scaled by the occupation ratio. The core velocity field is shown on the right of Figure 4.41. Since the lower core assembly is closed the w component of the velocity field is constant and the u and v components are zero across the lower core region. In the upper core region the velocity field becomes three-dimensional. The temperature and pressure in the same vertical line along all the reactor are shown in Figure 4.42 on the left and on the right, respectively. The temperature starts to increase inside the upper core region where the fuel assemblies generate heat and reaches the maximum value at the outlet of the core. In the upper plenum the coolant with different temperatures mixes before exiting from the reactor. The pressure decreases mainly inside the bottom and upper core regions due to the friction factor inside the sub-assembly. Again the pressure drop inside the core is estimated to be approximately $0.32\,bar$. The velocity field in the lower plenum can be seen in Figures

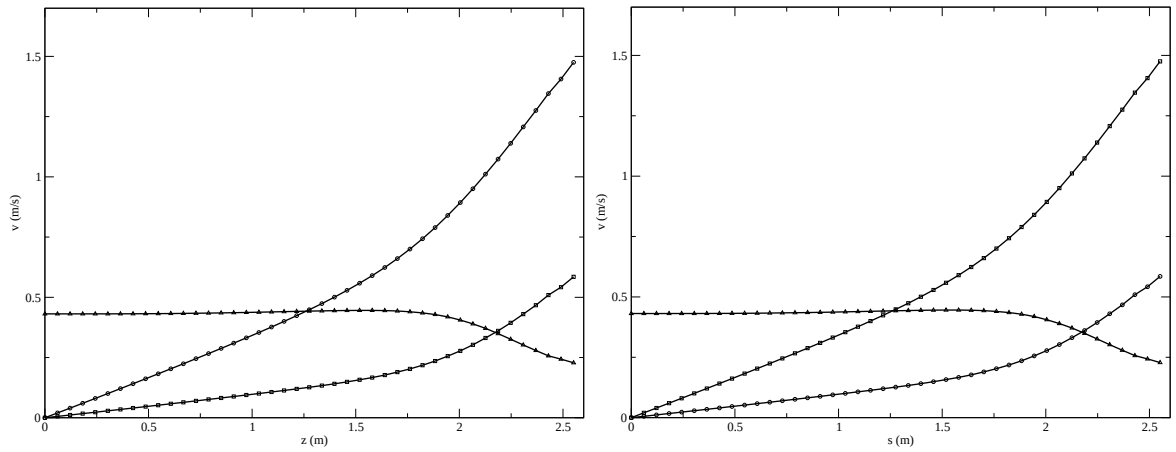

Figure 4.46: Test 2. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the upper plenum horizontal middle plane for the diagonal line to the right outlet (on the left) and to the left outlet (on the right).

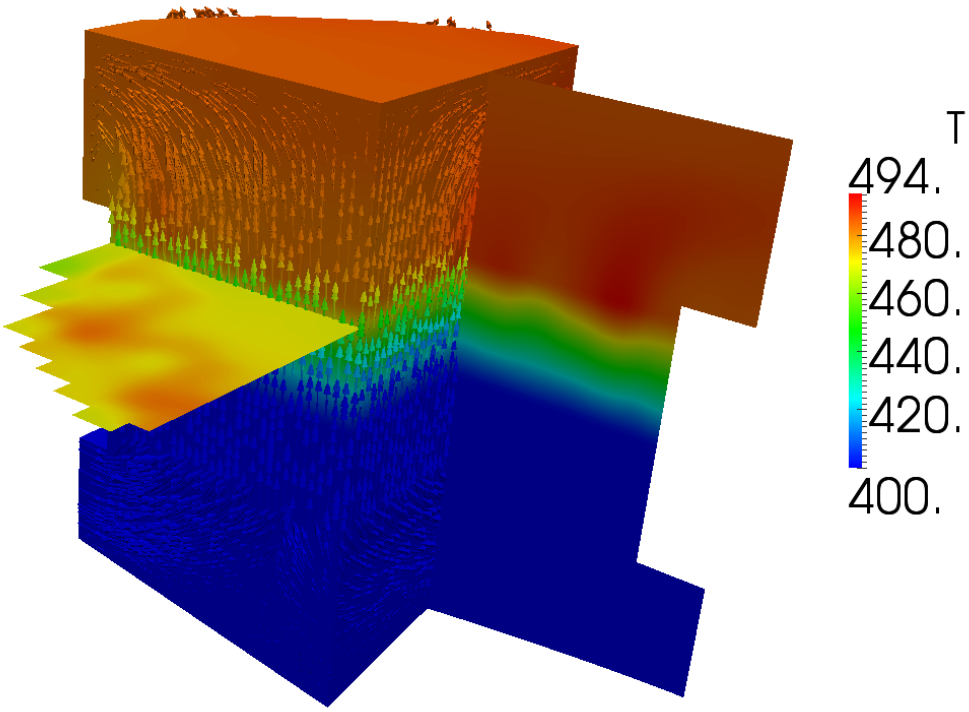

Figure 4.47: Test 2. The vector $\mathbf{v}^*$ and temperature fields in various sections of the reactor.

4.43-4.44. In Figure 4.44 the results are plotted along the diagonal line at $z = -0.5$. In Figures 4.45-4.48 we can see the vector $\mathbf{v}^*$, the velocity and the temperature fields in the upper plenum region. As in Test 1 the fields are reported along two lines on the upper

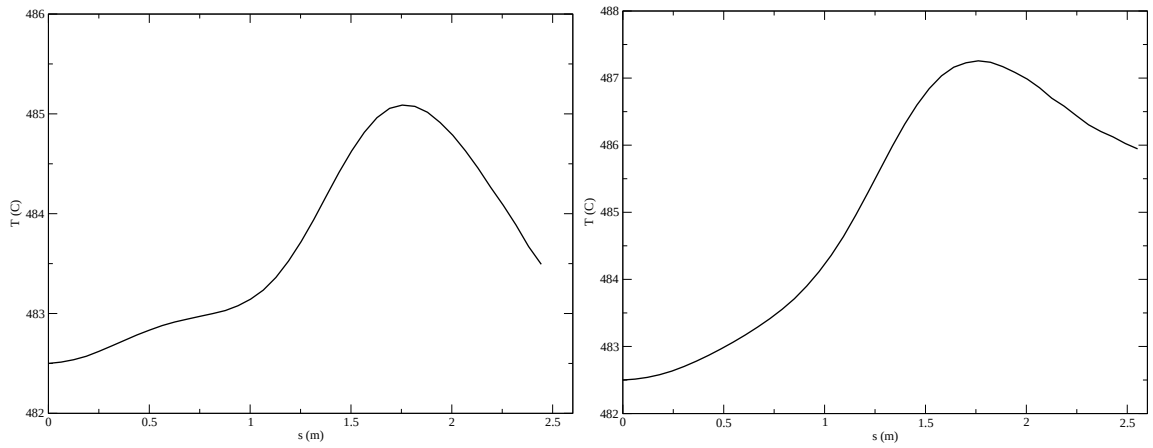

Figure 4.48: Test 2. Temperature in the upper plenum horizontal middle plane along the diagonal line to the right outlet (on the left) and to the left outlet (on the right).

plenum middle plane. These lines connect the axis point to the middle of the exit window close to the y-axis (left) and to the x-axis (right) respectively.

4.4.3 Test 3. Open core model

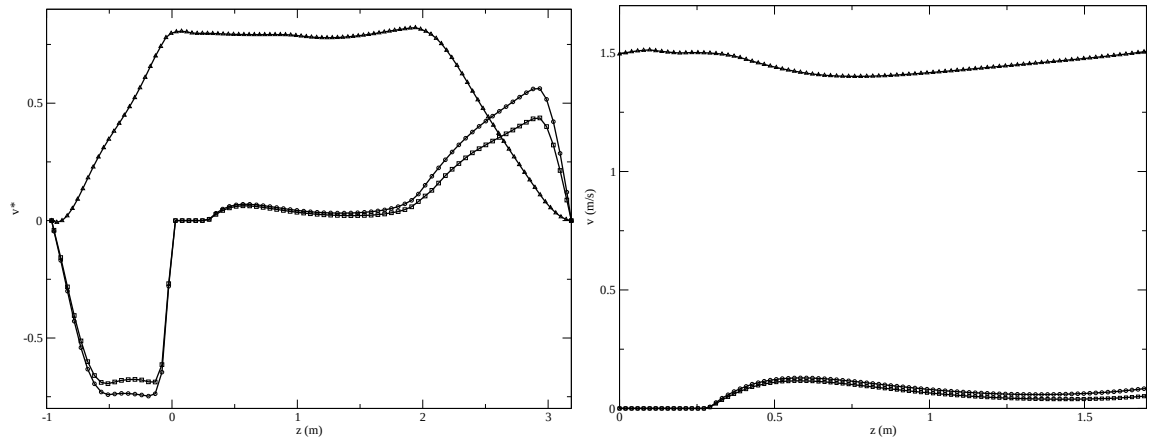

Figure 4.49: Test 3. On the left the x (circle), y (square) and z (triangle) components of the vector field $\mathbf{v}^*$ along the vertical line at $x = 1.323$ and $y = 1.2495$ along all the reactor. On the right the same profile but for the velocity field $\mathbf{v}$ in the core region.

In this test we assume open assemblies. The open assembly model is obtained by imposing the velocity fields to be parallel to the z-direction only at the core inlet and by solving the complete three-dimensional core system over the rest of the core. As in Test 1 and Test 2 we can define the continuous vector field $\mathbf{v}^*$ as in 4.16. In Figures 4.49-4.50 we see the components of the vector field $\mathbf{v}^*$, the core velocity field $\mathbf{v}$, the temperature and the pressure along a line parallel to the z-axis and located at coordinates $x = 1.323$ and $y = 1.2495$. In Figures 4.49 the x, y and z components of $\mathbf{v}^*$ are shown as lines with circles, squares and triangles respectively. The velocity field in the upper and lower plena matches the vector

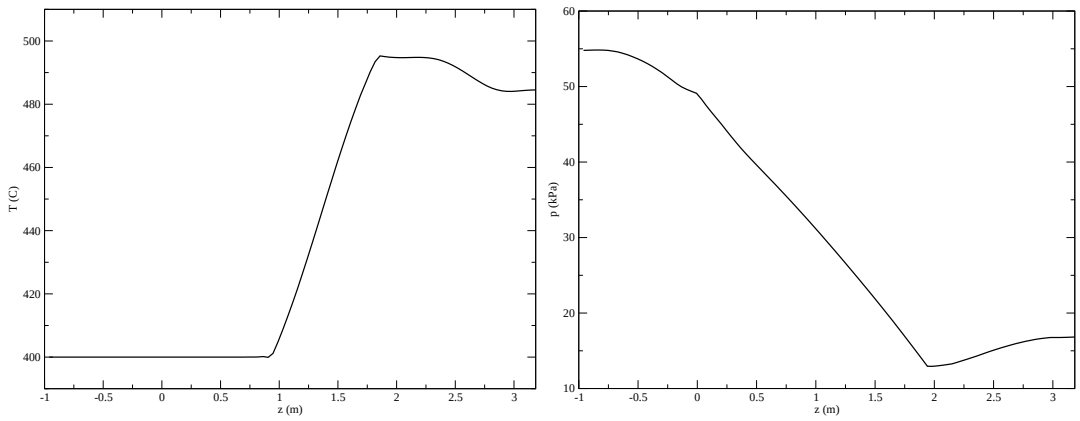

Figure 4.50: Test 3. The temperature and pressure along the vertical line at $x = 1.323$ and $y = 1.2495$.

$\mathbf{v}^*$ but in the core the velocity is scaled by the occupation ratio. The core velocity field is shown on the right of Figure 4.49. Since the core assembly is open the velocity field becomes three-dimensional over all the core region. The temperature and pressure on the same vertical line along all the reactor are shown in Figure 4.50 on the left and on the right respectively. The temperature starts to increase inside the upper core region where the fuel assemblies generate heat and reaches the maximum value at the outlet of the core. In the upper plenum the coolant with different temperatures mixes before exiting from the reactor. The pressure decreases mainly inside the bottom and top core regions due to the friction factor inside the sub-assemblies.

Temperature and pressure

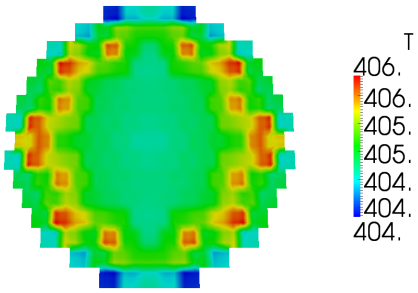

Figure 4.51: Test 3. Temperature distribution over the plane $z = 1.$ (upper core inlet).

In Figures 4.51-4.54 we have the temperature distribution over the planes $z = 1.$ (upper core inlet), $z = 1.5$ (upper core middle plane), $z = 2.$ (upper plenum inlet) and $z = 2.5$ (upper plenum middle plane). We note that the maximum temperature over the upper plenum inlet is higher than the maximum temperature over the upper plenum middle plane. In Figures

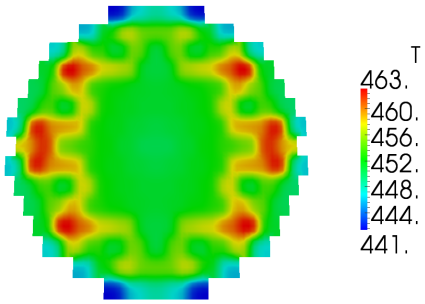

Figure 4.52: Test 3. Temperature distribution over the plane $z = 1.5$ (upper core middle plane).

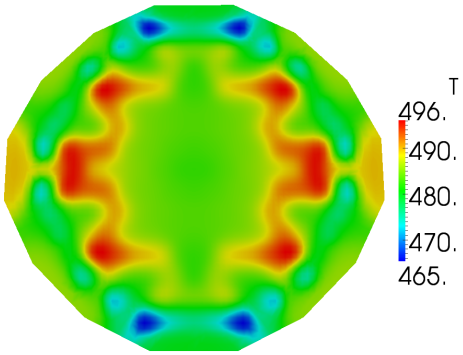

Figure 4.53: Test 3. Temperature distribution over the plane $z = 2.$ (upper plenum inlet).

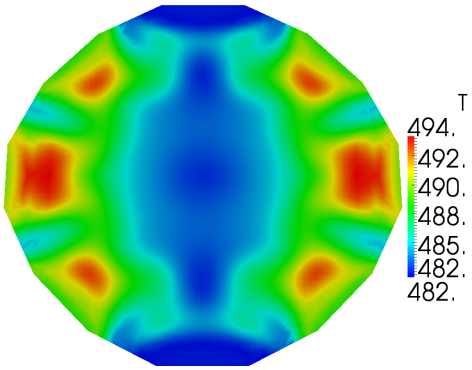

Figure 4.54: Test 3. Temperature distribution over the plane $z = 2.5$ (upper plenum middle plane).

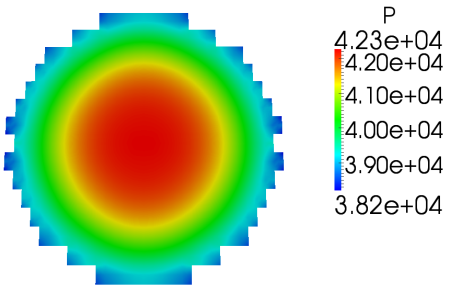

Figure 4.55: Test 3. Pressure distribution over the plane $z = 0.5$ (lower core).

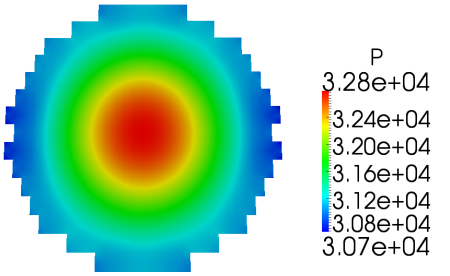

Figure 4.56: Test 3. Pressure distribution over the plane $z = 1.$ (upper core inlet).

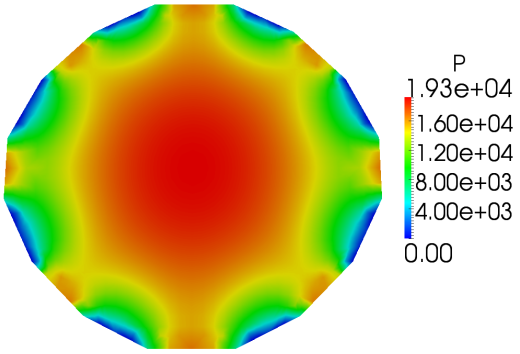

Figure 4.57: Test 3. Pressure distribution over the plane $z = 2.5$ (upper plenum).

4.55-4.57 we have the pressure distribution over the planes $z = 0.5$ (lower core), $z = 1$ (upper core inlet) and $z = 2.5$ (upper plenum). The reference pressure is set to zero outside the

reactor upper plenum.

Velocity

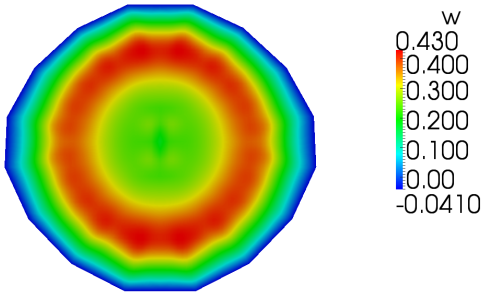

Figure 4.58: Test 3. The velocity component w over the plane $z = -0.5$ (lower plenum).

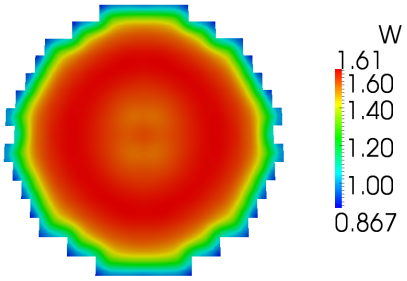

Figure 4.59: Test 3. The velocity component w over the plane $z = 0.5$ (lower core).

In Figures 4.58-4.61 we show the w-component distribution over the planes $z = -0.5$ (lower plenum), $z = 0.5$ (lower core), $z = 1$ (upper core inlet) and $z = 2.5$ (upper plenum). We note that the w^* component is continuous but the w component of the velocity field scales with the occupation ratio r. In Figures 4.62-4.65 we show the u component distribution over the planes $z = -0.5$ (lower plenum), $z = 1$ (upper core inlet), $z = 1.5$ (upper core) and $z = 2.5$ (upper plenum). In Figures 4.66-4.68 we show the v component distribution over the planes $z = -0.5$ (lower plenum), $z = 1.$ (upper core inlet) and $z = 1.5$ (upper core). We remark that the u and v components do not vanish across the core and the motion is fully three-dimensional.

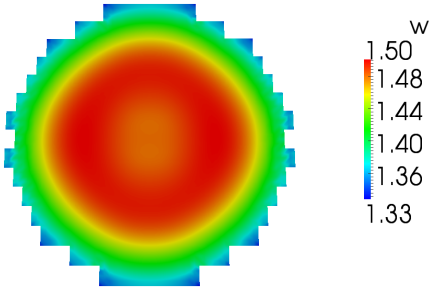

Figure 4.60: Test 3. The velocity component w over the plane $z = 1$. (upper core inlet).

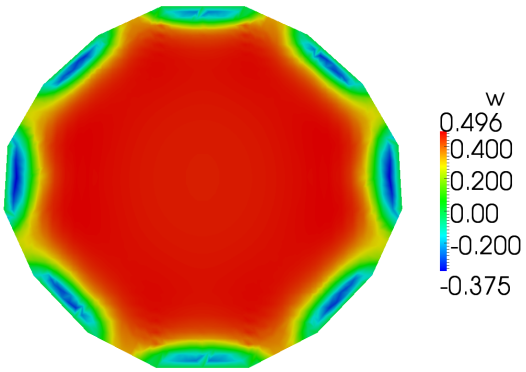

Figure 4.61: Test 3. The velocity component w over the plane $z = 2.5$ (upper plenum).

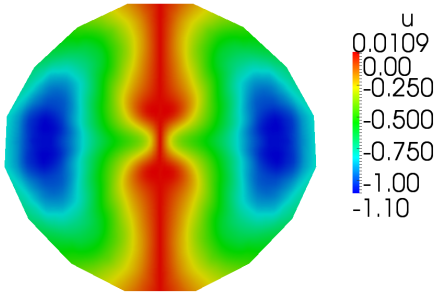

Figure 4.62: Test 3. The velocity component u over the plane $z = -0.5$ (lower plenum).

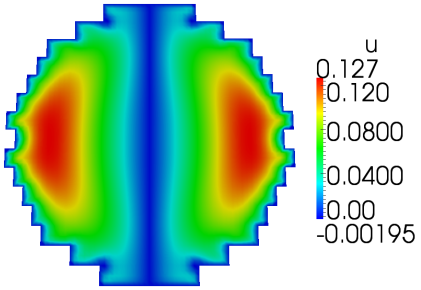

Figure 4.63: Test 3. The velocity component u over the plane $z = 1$. (upper core inlet).

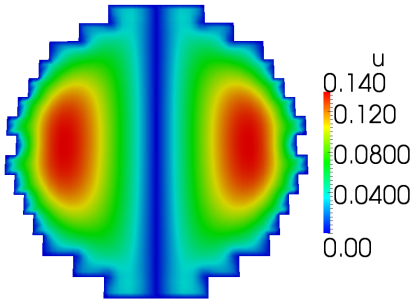

Figure 4.64: Test 3. The velocity component u over the plane $z = 1.5$ (upper core).

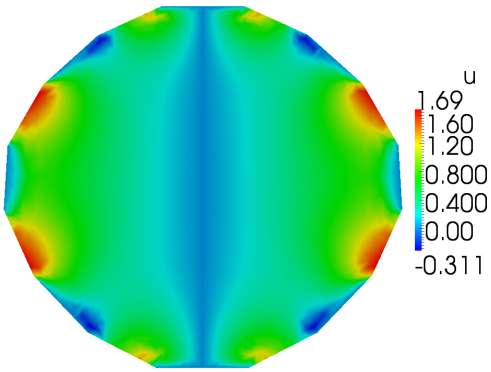

Figure 4.65: Test 3. The velocity component u over the plane $z = 2.5$ (upper plenum).

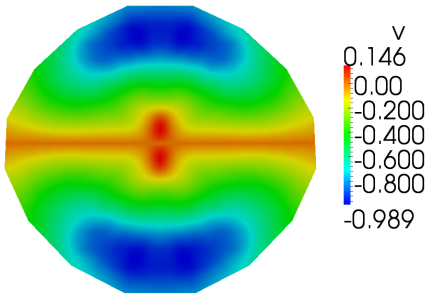

Figure 4.66: Test 3. The velocity component v over the plane $z = -0.5$ (lower plenum).

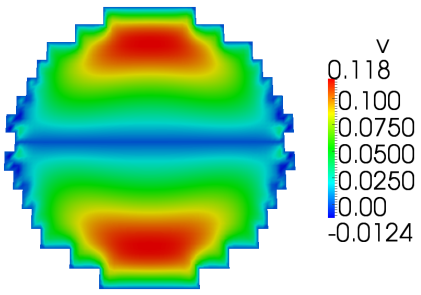

Figure 4.67: Test 3. The velocity component v over the plane $z = 1$. (upper core inlet).

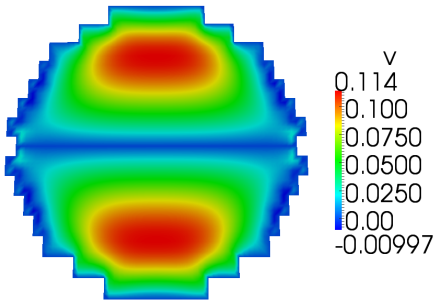

Figure 4.68: Test 3. The velocity component v over the plane $z = 1.5$ (upper core).

4.4.4 Test 4. Open core model with control assemblies

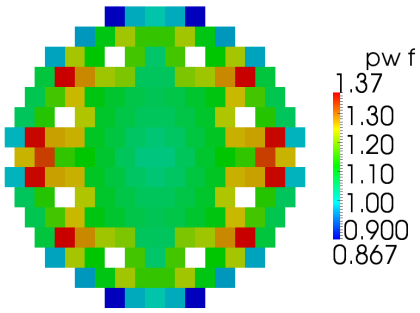

Figure 4.69: Test 4. The core horizontal power distribution factor over the 170 assemblies.

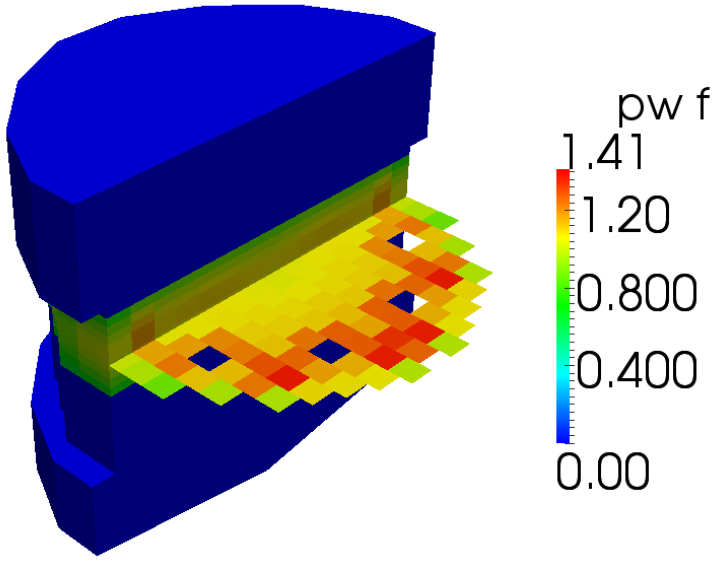

Figure 4.70: Test 4. Core view of power distribution factors.

In this test we study the power distribution shown in Figures 4.69 and 4.70. In this case there are eight control assemblies inside the core which are used to house special control rods (see [1]). In the lower and upper plena we solve the three-dimensional Navier-Stokes and

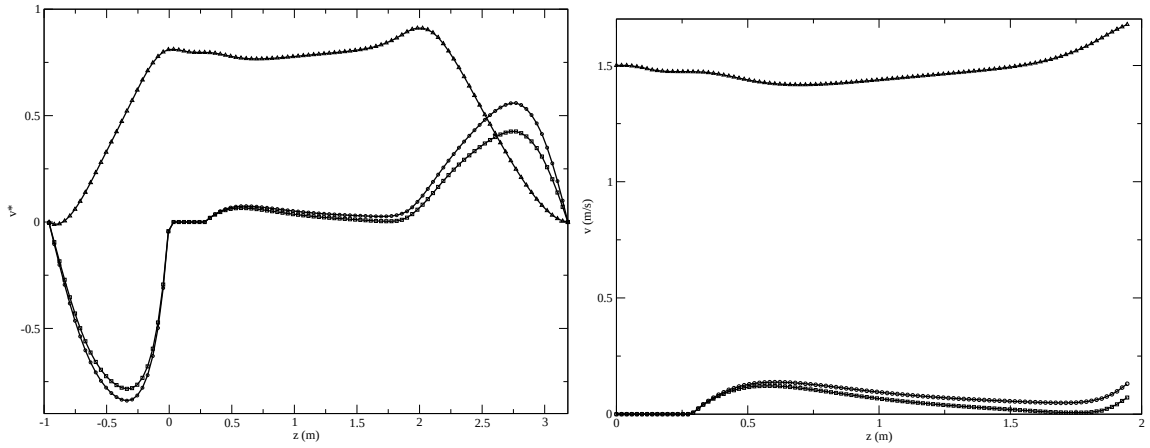

Figure 4.71: Test 4. On the left the x (circle), y (square) and z (triangle) components of the vector $\mathbf{v}^*$ along a vertical line at $x = 1.323$ and $y = 1.2495$. On the right the same profile but for the velocity field in the core region.

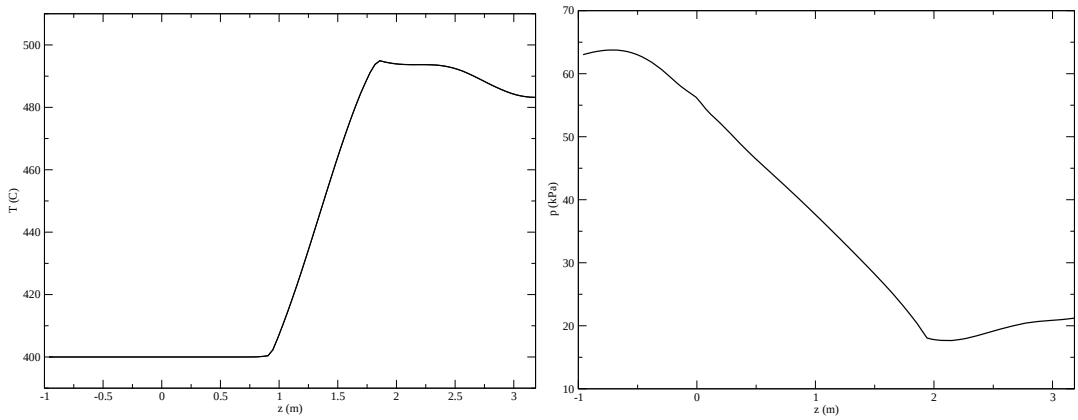

Figure 4.72: Test 4. The temperature and pressure along a vertical line at $x = 1.323$ and $y = 1.2495$.

energy system while in the core we use the usual appropriate model. The total core power is divided over 162 assemblies to have the average specific and linear assembly heat power of

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

and

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

In this test we assume open assemblies. This enforces the velocity field to be parallel to the z-direction only at the core inlet. The u and v velocity components are set to zero only at the core inlet level on the assembly lateral surfaces. As in the other tests, the z component of the velocity field cannot be continuous and therefore we define $\mathbf{v}^*$ as

$$\mathbf{v}^* = \begin{cases} \mathbf{v} & \text{on } \Omega_{lp} \cup \Omega_{up} \\ \frac{\mathbf{v}}{r} & \text{on } \Omega_c \end{cases} \tag{4.16}$$

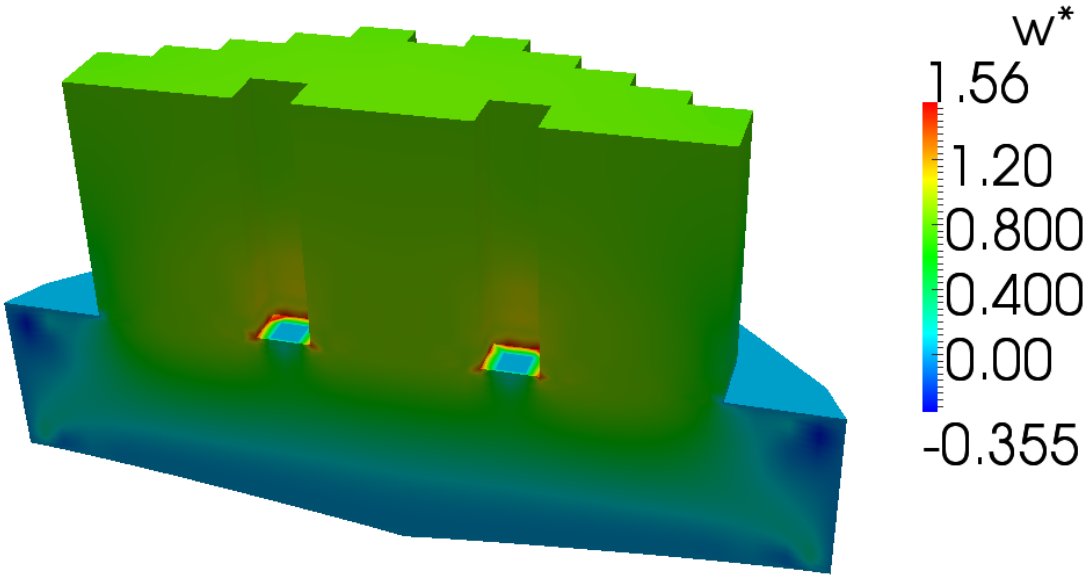

Figure 4.73: Test 4. The w velocity component around the control assemblies.

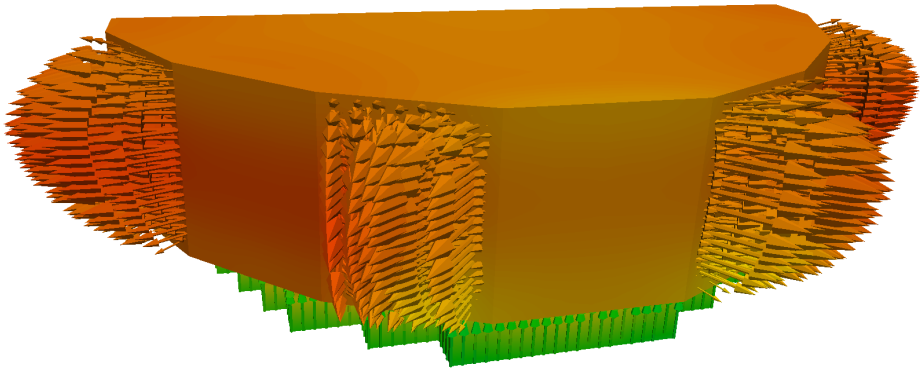

Figure 4.74: Test 4. The velocity field at the upper plenum outlet.

with $r = 0.5408$. In Figures 4.71-4.72 we see the components of the vector $\mathbf{v}^*$, velocity,

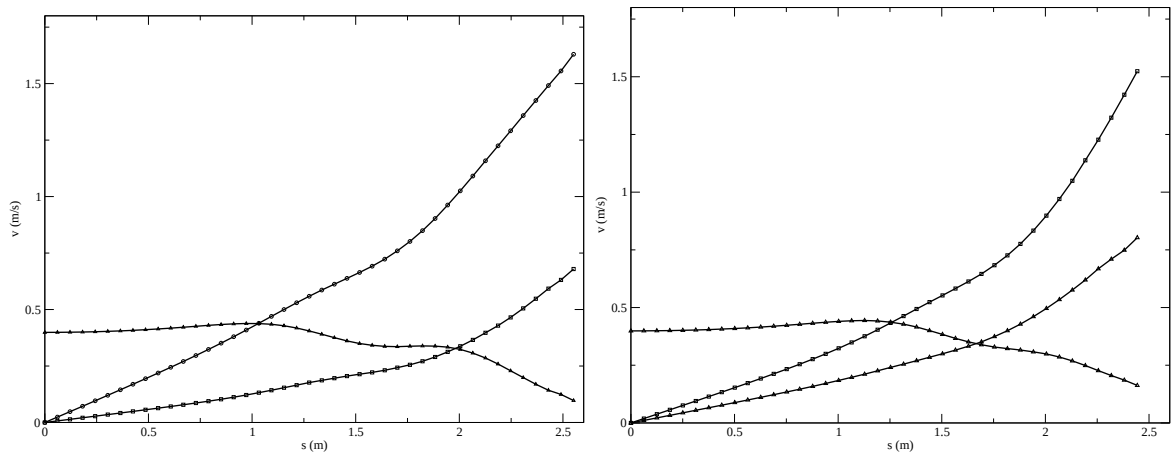

Figure 4.75: Test 4. The x (circle), y (square) and z (triangle) components of the velocity vector $\mathbf{v}$ in the upper plenum for the diagonal line on the right outlet (on the left) and on the left outlet (on the right).

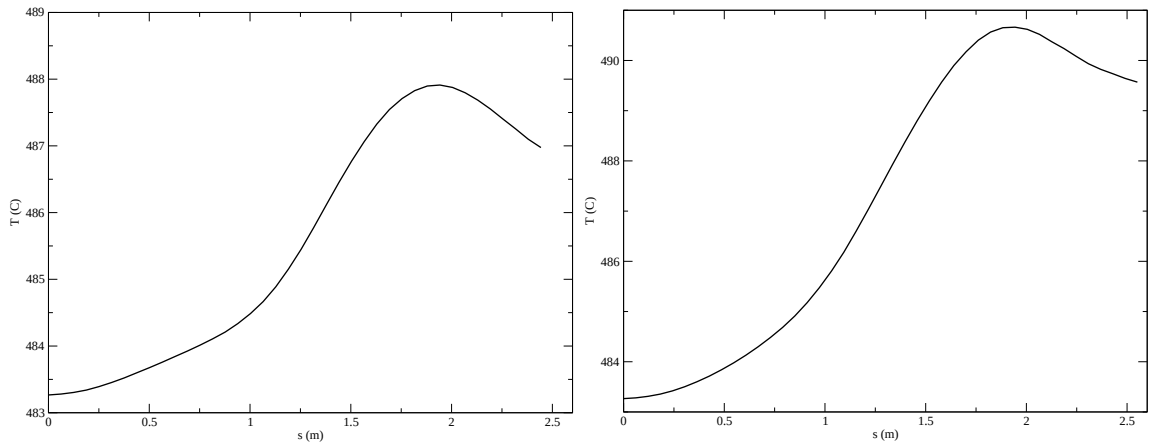

Figure 4.76: Test 4. Temperature in the upper plenum along the diagonal line on the right outlet (on the left) and on the left outlet (on the right).

temperature and pressure fields along a line parallel to the z-axis. The line is located at coordinates $x = 1.323$ and $y = 1.2495$. In Figures 4.71 the x, y and z components of $\mathbf{v}^*$ are shown as lines with circles, squares and triangles respectively. The velocity field in the upper and lower plena matches the vector $\mathbf{v}^*$ but in the core the velocity is scaled by the occupation ratio. The velocity field in the core is shown on the right. Since the core assembly is open none of the three velocity components vanishes. The temperature and pressure on the same vertical line along all the reactor are shown in Figure 4.72 on the left and on the right respectively. The temperature starts to increase inside the upper core region where the fuel assemblies generate heat and reaches the maximum value at the outlet of the core. In the upper plenum the coolant with different temperatures mixes before exiting from the reactor. The pressure decreases mainly inside the lower and upper core regions due to the friction factor inside the sub-assemblies. Figure 4.73 shows the w velocity component around the control assemblies. In Figures 4.74-4.76 we can see the velocity and temperature in the

upper plenum. As in all the other tests the fields are reported along two lines on the upper plenum middle plane which connect the axis point to the middle of the exit window close to the y-axis (left) and to the x-axis (right) respectively.

Temperature

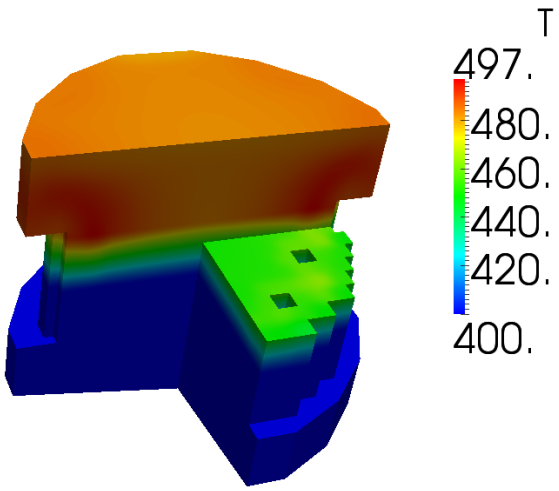

Figure 4.77: Test 4. Overview of the temperature T over the reactor

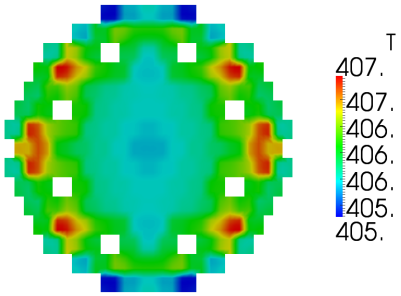

Figure 4.78: Test 4. Temperature distribution over the plane $z = 1.$ (upper core inlet).

In Figure 4.77 there is an overview of the temperature distribution T over the reactor. In Figures 4.78-4.81 we have the temperature distribution over the planes $z = 1.$ (upper core inlet), $z = 1.5$ (upper core), $z = 2$ (upper plenum inlet) and $z = 2.5$ (upper plenum). We

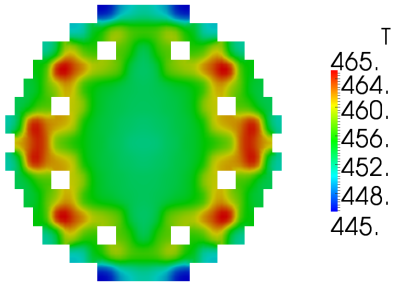

Figure 4.79: Test 4. Temperature distribution over the plane $z = 1.5$ (upper core).

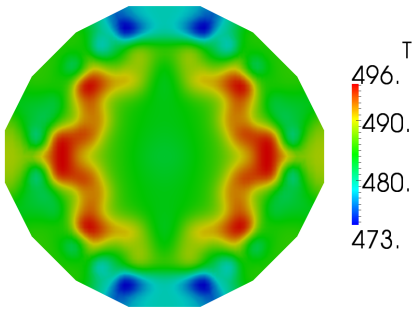

Figure 4.80: Test 4. Temperature distribution over the plane $z = 2.$ (upper plenum inlet).

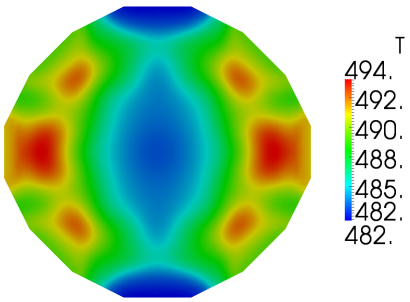

Figure 4.81: Test 4. Temperature distribution over the plane $z = 2.5$ (upper plenum).

note that the maximum temperature over the upper plenum inlet is higher than the maximum temperature over the upper plenum middle horizontal plane.

Pressure

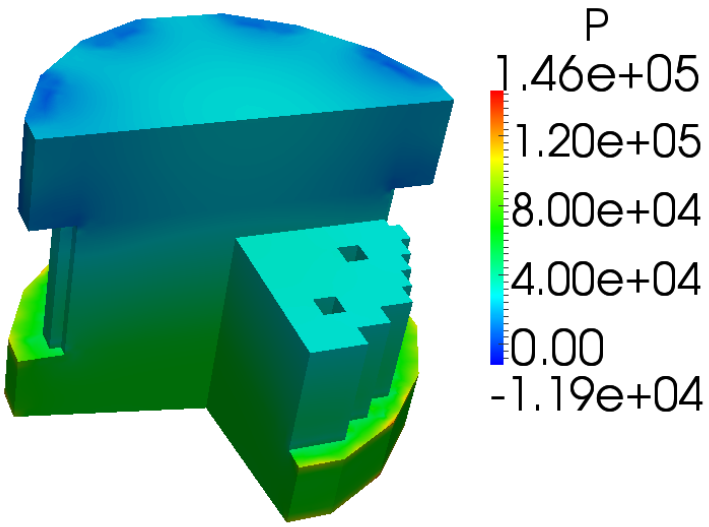

Figure 4.82: Test 4. Overview of the pressure p over the reactor

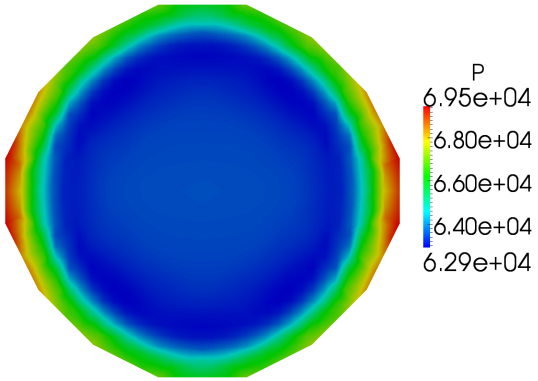

Figure 4.83: Test 4. Pressure distribution over the plane $z = -0.5$ (lower plenum).

In Figure 4.82 there is an overview of the pressure distribution p over the reactor. In Figures 4.83-4.86 we have the pressure distribution over the planes $z = -0.5$ (lower plenum), $z = 0.5$ (lower core), $z = 1$ (upper core inlet) and $z = 2.5$ (upper plenum). The reference pressure is set to zero outside the reactor upper plenum.

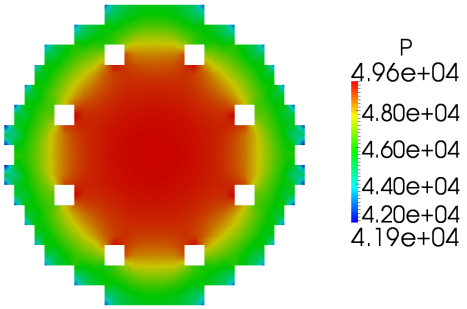

Figure 4.84: Test 4. Pressure distribution over the plane $z = 0.5$ (lower core).

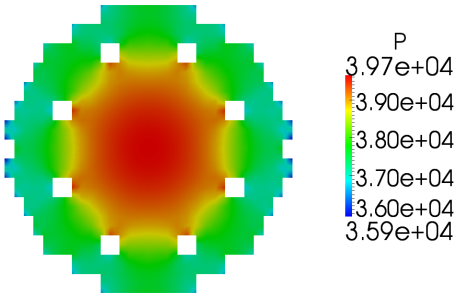

Figure 4.85: Test 4. Pressure distribution over the plane $z = 1.$ (upper core inlet).

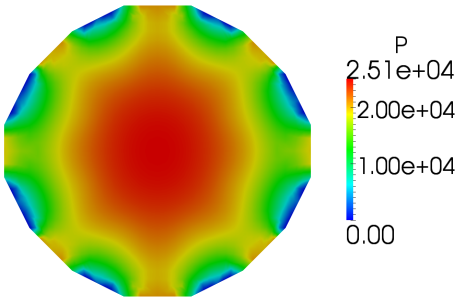

Figure 4.86: Test 4. Pressure distribution over the plane $z = 2.5$ (upper plenum).

Velocity (w-component)

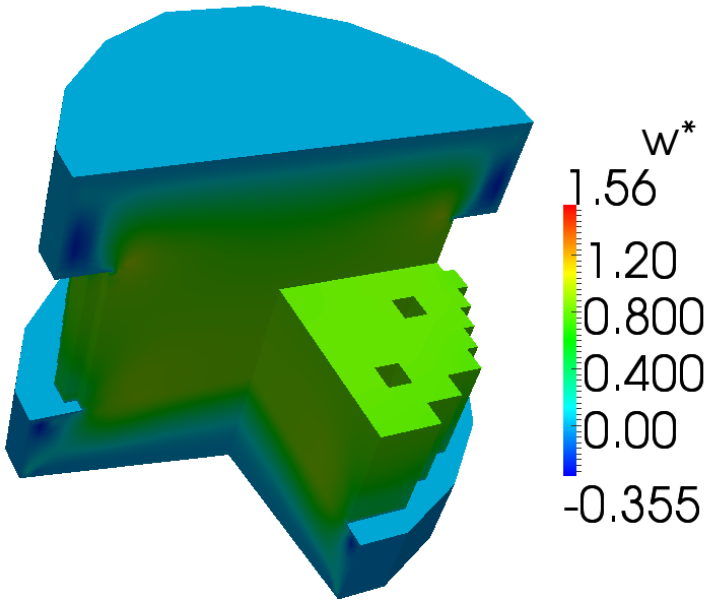

Figure 4.87: Test 4. Overview of the w^* component of the vector $\mathbf{v}^*$ (non-dimensional velocity) over the reactor

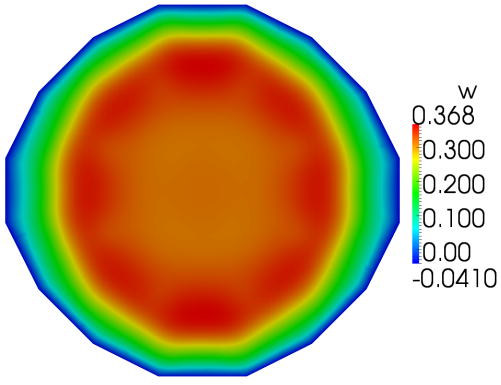

Figure 4.88: Test 4. The velocity component w over the plane $z = -0.5$ (lower plenum).

In Figure 4.87 there is an overview of the w^*-component distribution of the vector field $\mathbf{v}^*$ over the reactor. In Figures 4.88-4.91 we show the w-component distribution over the planes $z = -0.5$ (lower plenum), $z = 0.5$ (lower core), $z = 1$ (upper core inlet) and $z = 2.5$ (upper

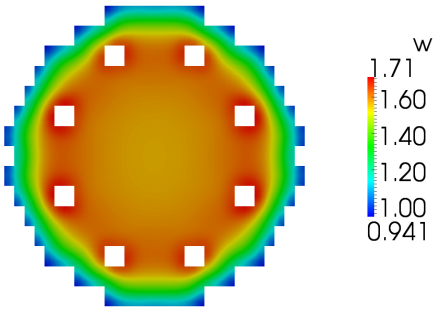

Figure 4.89: Test 4. The velocity component w over the plane $z = 0.5$ (lower core).

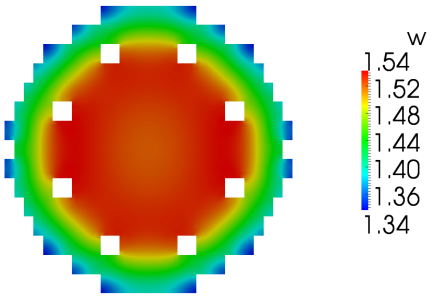

Figure 4.90: Test 4. The velocity component w over the plane $z = 1.$ (upper core inlet).

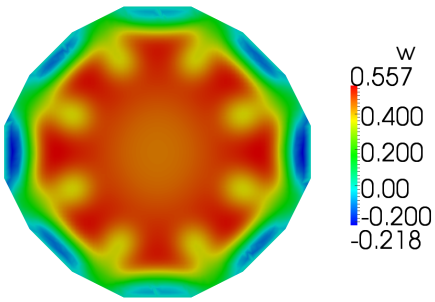

Figure 4.91: Test 4. The velocity component w over the plane $z = 2.5$ (upper plenum).

plenum). We note that the w^*-component is continuous but the w-component of the velocity field scales with the occupation ratio r.

Velocity (u-component)

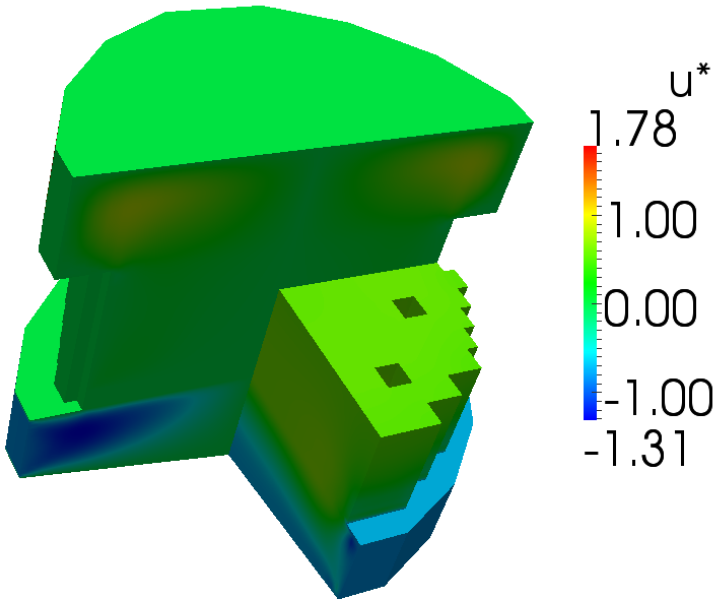

Figure 4.92: Test 4. Overview of the u^* component of the vector $\mathbf{v}^*$ (non-dimensional velocity) over the reactor

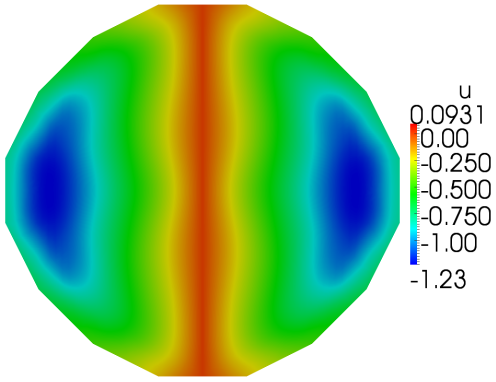

Figure 4.93: Test 4. The velocity component u over the plane $z = -0.5$ (lower plenum).

In Figure 4.92 there is an overview of the u^*-component distribution of the vector field $\mathbf{v}^*$ over the reactor. In Figures 4.93-4.96 we show the u-component distribution over the planes $z = -0.5$ (lower plenum), $z = 1$ (upper core inlet), $z = 1.5$ (upper core) and $z = 2.5$ (upper

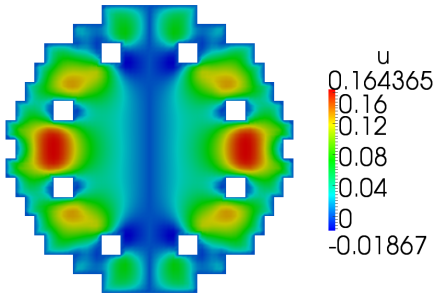

Figure 4.94: Test 4. The velocity component u over the plane $z = 1$. (upper core inlet).

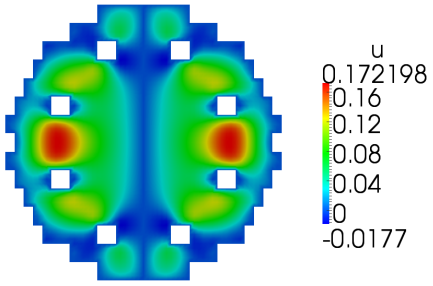

Figure 4.95: Test 4. The velocity component u over the plane $z = 1.5$ (upper core).

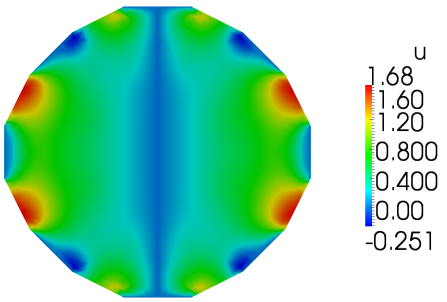

Figure 4.96: Test 4. The velocity component u over the plane $z = 2.5$ (upper plenum).

plenum). We note that the u-component does not vanish in the core.

Velocity (v-component)

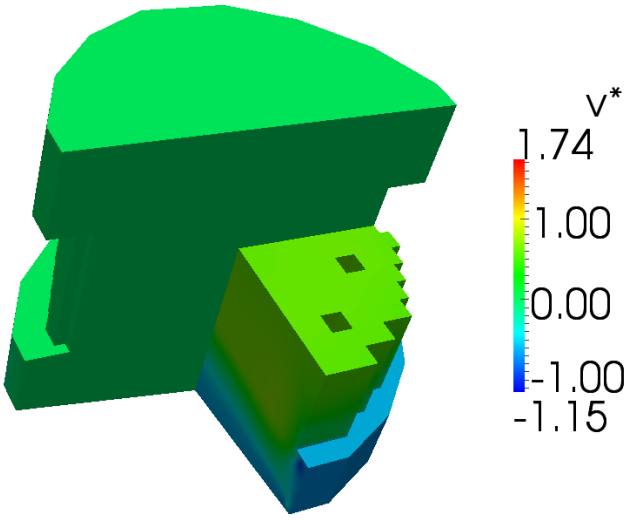

Figure 4.97: Test 4. Overview of the velocity component v^* over the reactor

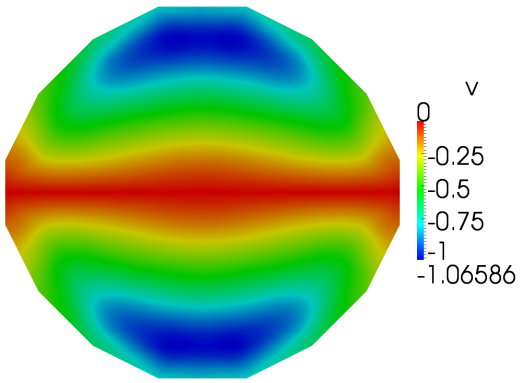

Figure 4.98: Test 4. The velocity component v over the plane $z = -0.5$ (lower plenum).

In Figure 4.97 there is an overview of the v^*-component distribution of the vector field $\mathbf{v}^*$ over the reactor. In Figures 4.98-4.101 we show the v-component distribution over the plane $z = -0.5$ (lower plenum), $z = 1.$ (upper core inlet), $z = 1.5$ (upper core) and $z = 2.5$ (upper plenum). As the u-component, we note that the v-component is not zero in the core and the motion is fully three-dimensional.

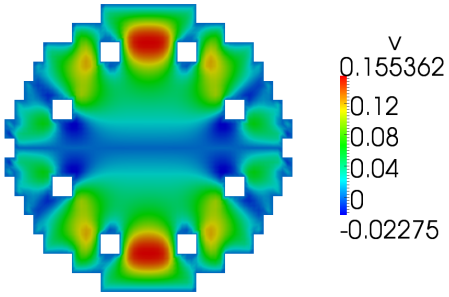

Figure 4.99: Test 4. The velocity component v over the plane $z = 1$. (upper core inlet).

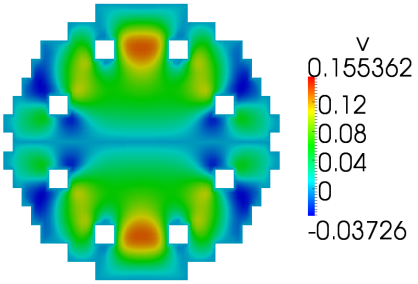

Figure 4.100: Test 4. The velocity component v over the plane $z = 1.5$ (upper core).

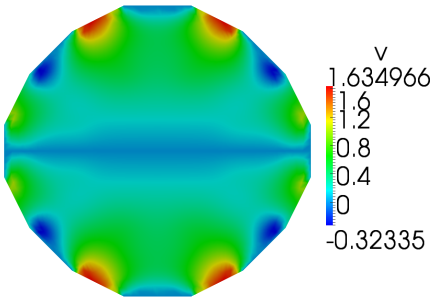

Figure 4.101: Test 4. The velocity component v over the plane $z = 2.5$ (upper plenum).

Bibliography

[1] ELSY Work Program. European Lead-cooled SYstem (ELSY) Project. Technical report, EURATOM, Management of Radioactive Waste, 2006. 103

[2] F.Bassenghi, G.Bornia, A. Cervone and S. Manservisi, *The ENEA-CRESCO platform for simulating liquid metal reactors*, Technical report LIN-THRG 210, (2010) 23

[3] 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, 18, 33, 71, 81

[4] Handbook on Lead-Bismuth eutectic alloy and lead, properties, materials, compatibility, thermalhydraulics and technologies, Chapter 2, OECD-AEN/NEA Report No. 6195, ISBN 978-92-64-99002-9, 2007. 9, 27

[5] K.Litfin, Final report on the fuel bundle in KALLA. Karlsruhe Institute of Technology, Karlsruhe, Germany, 2009. 23

[6] David C. Wilcox, Turbulence Modeling for CFD. DCWIndustries,Inc. LaCanada. California, USA. 14

[7] Namane Mechitoua, Marc Zakiz, Code SATURNE : A Finite Volume Code For The Computation Of Turbulent Incompressible Flow – Industrial Application. Frederic Archambeau, EDF R&D. 27

[8] Code SATURNE 1.3.2 documentation : Theory and Programmer's Guide. 2008, http://www.code-saturne.org. 27

[9] SALOME Documentation, CEA/DEN, EDF R&D, OPEN CASCADE, 2007-2008, http://www.salome-platform.org. 27, 41

[10] FLUENT 2007, FLUENT 6.3 User's Guide, FLUENT Inc., USA, http://www.fluent.com. 27

[11] LASPACK (Linear Algebra Sparse Matrix Package): http://www.mgnet.org/mgnet/Codes/laspack/html/laspack.html. 38

[12] LIBMESH package: http://libmesh.sourceforge.net/. 33

[13] PARAVIEW visualization software: http://www.paraview.org. 35, 47

[14] VTK library: http://www.vtk.org. 47

[15] XDMF library: http://www.xdmf.org. 47

[16] HDF5 library: http://www.hdfgroup.org/HDF5/. 47

[17] PETSC library: http://www.mcs.anl.gov/petsc/petsc-as/. 38