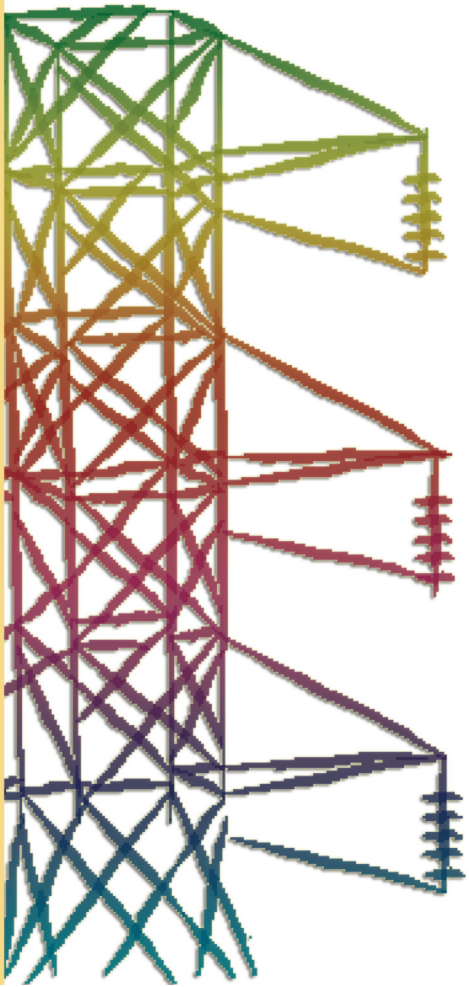
Ente per le Nuove tecnologie,
l'Energia e l'Ambiente

Ministero dello Sviluppo Economico

RICERCA SISTEMA ELETTRICO

A Three-Dimensional CFD Program for the Simulation of the Thermo-Hydraulic Behaviour of a an Open Core Liquid Metal Reactor

Antonio Cervone, Massimo Manservisi



Report RSE/2009/85

Ente per le Nuove tecnologie,
l'Energia e l'Ambiente

Ministero dello Sviluppo Economico

RICERCA SISTEMA ELETTRICO

A Three-Dimensional CFD Program for the Simulation of the Thermo-Hydraulic Behaviour of a an Open Core Liquid Metal Reactor

Antonio Cervone, Sandro Manservisi

Report RSE/2009/85

A THREE-DIMENSIONAL CFD PROGRAM FOR THE SIMULATION OF THE THERMO-HYDRAULIC BEHAVIOUR OF A AN OPEN CORE LIQUID METAL REACTOR

Antonio Cervone, Sandro Manservisi (Università di Bologna)

Dicembre 2008

Report Ricerca 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

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

TECHNICAL REPORT LIN-THRG 108: A THREE-DIMENSIONAL CFD PROGRAM FOR THE SIMULATION OF THE THERMO-HYDRAULIC BEHAVIOUR OF AN OPEN CORE LIQUID METAL REACTOR

Dec 16 2008

Authors: A.Cervone and S.Manservisi
antonio.cervone@mail.ing.unibo.it
sandro.manservisi@mail.ing.unibo.it

Abstract. A thermo-fluid dynamics code with the purpose to investigate three-dimensional pressure, velocity and temperature fields inside nuclear reactors is presented in this work. The code computes pressure,velocity and temperature fields at the coarse fuel assembly level when all the sub-channel details are summarized in parametric coefficients. A 3D CFD model for an open core liquid metal reactor has been implemented on a finite element code and some preliminary tests for generic geometries are reported.

Contents

Chapter 1

Introduction

A full 3D CFD code with the purpose of analyzing the thermal hydraulic behaviour of an open core liquid metal reactor 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 when all the sub-channel details are summarized in parametric coefficients. The solution of the Navier-Stokes system and the energy equation is obtained by using the finite element method. This report consists of three Chapters and one Appendix: Chapter 1 introduces the mathematical model, Chapter 2 describes the CFD code, Chapter 3 evaluates the code performance for some reactor configurations and Appendix A can be used for class references.

In Section 1.1, the full three-dimensional incompressible Navier-Stokes and energy equations are introduced. The numerical simulations take place at a coarse, assembly length level and are linked to the fine, sub-channel level state through transfer operators based on parametric coefficients that summarize local fluctuations. The overall effects between assembly flows are evaluated by using average assembly turbulent viscosity and energy exchange coefficients. In Section 1.2 the computational reactor model is described: the computational mesh, the reactor transfer operators for the reactor in working conditions and the lead thermophysical properties are defined.

The CFD code structure and the main commands are illustrated in different sections of Chapter 2: the code installation, the preprocessing, the run and the postprocessing of the code results can be found in Section 2.1. In Section 2.2 the code directory structure is illustrated. The Section 2.3 is dedicated to the configuration and data files. This section explains how to set the initial and boundary conditions and power and pressure loss coefficients.

In Section 3.1 some basic tests are performed for an open core geometry in order to compare this three-dimensional approach with the standard mono-dimensional one. The final section of Chapter 3 describes different simulations for two 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.

The program described in this report is a class object oriented code and the appendix briefly presents these classes to help the interested reader to quickly modify part of the program. More details and more extensive information can be found in the html/tex/xml code documentation.

1.1 Finite element model

1.1.1 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

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

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

$$\frac{\partial \rho\, C_p\, T}{\partial t} + \nabla \cdot (\rho \mathbf{v} C_p\, T) = \Phi + \nabla \cdot (k \nabla T) + \dot{Q}\,, \tag{1.3}$$

where we can easily recognize the Navier-Stokes and energy equations. 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

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

with $i, j = x, y, z$. In a similar way the tensor $\overline{\mathbf{v}\mathbf{v}}$ is defined as $\overline{\mathbf{v}\mathbf{v}}_{ij} = v_i\, v_j$. The quantity $\mathbf{g}$ denotes the gravity acceleration vector, C_p is the pressure specific heat and k the heat conductivity. $\dot{Q}$ is the volume heat source and Φ the dissipative heat term.

1.1.2 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 the (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

$$\int_\Omega \psi\, \frac{\partial \rho}{\partial t} + \psi\, \nabla \cdot\, (\rho\, \mathbf{v})\, d\mathbf{x} = 0 \quad \forall \psi \in P(\Omega)\,. \tag{1.5}$$

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

$$\begin{aligned} &\int_\Omega \frac{\partial\, \rho\, \mathbf{v}}{\partial t} \cdot \phi\, d\mathbf{x} + \int_\Omega (\nabla \cdot\, \rho \overline{\mathbf{v}\mathbf{v}}) \cdot \phi\, d\mathbf{x} = \\ &\quad \int_\Omega p\, \nabla \cdot \phi\, d\mathbf{x} - \int_\Omega \bar{\tau} : \overline{\nabla \phi}\, d\mathbf{x} + \int_\Omega \rho \mathbf{g} \cdot \phi\, d\mathbf{x} - \qquad \forall \phi \in \mathbf{V}(\boldsymbol{\Omega}) \\ &\quad \int_\Gamma (p \vec{n} - \bar{\tau} \cdot \vec{n}) \cdot \phi\, d\mathbf{s}\,. \end{aligned} \tag{1.6}$$

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 (1.6) is the equation for the evolution of the rate of the 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

$$\begin{aligned}\int_\Omega \frac{\partial\, \rho\, C_p\, T}{\partial t}\, \varphi\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\, \mathbf{v} C_p\, T)\, \varphi\, d\mathbf{x} = \\ \int_\Omega \Phi\, \varphi\, d\mathbf{x} - \int_\Omega k\, \nabla T\cdot\nabla\varphi\, d\mathbf{x} + \int_\Omega \dot{Q}\varphi\, d\mathbf{x} + \qquad \forall\varphi\in\mathbf{H}(\Omega) \\ \int_\Gamma (k\nabla T\cdot\vec{n})\, \varphi\, d\mathbf{s}\,.\end{aligned} \tag{1.7}$$

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

1.1.3 Finite element Navier-stokes system

The pressure space $P(\Omega)$, the velocity space $\mathbf{V}(\Omega)$ and the energy space $H(\Omega)$ in (1.5-1.7) 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 space of linear polynomials for $P_h(\Omega)$ and the finite space of quadratic polynomials for $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$. In this report the domain Ω is discretized always by Lagrangian finite element families with parameter h.

The finite element Navier-Stokes system becomes

a) **Fem incompressibility constraint**

$$\int_\Omega \psi_h \frac{\partial\rho}{\partial t} + \psi_h\, \nabla\cdot\,(\rho\, \mathbf{v}_h)\, d\mathbf{x} = 0 \quad \forall\psi_h\in P_h(\Omega)\,, \tag{1.8}$$

b) **Fem momentum equation**

$$\begin{aligned}\int_\Omega \frac{\partial\, \rho\, \mathbf{v}_h}{\partial t}\cdot\phi_h\, d\mathbf{x} + \int_\Omega (\nabla\cdot\rho\,\overline{\mathbf{v}_h\mathbf{v}_h})\cdot\phi_h\, d\mathbf{x} = \\ \int_\Omega p_h\, \nabla\cdot\phi_h\, d\mathbf{x} - \int_\Omega \bar{\tau}_h : \overline{\nabla\phi_h}\, d\mathbf{x} + \int_\Omega \rho\mathbf{g}\cdot\phi_h\, d\mathbf{x} - \qquad \forall\phi_h\in\mathbf{V}_h(\Omega) \\ \int_\Gamma (p_h\vec{n} - \bar{\tau}_h\cdot\vec{n})\cdot\phi_{\mathbf{h}}\, d\mathbf{s}\,,\end{aligned} \tag{1.9}$$

c) **Fem energy equation**

$$\begin{aligned}\int_\Omega \frac{\partial\rho\, C_p\, T_h}{\partial t}\, \varphi_h\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\mathbf{v}_h C_p\, T_h)\, \varphi_h\, d\mathbf{x} = \qquad \forall\varphi_h\in\mathbf{H}_h(\Omega) \\ \int_\Omega \Phi_h\, \varphi_h\, d\mathbf{x} - \int_\Omega k\, \nabla T_h\cdot\nabla\varphi_h\, d\mathbf{x} + \int_\Omega \dot{Q}_h\, \varphi_h\, d\mathbf{x} + \int_\Gamma k\ \nabla T_h\cdot\vec{n}\, \varphi_h\, d\mathbf{x}\,.\end{aligned} \tag{1.10}$$

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 (1.8-1.10) yields a system of equations which has one equation for each fem basis element.

1.1.4 Two-level finite element Navier-Stokes system

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 $(\hat{p}_h, \hat{\mathbf{v}}_h, \hat{T}_h)$ and the solution spaces by $\widehat{P}_h(\Omega)$, $\widehat{\mathbf{V}}_h(\Omega)$ and $\widehat{H}_h(\Omega)$ respectively.

Momentum equation. 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

$$\int_\Omega NS(p_h, \mathbf{v}_h) \cdot \phi_h(i)\, d\mathbf{x} = 0\,, \tag{1.11}$$

or

$$\begin{aligned}\int_\Omega \frac{\partial\, \rho\, \mathbf{v}_h}{\partial t} \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h}) \cdot \phi_h(i)\, d\mathbf{x} - \\ \int_\Gamma (\bar{\tau}_h \cdot \vec{n} - p_h \vec{n}) \cdot \phi_{\mathbf{h}}(i)\, d\mathbf{s} - \\ \int_\Omega p_h\, \nabla \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega \bar{\tau}_h : \overline{\nabla \phi_h}(i)\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \phi_h(i)\, d\mathbf{x} = 0\end{aligned} \tag{1.12}$$

for all the elements of the basis $\{\phi_h(i)\}_i$ in $\mathbf{V}_h(\Omega)$. The test function ϕ_h must have a small compact support necessary to describe all the channel details and satisfy all the boundary conditions. Now consider the solution $(\hat{p}_h, \hat{\mathbf{v}}_h)$ at the coarse level (fuel assembly level). It is clear that $(\hat{p}_h, \hat{\mathbf{v}}_h)$ 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 $(\hat{p}_h, \hat{\mathbf{v}}_h)$ in (1.12) we have

$$\begin{aligned}\int_\Omega \frac{\partial\, \rho\, \hat{\mathbf{v}}_h}{\partial t} \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\hat{\mathbf{v}}_h \hat{\mathbf{v}}_h}) \cdot \phi_h(i)\, d\mathbf{x} + \\ \int_\Gamma (\hat{p}_h \vec{n} - \bar{\hat{\tau}}_h \cdot \vec{n}) \cdot \phi_{\mathbf{h}}(i)\, d\mathbf{s} - \\ \int_\Omega \hat{p}_h\, \nabla \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega \bar{\hat{\tau}}_h : \overline{\nabla \phi_h}(i)\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \phi_h(i)\, d\mathbf{x} = \\ \int_\Omega P^m_{cf}(\hat{p}_h - p_h, \hat{\mathbf{v}}_h - \mathbf{v}_h) \cdot \phi_h(i)\, d\mathbf{x} + \int_\Omega T^m_{cf}(\mathbf{v}_h, \hat{\mathbf{v}}_h) \cdot \phi_h(i)\, d\mathbf{x}\,,\end{aligned} \tag{1.13}$$

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

$$P^m_{cf}(\hat{p}_h - p_h, \hat{\mathbf{v}}_h - \mathbf{v}_h) = NS(\hat{p}_h - p_h, \hat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.14}$$

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

$$T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = -\nabla \cdot \rho\, \overline{\mathbf{v}_h \mathbf{v}_h} + \nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h} - \nabla \cdot \rho\, \overline{(\widehat{\mathbf{v}}_h - \mathbf{v}_h)(\widehat{\mathbf{v}}_h - \mathbf{v}_h)}\,. \tag{1.15}$$

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. 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.

Since the coarse scale is defined by a set of solutions in $\widehat{P}_h(\Omega)$, $\widehat{\mathbf{V}}_h(\Omega)$ we must write the equation (1.15) with the test functions $(\widehat{\psi}_h, \widehat{\phi}_h)$ in $\widehat{P}_h(\Omega) \times \widehat{\mathbf{V}}_h$. We assume that the finite element space on the fine and coarse grid are embedded, namely $\widehat{P}_h(\Omega) \subset P_h(\Omega)$ and $\widehat{\mathbf{V}}_h(\Omega) \subset \mathbf{V}_h(\Omega)$. This implies that $\widehat{\phi}_h \in \widehat{V}_h(\Omega)$ can be computed as a linear combination of functions in $V_h(\Omega)$. Since these are finite dimensional spaces we have

$$\widehat{\phi}_h(k) = \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\,. \tag{1.16}$$

For finite element spaces constructed with Lagrangian polynomials, the coefficients $a_i(k)$ are $\widehat{\phi}_h(x_i)(k)$ where x_i is a vertex point associated with the basis function $\widehat{\phi}_{hi}$ at the fine level. If we multiply the (1.12) by $a_i(k)$ and add all the equations in $\widehat{\phi}_{hi}$ over the N_k basis functions, after straightforward computations, we have

$$\begin{aligned}
&\int_\Omega \frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x} + \\
&\quad \int_\Gamma (\widehat{p}_h \vec{n} - \bar{\bar{\tau}}_h \cdot \vec{n}) \cdot \sum_{i=1}^{N_k} a_i(k)\phi_{\mathbf{h}}(i)\, d\mathbf{s} - \\
&\quad\quad \int_\Omega \widehat{p}_h\, \nabla \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x} + \int_\Omega \bar{\bar{\tau}}_h : \overline{\nabla \sum_{i=1}^{N_k} a_i(k)\phi_h(i)}\, d\mathbf{x} - \int_\Omega \rho\mathbf{g} \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x} = \\
&\quad\quad\quad \int_\Omega P^m_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x} + \int_\Omega T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) \cdot \sum_{i=1}^{N_k} a_i(k)\phi_h(i)\, d\mathbf{x}\,,
\end{aligned} \tag{1.17}$$

which is

$$\begin{aligned}
&\int_\Omega \frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \\
&\quad \int_\Gamma (\widehat{p}_h \vec{n} - \bar{\bar{\tau}}_h \cdot \vec{n}) \cdot \widehat{\phi}_h(k)\, d\mathbf{s} - \\
&\quad\quad \int_\Omega \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega \bar{\bar{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)}\, d\mathbf{x} - \int_\Omega \rho\mathbf{g} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \\
&\quad\quad\quad \int_\Omega P^m_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega T^m_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x}\,.
\end{aligned} \tag{1.18}$$

If we define the operator $S^m_{cf}(\widehat{p}_h)$ such that

$$\int_\Omega S^m_{cf}(p_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = -\int_\Gamma p_h\, \vec{n} \cdot \widehat{\phi}_h(k)\, d\mathbf{s} \tag{1.19}$$

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

$$\int_\Omega K^m_{cf}(\mathbf{v}_h) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = \int_\Gamma (\bar{\tau}_h \cdot \vec{n}) \cdot \widehat{\phi}_h(k)\, d\mathbf{s} \tag{1.20}$$

then we can write

$$\int_\Omega \frac{\partial\,\rho\,\widehat{\mathbf{v}}_h}{\partial t}\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega(\nabla\cdot\rho\,\overline{\widehat{\mathbf{v}}_h\widehat{\mathbf{v}}_h})\cdot\widehat{\phi}_h(k)\,d\mathbf{x}-$$
$$\int_\Omega \widehat{p}_h\,\nabla\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega\bar{\bar{\tau}}_h:\overline{\nabla\widehat{\phi}_h(k)}\,d\mathbf{x}-\int_\Omega\rho\mathbf{g}\cdot\widehat{\phi}_h(k)\,d\mathbf{x}= \tag{1.21}$$
$$\int_\Omega P^m_{cf}(p_h-\widehat{p}_h,\mathbf{v}_h-\widehat{\mathbf{v}}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega T^m_{cf}(\mathbf{v}_h,\widehat{\mathbf{v}}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+$$
$$\int_\Omega S^m_{cf}(p_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega K^m_{cf}(\mathbf{v}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}\,,$$

which is the equation for the coarse level. 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. In fact we can write for the coarse grid state $(\hat{p}_h,\widehat{\mathbf{v}}_h)$ the equation

$$\int_\Omega \frac{\partial\,\rho\,\widehat{\mathbf{v}}_h}{\partial t}\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega(\nabla\cdot\rho\,\overline{\widehat{\mathbf{v}}_h\widehat{\mathbf{v}}_h})\cdot\widehat{\phi}_h(k)\,d\mathbf{x}-$$
$$\int_\Omega \widehat{p}_h\,\nabla\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega\bar{\bar{\tau}}_h:\overline{\nabla\widehat{\phi}_h(k)}\,d\mathbf{x}-\int_\Omega\rho\mathbf{g}\cdot\widehat{\phi}_h(k)\,d\mathbf{x}= \tag{1.22}$$
$$\int_\Omega R^m_{cf}(p_h,\mathbf{v}_h,\widehat{p}_h,\widehat{\mathbf{v}}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}\,,$$

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

$$\int_\Omega R^m_{cf}(p_h,\mathbf{v}_h,\widehat{p}_h,\widehat{\mathbf{v}}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}= \tag{1.23}$$
$$\int_\Omega P^m_{cf}(\widehat{p}_h-p_h,\widehat{\mathbf{v}}_h-\mathbf{v}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega T^m_{cf}(\mathbf{v}_h,\widehat{\mathbf{v}}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+$$
$$\int_\Omega S^m_{cf}(p_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}+\int_\Omega K^m_{cf}(\mathbf{v}_h)\cdot\widehat{\phi}_h(k)\,d\mathbf{x}\,.$$

Energy equation. We can apply the same procedure at the energy equation. 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

$$\int_\Omega \frac{\partial\rho\,C_p\,T_h}{\partial t}\,\varphi_h(i)\,d\mathbf{x}+\int_\Omega\nabla\cdot(\rho\,\mathbf{v}_h C_p\,T_h)\,\varphi_h(i)\,d\mathbf{x}- \tag{1.24}$$
$$\int_\Gamma k\,(\nabla T_h\cdot\vec{n})\,\varphi_h(i)\,d\mathbf{s}-\int_\Omega\Phi_h\,\varphi_h(i)\,d\mathbf{x}+\int_\Omega k\,\nabla T_h\cdot\nabla\varphi_h(i)\,d\mathbf{x}-\int_\Omega Q_h\,\varphi_h(i)\,d\mathbf{x}=0$$

or

$$\int_\Omega EN(T_h,\mathbf{v}_h)\,\varphi_h(i)\,d\mathbf{x}=0 \tag{1.25}$$

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

If we introduce the coarse level solution $\widehat{T}_h$ in (1.24) we have

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \varphi_h(i)\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h)\, \varphi_h(i)\, d\mathbf{x} - \tag{1.26}$$
$$\int_\Gamma k\, (\nabla T_h\cdot\vec{n})\, \varphi_h(i)\, d\mathbf{s} - \int_\Omega \Phi_h\, \varphi_h(i)\, d\mathbf{x} + \int_\Omega k\, \nabla\widehat{T}_h\cdot\nabla\varphi_h(i)\, d\mathbf{x} - \int_\Omega Q_h\, \varphi_h(i)\, d\mathbf{x} =$$
$$\int_\Omega P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)\, \varphi_h(i)\, d\mathbf{x} + \int_\Omega T^e_{cf}(\widehat{\mathbf{v}}_h, \mathbf{v}_h)\, \varphi_h(i)\, d\mathbf{x}\,,$$

where

$$P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) = EN(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.27}$$

is the energy fine-coarse transfer operator and

$$T^e_{cf}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla\cdot(\rho\, C_p\, \widehat{\mathbf{v}}_h\, \widehat{T}_h) - \nabla\cdot(\rho\, C_p\, \mathbf{v}_h\, T_h) - \nabla\cdot(\rho\, C_p\, (\widehat{\mathbf{v}}_h - \mathbf{v}_h)\, (\widehat{T}_h - T_h))\,. \tag{1.28}$$

We assume that the finite element spaces on the fine level and coarse levels are embedded ($\widehat{H}_h(\Omega) \subset H_h(\Omega)$). This implies that $\widehat{\varphi}_h \in \widehat{H}_h(\Omega)$ can be computed as a linear combination of functions in $H_h(\Omega)$ as

$$\widehat{\varphi}_h(k) = \sum_{i=1}^{N_k} b_i(k)\varphi_{hi}\,. \tag{1.29}$$

If we multiply the (1.26) by $b_i(k)$ and add all the equation in φ_{hi} over the N_k basis functions, after straightforward computations, we have

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} - \tag{1.30}$$
$$\int_\Gamma k\, (\nabla\widehat{T}_h\cdot\vec{n})\, \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega \Phi_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega k\, \nabla\widehat{T}_h\cdot\nabla\widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega Q_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} =$$
$$\int_\Omega P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,.$$

If we define the operator $S^e_{cf}(T_h)$ such that

$$\int_\Omega S^e_{cf}(T_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} = \int_\Gamma k\, (\nabla T_h\cdot\vec{n})\, \widehat{\varphi}_h(k)\, d\mathbf{x}\,, \tag{1.31}$$

then we can write

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega \nabla\cdot(\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} -$$
$$\int_\Omega \Phi_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega k\, \nabla\widehat{T}_h\cdot\nabla\widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega Q_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} = \tag{1.32}$$
$$\int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,,$$

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

$$\int_\Omega R^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h.\mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x} = \int_\Omega S^e_{cf}(T_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \tag{1.33}$$
$$\int_\Omega P^e_{cf}(\widehat{T}_h - T_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h)\ \widehat{\varphi}_h(k)\, d\mathbf{x}\,.$$

Incompressibility constraint. In a similar way we have

$$\int_\Omega (\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \widehat{\mathbf{v}}))\, \widehat{\psi}_h(k)\, d\mathbf{x} = \int_\Omega P^c_{ef}(\widehat{\mathbf{v}}_h - \mathbf{v}_h)\, d\mathbf{x} \tag{1.34}$$

$$\tag{1.35}$$

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

$$R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = R^c_{ef}(\widehat{\mathbf{v}}_h, \mathbf{v}_h) = \int_\Omega \nabla \cdot \rho(\widehat{\mathbf{v}}_h - \mathbf{v})\, \widehat{\psi}_h(k)\, d\mathbf{x}\,. \tag{1.36}$$

1.2 Reactor model

1.2.1 Reactor geometry and mesh generation

The domain Ω is discretized only on the coarse level by standard Lagrangian finite element families which satisfy the standard approximation properties. A typical reactor mesh and its x, y, z view are shown in Figure 1.1. Since the reactor is symmetric along $x = 0$ and $y = 0$ planes we can divide the reactor in 4 parts and compute only one of them. In order to solve the pressure,velocity and energy fields we use the finite space of linear polynomials for $P_h(\Omega)$ and the finite space of quadratic polynomials for $\mathbf{V}_h(\Omega)$ and $H_h(\Omega)$. The mesh is stored in the file `data_in/mesh.in` which is included in the program. Since the computation of such a mesh is demanding on a single machine architecture a simple example cube reactor mesh is also given to quickly test new implementations.
The procedure necessary to generate the `mesh.in` file is the following:
1) The mesh is generated by the Gambit mesh generator (see `http://www.fluent.com/`) and stored in the file `mesh.msh`
2) Two mesh levels are generated by using the libmesh library and the case option of the configuration files (see `http://libmesh.sourceforge.net/`).
3) The two mesh level geometry is stored in the file `data_in/mesh.in` and ready to use.
Since the solver is a multigrid solver, the reactor mesh file has two mesh levels: a coarse level (level 0) and a fine level (level 1). The level 0 is the coarsest mesh necessary to describe the power and pressure loss distributions. In order to avoid the installation of the libmesh library the necessary file for the reactor is given with the program. If one wants to change the mesh is necessary to use and install the open source libmesh library and follow the above steps.

1.2.2 Reactor transfer Operators in working conditions

In order to complete the equation 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

$$P^c_{ef} = 0\,. \tag{1.37}$$

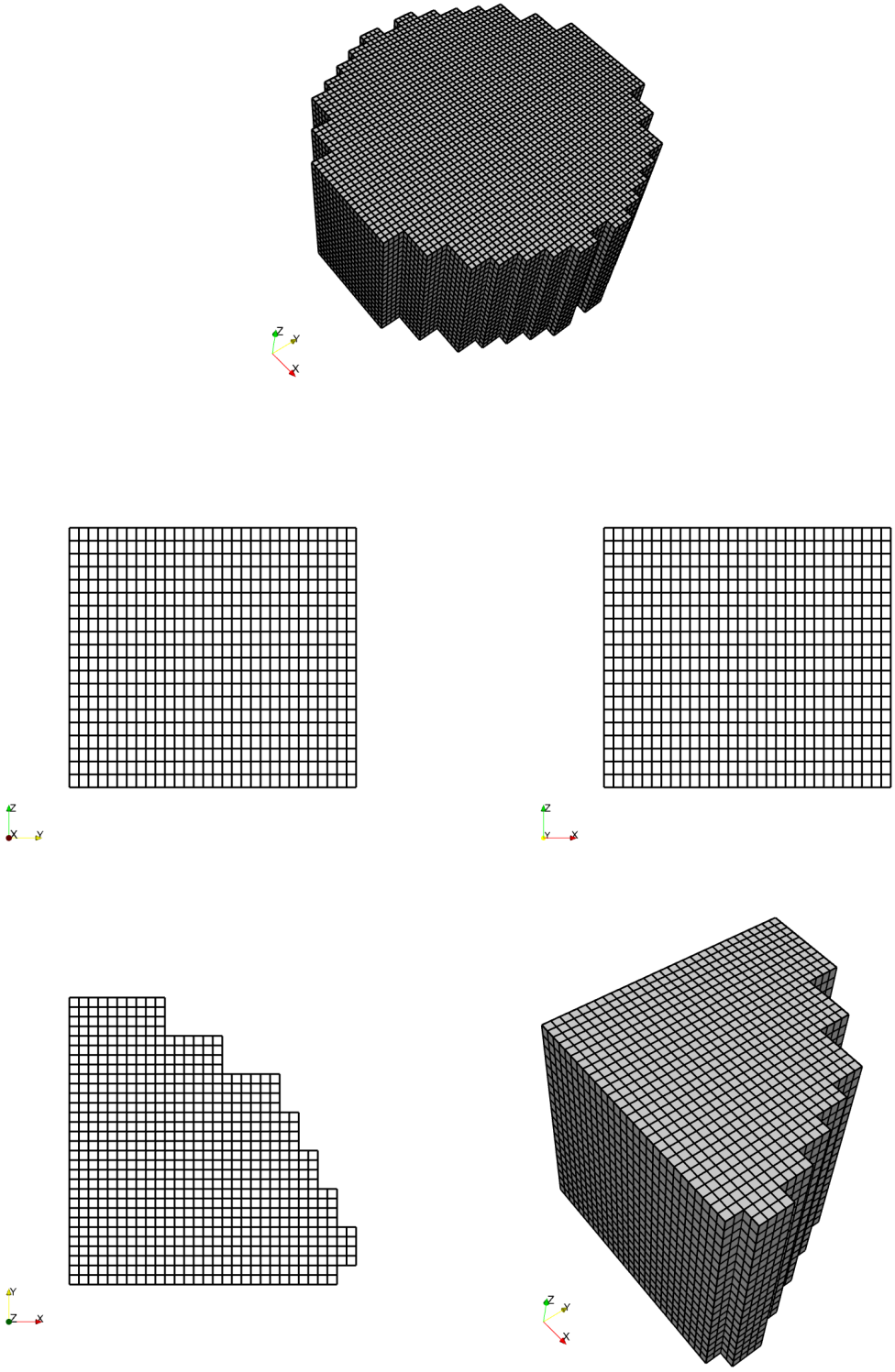


Figure 1.1: x, y, z and 3D view of the computational model

The assumption is exact since

$$P^c_{ef}(\widehat{\mathbf{v}}_h - \mathbf{v}_h) = \nabla \cdot \rho(\widehat{\mathbf{v}}_h - \mathbf{v}_h) \tag{1.38}$$

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

$$T_{cf}(\mathbf{v}_h, \widehat{\mathbf{v}}_h) = \nabla \cdot \bar{\bar{\tau}}^\tau_h \tag{1.39}$$

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

$$\bar{\bar{\tau}}^\tau_h = 2\mu_\tau D(\widehat{\mathbf{v}}_h) \tag{1.40}$$

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

$$\begin{aligned} &P_{cf}(\widehat{p}_h - p_h, \widehat{\mathbf{v}}_h - \mathbf{v}_h) = \\ &\zeta(\mathbf{x})\Big(\frac{\partial\, \rho\, \widehat{\mathbf{v}}_h}{\partial t} \cdot \widehat{\phi}_h(k) \;+ \int_\Omega (\nabla \cdot \rho\, \overline{\widehat{\mathbf{v}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \\ &\qquad \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \bar{\bar{\tau}}_h : \overline{\nabla \widehat{\phi}_h(k)} \;-\; \rho \mathbf{g} \cdot \widehat{\phi}_h(k) - \nabla \cdot \bar{\bar{\tau}}^{eff}\Big) \end{aligned} \tag{1.41}$$

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

$$\bar{\bar{\tau}}^{eff}_h = 2\mu^{eff} D(\widehat{\mathbf{v}}_h)\,. \tag{1.42}$$

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

$$S^m_{cf}(p_h) = 0\,. \tag{1.43}$$

- $K^m_{cf}(v_h)$. 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

$$K^m_{cf}(\mathbf{v}_h) = \zeta(\mathbf{x}) \frac{\rho 2 \widehat{\mathbf{v}}_h |\widehat{\mathbf{v}}_h|}{D_{eq}} \lambda \tag{1.44}$$

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

$$T^e_{cf}(\widehat{T}_h, T_h, \widehat{\mathbf{v}}_h, \mathbf{v}_h) = \nabla \cdot (\frac{\mu_t}{Pr_t} \nabla \widehat{T}_h) \tag{1.45}$$

  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

$$\begin{aligned} &P^e_{cf}(T_h - \widehat{T}_h) = \\ &\zeta(\mathbf{x})\Big(\frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t} + \nabla \cdot (\rho\, C_p \widehat{\mathbf{v}}_h\, \widehat{T}_h) - \Phi_h - Q_h - \nabla \cdot (k^{eff} \nabla \widehat{T}_h)\Big) \end{aligned} \tag{1.46}$$

  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

$$S^e_{cf}(p_h) = W_0 \cos(\frac{\pi z + H_{in}}{H_{out} - H_{in}})\,. \tag{1.47}$$

  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 2.3.6)

1.2.3 Reactor Equations in working conditions

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

$$\rho(T, P) = \rho_0(T) exp(\beta p) \tag{1.48}$$

with $\beta \approx 0$ and define $\rho_{in} = \rho_0(T_{in})$. For this purpose it is more convenient to define a new velocity field $\mathbf{u}_h$ such that

$$\mathbf{u}_h = \mathbf{v}_h \frac{\rho}{\rho_{in}}\,. \tag{1.49}$$

For the reactor model, with vertical forced motion in working conditions, the state variables $(\widehat{\mathbf{u}}, \widehat{p}, \widehat{T})$ are the solution of the following finite element system

a) **Fem incompressibility equation**

$$\int_\Omega \Big(\beta \frac{\partial p_h}{\partial t} + (\nabla \cdot \rho_{in}\, \widehat{\mathbf{u}}_h)\Big)\, \widehat{\psi}_h(k)\, d\mathbf{x} = 0\,. \tag{1.50}$$

b) **Fem momentum equation**

$$\int_\Omega \frac{\partial\, \rho_{in}\, \widehat{\mathbf{u}}_h}{\partial t} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega (\nabla \cdot \rho_{in}\, \overline{\widehat{\mathbf{u}}_h \widehat{\mathbf{v}}_h}) \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \int_\Omega \widehat{p}_h\, \nabla \cdot \widehat{\phi}_h(k)\, d\mathbf{x} + \int_\Omega (\bar{\bar{\tau}}_h + \bar{\bar{\tau}}_h^{\tau} + \bar{\bar{\tau}}_h^{eff}) : \overline{\nabla \widehat{\phi}_h(k)}\, d\mathbf{x} + \int_\Omega \frac{2\rho_{in}\, \widehat{\mathbf{u}}_h |\widehat{\mathbf{v}}_h|}{D_{eq}} \lambda \cdot \widehat{\phi}_h(k)\, d\mathbf{x} - \int_\Omega \rho \mathbf{g} \cdot \widehat{\phi}_h(k)\, d\mathbf{x} = 0 \tag{1.51}$$

b) **Fem energy equation**

$$\int_\Omega \frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t}\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega \nabla \cdot (\rho_{in}\, C_p \widehat{\mathbf{u}}_h\, \widehat{T}_h)\, \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega \Phi_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} + \int_\Omega (k + k^{eff} + \frac{\mu_t}{Pr_t})\, \nabla \widehat{T}_h \cdot \nabla \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega Q_h\, \widehat{\varphi}_h(k)\, d\mathbf{x} - \int_\Omega W_{max}\, r(\mathbf{x})\, \cos\Big(\frac{z + H_{in}}{H_{out} - H_{in}}\Big)\, \widehat{\varphi}_h(k)\, d\mathbf{x} = 0\,. \tag{1.52}$$

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 (1.50-1.52) is equivalent to

$$\beta \frac{\partial p_h}{\partial t} + \nabla \cdot (\rho_{in}\, \widehat{\mathbf{u}}_h) = 0 \tag{1.53}$$

$$\frac{\partial \rho_{in}\, \widehat{\mathbf{u}}_h}{\partial t} + (\nabla \cdot \rho_{in}\, \overline{\widehat{\mathbf{u}}_h \widehat{\mathbf{v}}_h}) = -\nabla \widehat{p}_h + \nabla \cdot (\bar{\bar{\tau}}_h + \bar{\bar{\tau}}_h^{\tau} + \bar{\bar{\tau}}_h^{eff}) - \frac{2\rho_{in}\, \widehat{\mathbf{u}}_h |\mathbf{v}_h|}{D_{eq}} \lambda + \rho \mathbf{g} \tag{1.54}$$

$$\frac{\partial\, \rho\, C_p\, \widehat{T}_h}{\partial t} + \nabla \cdot (\rho_{in} C_p \widehat{\mathbf{u}}_h\, \widehat{T}_h) = \Phi_h + \nabla \cdot (k + k^{eff} + \frac{\mu_t}{Pr_t})\, \nabla \widehat{T}_h + Q_h\, \widehat{\varphi}_h(k) + W_0\, r\, \cos\Big(\frac{z + H_{in}}{H_{out} - H_{in}}\Big)\,. \tag{1.55}$$

Boundary conditions The equations (1.53-1.55) must be completed with the corresponding boundary conditions. On the inlet surface for $z = 0$ we have pressure $p = P_{in}$, temperature $T = T_{in}$ and no tangential component for the velocity field. On the surface $x = 0$ we have symmetry; the y-velocity component v is set to zero and there is no heat flux through this surface, i.e. $\dot{\mathbf{q}} \cdot \mathbf{n} = 0$. In a similar way the surface $y = 0$ is surface of symmetry; the x-velocity component u is zero and there is no heat flux, i.e. $\mathbf{q} \cdot \mathbf{n} = 0$. On the top we have outflow boundary conditions with pressure $p = p_0 = 0$. On the reactor outer walls we impose totally slip boundary conditions and thermal insulation (no heat flux).

1.2.4 Thermophysical properties of liquid metals (lead)

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 (e.g. BREST-300 and BREST-600) 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.

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

$$\rho = (11367 - 1.1944 \times T)\,\frac{Kg}{m^3} \tag{1.56}$$

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

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

$$\mu = 4.55 \times 10^{-04}\, e^{(1069/T)}\, Pa \cdot s\,. \tag{1.57}$$

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

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

$$\alpha_v = 14.77 \times 10^{-6} + 12.20^{-9}(T - 273.16) + 12.18^{-12}(T - 273.16)^2\,\frac{m^3}{K} \tag{1.58}$$

Thermal conductivity The lead thermal conductivity κ is

$$\kappa = 15.8 + 108 \times 10^{-4}\,(T - 600.4)\,\frac{W}{m \cdot K} \tag{1.59}$$

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

$$C_p = 147.3\,\frac{J}{Kg \cdot K}\,. \tag{1.60}$$

Chapter 2

CFD Program

2.1 Introduction

2.1.1 Installation

In order to install the package, choose a directory, and run the following commands

- uncompress: `tar xzvf RMCFD-1.0.tar.gz`;
- go to the directory: `cd RMCFD-1.0`;
- run the configuration script: `./configure`;
- compile the program: `make ex13`.

Now you are ready to run the program (see Section 2.1.3).

2.1.2 Preprocessor and mesh generation

The mesh is generated by the commercial code Gambit (see `http://www.fluent.com`) and the open source libmesh library. The necessary meshes at different levels are enclosed with this package. The generation of the mesh is not argument of this report.

2.1.3 Running the code

Make. The code can be run and compiled by using the make command. The make commands are inside the file `Makefile`. Through the Makefile one can execute several commands

- `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 the code. In order to run the program one must set in the file `config.h`

```
// #define RESTART 100
#define STARTIME 0.
```

and execute the following commands

```
make ex13
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 parameter in the `config.h` file

```
#define RESTART n
#define STARTIME t
```

and then execute

```
make ex13
ex13
```

Method. It is possible to run the code in two different ways specifying the shell parameter `METHOD` before the 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`.

2.1.4 Postprocessing

The output is in vtk format and saved in the `output` directory. The open source paraview program is used. For tutorials and manuals one can see `htpp://www.paraview.org` and `http://www.vtk.org/`.

2.2 Directory structure

2.2.1 Generalities

The program consists of a main directory where the source code is stored and five additional subdirectories. The five subdirectories are:

- `config` is the configuration directory where all the configuration files are stored: `data.h` and `config.h`.
- `contrib` is the contribution directory and stores laspack library and datagen program.
- `data_in` is the data directory. All the data necessary for the simulation must be stored in this directory. Enclosed with the program we have three compressed files containing the test packages. One for each test sample reported above. The tests are
  - A simple hexahedral geometry (in `cube.tar.gz`).
  - The reactor model geometry with no control rod channels (in `RM1.tar.gz`).
  - The reactor model geometry with eight empty control rod channels (in `RM2.tar.gz`).
- `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.
- `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 vtk format and can be read using the software paraview.

2.2.2 Main directory and C++ classes.

The code is written in C++. The main file is
`ex13.C`
which calls all the necessary functions organized in five C++ classes. The five classes are

- 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 `MGSol`, written in the files `MGSolver.C` `MGSolver3DNS.C` and `MGSolver.h`, defines the Navier-Stokes equation solver;
- the class `MGSolT`, written in the files `MGSolverT.C` `MGSolver3DT.C` and `MGSolverT.h`, defines the energy equation solver.

Here are the classes, structs, unions and interfaces with brief descriptions:

2.2.3 Directory `config`

This directory consists of two configuration files

- `config.h` with the configuration variables (solution of Navier-Stokes system, solution of energy system, boundary integration, etc ...)
- `data.h` with constant material properties and numerical parameters.

2.2.4 Directory `contrib`

In this directory you can find two packages

- `laspack` is a linear algebra library for solving large sparse linear systems. For details see `http://www.mgnet.org/mgnet/Codes/laspack/html/laspack.html`.
- `datagen` contains the program that generates the desired volumetric power and pressure loss distribution. For datails see Section 2.3.6.

2.2.5 Directory `data_in`

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 directory `data_in`. Each geometry required a new set of files. The libmesh library is needed for this generation.

All the necessary files are contained in `data_in/`*packagename*`.tar.gz`. In order to expand the compressed file one must set the cursor in the directory `data_in` and then use the command `tar xvf` *packagename*`.tar.gz`. In this directory you should find the following files

- `mesh.in`. The mesh file
- `matrix0.in`. The Navier-Stokes matrix file at level 0
- `matrix1.in`. The Navier-Stokes matrix file at level 1
- `matrixT0.in`. The energy matrix file at level 0
- `matrixT1.in`. The energy matrix file at level 1
- `prol1.in`. The prolongation operator for Navier-Stokes equations

- `prolT1.in`. The prolongation operator for energy equation
- `rest0.in`. The restrictor operator for Navier-Stokes equations
- `restT0.in`. The restrictor operator for energy equation
- `data.in`. The data power and loss pressure factor file (level 1)

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

2.2.6 Directory `fem`

This directory consists of six files with Gaussian integration point values

- `shape1D_0302.in` with 1-dim Gaussian points for integration of a fem Edge2 linear element
- `shape1D_0302.in` with 1-dim Gaussian points for integration of a fem Edge3 linear element
- `shape2D_0904.in` with 2-dim Gaussian points for integration of a fem Quad4 linear element
- `shape2D_0909.in` with 2-dim Gaussian points for integration of a fem Quad9 quadratic element
- `shape3D_2708.in` with 3-dim Gaussian points for integration of a fem Hex8 linear element
- `shape3D_2727.in` with 3-dim Gaussian points for integration of a fem Hex27 quadratic element

2.2.7 Directory `output`

This is the directory where the results are printed. In order to analyze the data with paraview viewer you can open different file

- `mesh.0.vtu` contains only the 3D geometric mesh. It is used to check the input geometry.
- `boundary.0.vtu` contains the boundary geometry. With this file you can visualize and check boundary conditions.
- `Out.`n`.vtu` contains the pressure, temperature and velocity solution at the time step n.
- `time.pvd` contains the xml list of all time steps. If you load this file inside paraview you can visualize the time sequence of the solution.

2.3 Configuration, data and parameter setting

Enclosed with the program there are three sample outputs. One for each test reported here. The tests are as follows

- a) `cube.tar.gz` a simple lightweight hexahedral reactor for quick testing.
- b) `rm1.tar.gz` a test reactor without the rod control channel, i.e. the rod control channel is assumed to be full of coolant.
- c) `rm2.tar.gz` a test reactor with eight empty rod control channels.

In order to run a total new computation the following steps are necessary

- Untar the archive in the directory `data_in` (`tar xzvf packagename.tar.gz`)
- Set the configuration file `data_in/config.h`
- Set the physical properties in `data_in/data.h`
- Set the boundary conditions in the files `MGSolver3DNS.C` and `MGSolver3DT.C`
- Set the additional reactor parameters at the top of the files `MGSolver3DNS.C` and `MGSolver3DT.C`

Standard configuration for the mentioned tests is provided with the packages.

2.3.1 Configuration

The configuration file is `config/config.h`. An example of configuration file can be found in A.3. Options are set via define (`#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 define 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, shown in Table 2.3.1, are important for this code:

- `DIM2`. With this option you need to set the dimensions of the problem as in Table 2.3.1. For RM reactor and test cube reactor use always 3D option, so the 2D option should be eliminated, i.e.
  `// # define DIM2 1`

help	command	option	description
3D active	`//#define DIM2`	1	3D simulation
Bound. integr. active	`//#define BOUNDARY`	1	Boundary integration
NS equation active	`#define NS_EQUATIONS`	1	Solve the Navier-Stokes eqs
Energy equation active	`#define T_EQUATIONS`	1	Solve the Energy eqs
Restart time	`#define RESTARTIME`	1.1	Initial time set to $t = 1.1$
Restart active	`#define RESTART`	10	Restart from file label 10
Case active	`#define GENCASE`	1	Generate the case files
libemesh active	`#define LIBMESHF`	1	linking with libMesh lib
GMV graphics active	`#define OUTGMV`	1	output with GMV
Print step	`#define PRINT_STEP`	10	Print each 10 steps
Info active	`#define PRINT_INFO`	1	Print info after function execution
LX dimension	`#define LX`	2.	Set the x-dimension LX =2
LY dimension	`#define LY`	2.	Set the y-dimension LY =2
LZ dimension	`#define LZ`	2.	Set the z-dimension LZ =2

Table 2.1: Options in config.h file

- `BOUNDARY`. The boundary option is necessary for integration on boundary. Since we have a pressure condition on the boundary (reactor bottom and top) this option is necessary, i.e.
  `# define BOUNDARY 1`

- `NS_EQUATIONS, T_EQUATIONS`. If the system has the $(p, \mathbf{u})$ state then you must solve only the Navier-Stokes equations. If the system has the $(p, \mathbf{u}, T)$ state then you must solve the Navier-Stokes equation and the energy equation. For RM 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 ($t = 0.$) with initial conditions as in section 2.3.4 one must set
  `#define RESTARTIME 0.`
  `// #define RESTART 10 .`

- `GENCASE, LIBMESHF`. In order to avoid the installation of the libMesh library 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.`
  For this option you must install the libmesh library. See `http://libmesh.sourceforge.net/` for details.

- `GMV`. This set all the output file in the GMV format (see `http://laws.lanl.gov/XCM/gmv/GMVHome.html`). This is not the case for

this version where all the output files are in vtk format. Data analysis is carried out by Paraview (see http://www.paraview.org/). We set
`// #define GMV 1`

- `PRINT_STEP`. With this option the program prints every `PRINT_STEP` iterations. You must edit this option to print every 10 step as
`#define PRINT_STEP 10`

- `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 and set
`// #define PRINT_STEP 1`
during optimized execution.

- `LX, LY, LZ`. These are parameters useful for defining the dimensions of the reactor. In this program they are used only in setting the boundary conditions. The geometry is not affected by this parameters if the geometry is read from an external file. In our case the geometry is read from the file `mesh.in`. We set
```
#define LX 2.
#define LY 2.
#define LZ 2.
```
in order to have `LZ`$= 2m$ as a reference length of the reactor (see section 2.3.3)

2.3.2 Data

The configuration file is `config/data.h`. An example of data file can be found in section A.3.3. The data values are defined by using the define command (`#define` command in C or C++). For example if we see in the file `config/data.h` a define as
`# define RHO0 10000`
then the density `RHO0` takes the value of 10000. The units are always in agreement with the international system.

The parameters to set from this file are as follows:

- `ND_TIME_STEP ,TIME,N_TIME_STEPS`. This values are the values of the time step, the initial time and the number of time steps. In order to start at $t = 0$ and perform 100 time steps of length $\Delta t = 0.01$ one must set
```
#define ND_TIME_STEP 0.01
#define TIME 0.0
#define N_TIME_STEPS 100.
```

- `Uref,Lref,T_ref,RHOref`. These are the reference values of velocity, length, temperature and density. The pressure reference is `RHOref Uref Uref`. We set `T_ref`$= 1000$ in order to have a temperature in kK (kilo Kelvin) and `RHOref`$= 100000$ in order to have the pressure in $0.1MPa$. All these for graphical purpose in Paraview. Therefore in this case one sets

command	option	description
`#define Uref`	1.	Reference velocity
`#define Lref`	1.	Reference length
`#define T_ref`	1000.	Reference temperature
`#define RHOref`	100000.	Reference density
`#define T_IN`	673.15	Inlet temperature
`#define P_IN`	36000.	Inlet pressure
`#define RHO0`	11367.0	density at T^{*1}
`#define MU0`	.0022	viscosity at T^{*2}
`#define KAPPA0`	15.8	conductibility at T^{*3}
`#define BP0`	147.3	conductibility at T^{*4}
`#define KOMP0`	0.	compressibility at T^{*5}
`#define ND_TIME_STEP`	0.05	Time step Δt
`#define TIME`	0.0	Initial time t
`#define N_TIME_STEPS`	100	Number of time steps
`#define QHEAT`	1.15e+8	heat volume density source
`#define DIRGX`	0.	gravity x-direction
`#define DIRGY`	0.	gravity y-direction
`#define DIRGZ`	0.	gravity z-direction

Table 2.2: Parameters in `data.h` file

command	value	description
`#define ILUINIT`	100	ILU decomposition each 100 steps
`grid0`	0	bottom grid level
`gridn`	1	top grid level
`MaxIter`	15	Iteration multigrid (max)
`Eps`	1.e-6	residual accuracy
`MLSolverId`	`MGIterId`	Multigrid type
`Gamma`	2	Multigrid cycle (1=V cycle 2=W Cycle)
`RestrType`	1	restriction operator(1. simple 2. weighted)
`Nu1`	8	Number of pre-smoothing iterations
`Nu2`	8	Number of post-smoothing iterations
`Omega`	0.98	Relaxation parameter for smoothing
`SmoothProc`	`GMRESIter`	Smoothing method
`PrecondProc`	`ILUPrecond`	Preconditioner
`NuC`	40	Number of iterations on coarsest grid
`OmegaC`	0.98	Relaxation parameter on coarsest grid
`SolvProc`	`GMRESIter`	Coarsest level Iterative method
`PrecondProcC`	`(PrecondProcType)NULL`	Coarsest level preconditioner

Table 2.3: Parameters for multigrid in `data.h` file

```
#define Uref 1.
#define Lref 1.
#define T_ref 1000.
```

```
#define RHOref 100000.
```

- `T_IN, P_IN`. This are the temperature and the pressure at the inlet of the reactor. If we want a a loss of pressure across the reactor of $36000Pa$ and a temperature of $673.15K$ one must set
  ```
  #define T_IN 673.15
  #define P_IN 36000.
  ```

- `RHO0,MU0, KAPPA0, CP0`. The value `RHO0` of the lead density at temperature $T^{*1} = 0K$ is set to 11367, the value `MU0` of the lead viscosity to 0.0022, the value `KAPPA0` of the conductibility at temperature $T^{*3} = 673.15K$ to 15.8 and the value `CP0` of the pressure specific heat to 147.3. In our case we set
  ```
  #define RHO0 11367.0
  #define MU0 .0022
  #define KAPPA0 15.8
  #define CP0 147.3.
  ```
  The temperature dependence is set inside the single files: `MGSol3DT.C` (energy equation) and `MGSol3DNS.C` (Navier-Stokes system). For details see section 2.3.5.

- `QHEAT`. The average volumetric heat source for the reactor is set by this parameter as
  `#define QHEAT 1.15e+8.`
  The sinusoidal shape and other source factors are set directly inside the file `MGSol3DT.C`. The power factor distribution of the assemblies is in the file `data.in` in the directory `data_in`. See section 2.3.6.

- `DIRGX, DIRGY, DIRGZ`. Gravity can be set by using these parameters. If one wants gravity (ρ**g**) along the z-axis one must set
  ```
  #define DIRGX 0.
  #define DIRGY 0.
  #define DIRGZ 1.
  ```
  All the value are in $g = 9.81m/s^2$ units.

Among the solution methods one can choose one of these methods

```
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 and the values shown in Table 2.3.2.

2.3.3 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 MGSol::GenBc(const unsigned int Level)` in the file `MGSolver3DNS.C`. If you open the mentioned file you can find

```
/// Here is the space for user code contribution
// ******************************************
// boundary conditions box
if (zp < 0.001) { // top side (inlet)
  bc[Level][dof_u]=0; bc[Level][dof_v]=0;
  // inlet pressure p=p_in
  if (i<n_p_dofs)  bc[Level][idx_dof[k+3*n_nodes]]=0;
}
if (zp > LZ-0.001) {// bottom side (outlet)
  bc[Level][dof_u]=0; bc[Level][dof_v]=0;
  // outlet pressure p=0
}
//symmetry
if (xp < 0.001) { // side 1
  bc[Level][dof_u]=0;
}
if (xp > LX-0.001) { // side 3
  bc[Level][dof_u]=0;
}
if (yp < 0.001) { // side2
  bc[Level][dof_v]=0; bc[Level][dof_u]=0;
}
if (yp > LY-0.001) { // side4
  bc[Level][dof_v]=0; bc[Level][dof_u]=0;
}
// ************************************
```

The vertex point (xp, yp, zp) can be used to set all the necessary boundary conditions. The face middle point is also provided as (xm, ym, zm).

For boundary conditions of the energy equation one must edit the function `void MGSol::GenBc(const unsigned int Level)` in the file `MGSolver3DT.C`. If you open the mentioned file you may find

```
/// Here is the space for user code contribution
```

```
// ******************************************
// boundary conditions on a reactor
if (zp < 0.001)   bc[Level][dof_u]=0;
// ******************************************
```

The same points are provided also for this equation.

2.3.4 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::GenSol(const unsigned int Level)` inside the file `MGSolver3DNS.C`. If you open the mentioned file you may find the initial solution $\mathbf{u} = 0$ and p which changes linearly from `P_IN` to zero over the z-axis. Therefore in the appropriate part of the function you have

```
// ************************
 /// Here is the space for user contribution
 u_value =0.;
 w_value =0.;
 p_value=(P_IN/(RHOref*Uref*Uref))*(LZ-zp)/LZ;
 u_value=0.;
 // ************************
```

Initial Energy solution. If one wants to set the initial solution in temperature one must edit the function `void MGSolT::GenSol(const unsigned int Level)` inside the file `MGSolver3DT.C`. If you open the mentioned file you can find a constant initial solution `T=T_IN/Tref` over all the domain. The inlet temperature `T=T_IN` and the reference temperature `T=Tref`, are defined in the file `config/data.h`. The reference temperature is actually $Tref = 1000$ in order to have temperature values in the range $0 - 1kK$ (mainly for graphical purposes). In the user area of the file `MGSolver3DT.C` you will find

```
// ************************
/// Here is the space for user contribution
u_value =T_IN/Tref
// ************************
```

2.3.5 Physical property dependence on 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 is necessary to change the inline functions at the top of the followings:
- `MGSolver3DNS.C` for the momentum equations; here are defined $\rho = \rho(T)$ and $\nu = \nu(T)$
- `MGSolver3DT.C` for the energy equation; here are defined $\rho = \rho(T)$, $\kappa = \kappa(T)$ and $C_p = C_p(T)$.

Furthermore, you have to set `#define RT 1` in `config/data.h` in order to switch on temperature dependence.

2.3.6 Power distribution and pressure loss distribution

The power distribution and the pressure loss distribution are set in the file `data_in/data.in`. The file 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-coordinate, the y-coordinate, the z-coordinate, the value of the power distribution factor and the value of the pressure loss factor. The file can be generated by a program that is enclosed in the directory `contrib/datagen`. The direct editing is not very easy since there are at the level 1 more than 15000 elements. The file should be completed at all mesh levels. The procedure to have a new power distribution is as follows:

- delete the file `data_in/data.in`
- run the code with no `data_in/data.in` file. Then a new `data_in/data.in` is generated and the code stops.
- edit the values of power distribution or
- copy the file `data_in/data.in` in `contrib/datagen/data.orig`
- edit `contrib/datagen/datagen.H` inserting power distribution factors as in Figure 3.3. Power distribution from Figure 3.3 is set as follows

```
#define A1_1 1.05
#define A1_2 1.03
#define A1_3 1.02
#define A1_4 1.02
#define A1_5 1.04
#define A1_6 1.17
#define A1_7 1.12

#define A2_1 1.05
#define A2_2 1.03
#define A2_3 1.02
#define A2_4 1.02
#define A2_5 1.12
```

```
#define A2_6 1.12
#define A2_7 1.03
#define A2_8 0.81

#define A3_1 1.04
#define A3_2 1.04
#define A3_3 1.02
#define A3_4 1.10
#define A3_5 CONTROL_ROD
#define A3_6 0.96
#define A3_7 0.87

#define A4_1 1.04
#define A4_2 1.03
#define A4_3 1.03
#define A4_4 1.04
#define A4_5 1.10
#define A4_6 0.97
#define A4_7 0.86

#define A5_1 1.02
#define A5_2 1.11
#define A5_3 1.10
#define A5_4 1.15
#define A5_5 1.01
#define A5_6 0.85

#define A6_1 1.
#define A6_2 1.06
#define A6_3 CONTROL_ROD
#define A6_4 0.99
#define A6_5 1.07
#define A6_6 0.78

#define A7_1 1.04
#define A7_2 0.95
#define A7_3 0.98
#define A7_4 0.81

#define A8_1 0.94
#define A8_2 0.89
#define A8_3 0.74
```

- run `datagen` program in the directory `contrib/datagen`.

The power distribution in Figure 3.3 is enclosed with the provided package. The pressure loss distribution enclosed in the code is constant and equal to 1.

Chapter 3

Tests

3.1 Monodimensional test (test1)

3.1.1 Monodimensional equations

In steady working conditions with constant properties (see Table3.1.2) and velocity in the z-direction the (1.50-1.52) becomes
a) **Incompressibility constraint**

$$\frac{\partial \widehat{w}_h}{\partial z} = 0 \tag{3.1}$$

b) **Momentum equation**

$$\frac{\partial \widehat{p}_h}{\partial z} = \frac{2\rho \widehat{w}^2}{D_{eq}} \lambda \tag{3.2}$$

b) **Energy equation**

$$\rho\, C_p \widehat{w}_h \frac{\partial \widehat{T}_h}{\partial z} = \kappa \frac{\partial^2 \widehat{T}_h}{\partial z^2} + \dot{q}^{'''}\, r\,. \tag{3.3}$$

These equations can be solved and therefore the solution can be used to test the code in this trivial case.

3.1.2 Monodimensional analytic test

The problem (3.1-3.3) 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

$$\widehat{w}_h = const \tag{3.4}$$

$$\widehat{p}_{in} - \widehat{p}_{out} = \frac{2\rho \widehat{w}^2}{D_{eq}} \lambda(\widehat{w}_h)\, H \tag{3.5}$$

$$\rho\, C_p \widehat{w}_h \left(\widehat{T}_{out} - \widehat{T}_{in}\right) = \dot{Q}\, r \left(H_{out} - H_{in}\right), \tag{3.6}$$

properties	value
ρ	$(11367 - 1.1944 \times 673.15) = 10562$
μ	0.0022
D_{eq}	0.0129
κ	$15.8 + 108 \times 10^{-4}\,(673.15 - 600.4) = 16.58$
C_p	147.3
r	0.545

Table 3.1: Properties at T=673.15K

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 (w_h, p_h, T_h) we set each property to be constant (see Table 3.1.2).

Velocity. The velocity must be solved iteratively. At the step 0 we guess the solution as $w = 1.51m/s$. Then the Reynolds number gives

$$Re = \frac{\rho\, w\, D_{eq}}{\mu} = 10562 \times 1.51 \times 0.0129/0.0022 = 93526 \tag{3.7}$$

and the friction coefficient yields

$$\lambda = \frac{0.079}{Re^{0.25}} = 0.004517\,. \tag{3.8}$$

The velocity is then

$$w = \sqrt{\frac{2\, P_I N D_{eq}}{\rho\, 4 * \lambda L Z}} = \sqrt{\frac{2\, 36000 \times 0.0129}{10562 *\ 4 * 0.0045174 \times 2.}} = 1.5598\,. \tag{3.9}$$

Iteratively we find $w = 1.567m/s$. Each step is recorded in Table 3.2.

step	guess w	Re	λ	w
1	1.5100	93525	0.004517	1.559
2	1.559	96611	0.004481	1.566
3	1.566	97004	0.004476	1.567
4	1.567	97053	0.004476	1.567

Table 3.2: Iterative velocity computation.

Temperature. The temperature can be computed as

$$T_{out} - T_{in} = \frac{\dot{Q}}{r\, \dot{m}\, c_P\, (H_{out} - H_{in})} = \frac{1.14591 \times 10^8}{0.548 \times 1.567 \times 147.3} = 85.76K$$

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

$$T_{out} = T_{in} + 85.765 = 758.9K\,.$$

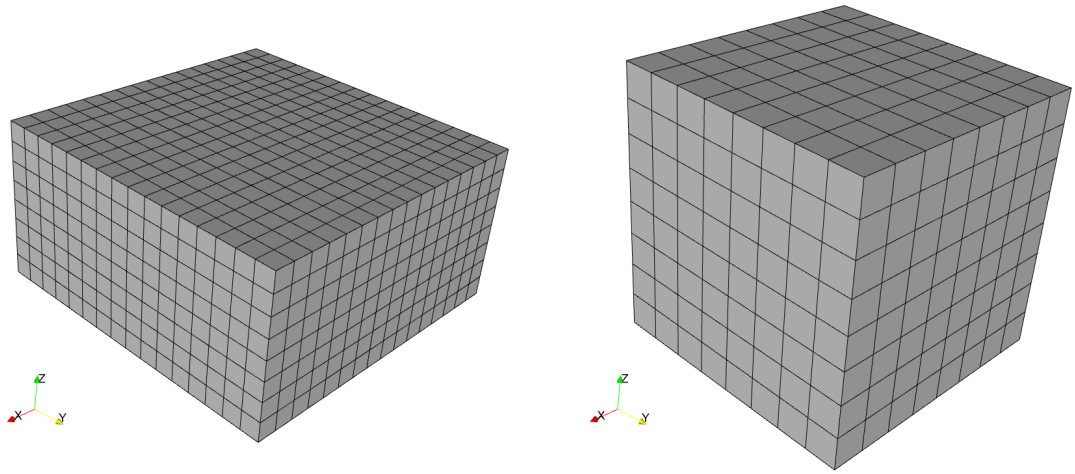


Figure 3.1: Test1: Full reactor and computational domain.

3.1.3 Simulations with constant axial power distribution (test1)

With constant power distribution along the z-axis we obtain a value of velocity in full agreement with (3.1.2). The geometry is shown in Figure 3.1 and packaged in `data_in/cube.tar.gz`. The main options for the `config/config.h` file are

```
#define BOUNDARY 1
#define NS_EQUATIONS  1
#define T_EQUATIONS 1
#define RESTARTIME 0.
#define PRINT_STEP 2
```

The data are set as following (`config/data.h`)

```
// time
#define ND_TIME_STEP  0.01
#define TIME          0.0
#define N_TIME_STEPS  500

// Reference quantities
#define Uref      1.
#define Lref      1.
#define Tref      1000.
#define RHOref    100000.

// Inlet
#define P_IN      36000.
#define T_IN      673.15
```

```
// fluid properties
#define RHO0      11367.0
#define MU0       .0022
#define NUT       .0
#define KAPPA0    15.8
#define CP0       147.3

// Heat source
#define QHEAT     1.14591e+8

//  temperature function
#define RT        0.

// Gravity
#define DIRGX     0.
#define DIRGY     0.
#define DIRGZ     0.
```

The parameter `RT` is set to 0, meaning that physical properties do not depend on temperature. The boundary conditions in `MGSolver3DNS.C` are

```
// boundary conditions box
if (xm < 0.001) { // y-z symmetry plane
bc[Level][dof_u]=0;
}
if (ym < 0.001) { // x-z symmetry plane
bc[Level][dof_v]=0;
}
if (xm > 0.001 && ym > 0.001  &&
     zm < LZ-0.001 && zm > 0.001) { // reactor boundary
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
}
if (zm > LZ-0.001 ) { // top
}
if (zm < 0.001) { // bottom
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
     if (i<n_p_dofs)  bc[Level][idx_dof[k+3*n_nodes]]=0;
}
```

The planes $x = 0$ and $y = 0$ are symmetry planes. On the reactor boundary we set no tangential and not-crossing conditions. On the bottom, we impose inlet pressure `P_IN` equals to 36000 and velocity parallel to z-axis. On the top we set free outflow boundary conditions.

The boundary conditions in the file `MGSolver3DT.C` are as follows

```
// boundary conditions on a box
if (zp < 0.001)   bc[Level][dof_u]=0;
```

The only condition imposed is at the inlet: the temperature is fixed at the `T_IN` equal 673.15 value. No heat flux condition is set over the remaining boundary. As expected the velocity profile is monodimensional. The w-component is constant along the channel and reaches the expected analytic solution value of $1.567 m/sec$. In the initial part of the reactor ($0 < z \leq H_{in}$), where no source generation is present, the temperature is constant ($T = T_{in} = 673.15K$). Then, it assumes a linear profile in consequence of constant power generation ($H_{in} < z < H_{out}$). In the upper part of the reactor ($H_{out} \leq z < H$) the profile is again constant. The value of temperature on the top is $758.9K$.

3.2 Simulations of a test reactor model

3.2.1 Reactor model core

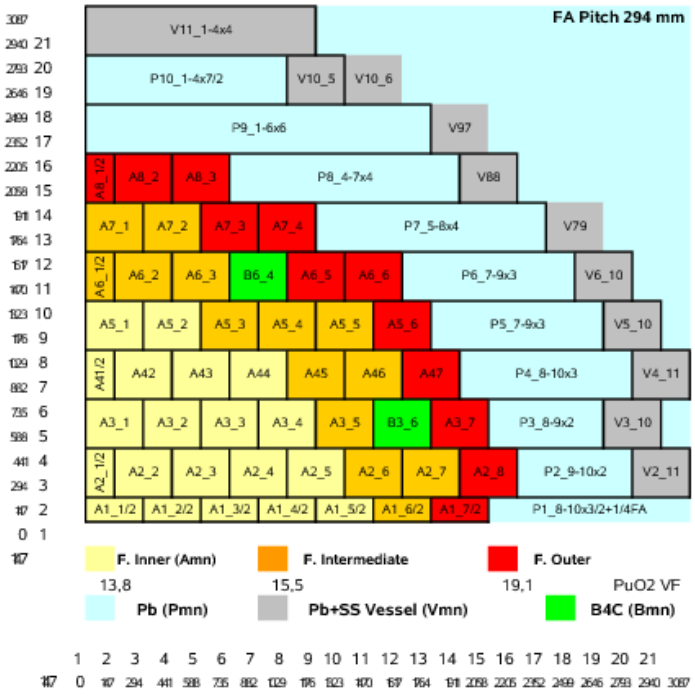


Figure 3.2: Scheme of a quarter of the Reactor model.

For the reactor model we consider the rector in Figure 3.2 where only a quarter of the total geometry is reported. The fuel assembly consists of $n \times n$ pins lattice resulting in 170 positions fitting the required core area in a circular arrangement. The model design distributes the fuel assemblies in three 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

upon the average fuel assembly power, are mapped in Figure 3.3. The maximum peak factor is 1.17, while the minimum is 0.74.

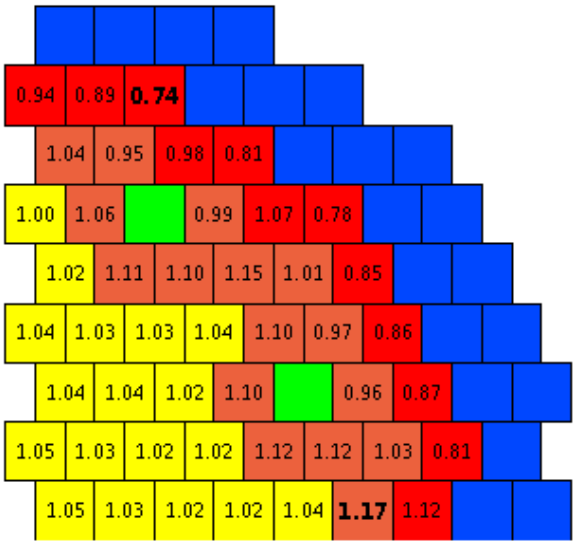


Figure 3.3: Reactor model power distribution.

We label the assemblies as in Figure 3.2; 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 over a staggered grid. The length of a square assembly (LFA) is $0.294m$ in working conditions at the temperature of $673.15K$. The reactor is cooled by lead which enters at the temperature of $673.15K$. We are interested only in core active part and therefore in our computational description we consider a region which is $0.9m$ below the core and $0.2m$ above the core for a total of $2m$. The heat generation zone or the active core zone for the reactor starts at $H_{in} = 0.9m$ and ends at $H_{out} = 1.8m$. The reactor is cooled by lead which enters at the temperature of $673.15K$. Since our model describes the reactor at assembly level the sub-assembly composition is seen as an homogeneous medium. Data about this model composition are assumed as Table 3.3. In particular we

	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	0.548

Table 3.3: Coolant assembly area ratio data

note that the coolant-assembly ratio is 0.548.

3.2.2 Test without control rod assemblies (test2)

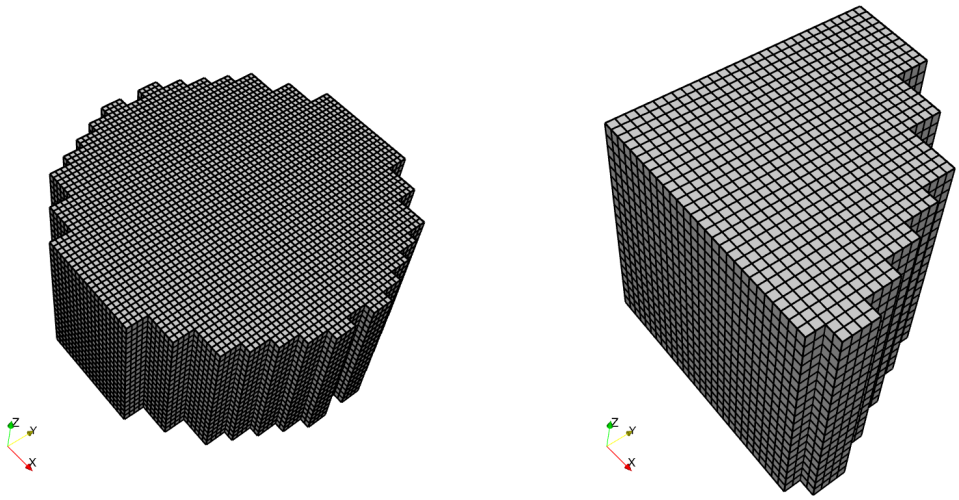


Figure 3.4: Test2. Full reactor and computational domain.

This test is a monodimensional simulation of the model reactor. No turbulent coefficients, no rod control assemblies and no pressure loss and power distribution factors are taken into account. In the above conditions, the solution may be considered monodimensional and therefore the results of the previous section must be recovered. The geometry is shown in Figure 3.4 and packaged in `data_in/RM.tar.gz`. Each assembly is represented by 4×10 Hex27 finite elements. Since the reactor is symmetric along $x = 0$ and $y = 0$ planes we can divide the reactor in 4 parts and compute only one of them, as shown in Figure 3.4. For this quarter of the reactor the nodes are 116481, the dofs 481485 (velocity, pressure and temperature) and the elements are 15561. In order to expand the compressed file one must reach the directory `data_in` and then type the command
`tar xvf RM.tar.gz`.
The main options for the `config/config.h` file are

```
#define BOUNDARY 1
#define NS_EQUATIONS  1
#define T_EQUATIONS 1
#define RESTARTIME 0.
#define PRINT_STEP 2
```

The data are set as following (`config/data.h`)

```
// time
#define ND_TIME_STEP  0.01
#define TIME          0.0
#define N_TIME_STEPS  500

// Reference quantities
```

```
#define Uref      1.
#define Lref      1.
#define Tref      1000.
#define RHOref    100000.

// Inlet
#define P_IN      36000.
#define T_IN      673.15

// fluid properties
#define RHO0      11367.0
#define MU0       .0022
#define NUT       .0
#define KAPPA0    15.8
#define PRT       0.9
#define CP0       147.3

// Heat source
#define QHEAT     1.14591e+8

//  temperature function
#define RT        0.

// Gravity
#define DIRGX     0.
#define DIRGY     0.
#define DIRGZ     0.
```

The parameter `RT` is set to 0, meaning that physical properties do not depend on temperature. It is necessary to keep a small time step `ND_TIME_STEP` in order to assure convergence. The value $\Delta t = 0.01$ may be sufficient. The boundary conditions in the file `MGSolver3DNS.C` are

```
// boundary conditions box
if (xm < 0.001) { // y-z symmetry plane
bc[Level][dof_u]=0;
}
if (ym < 0.001) { // x-z smmetry plane
bc[Level][dof_v]=0;
}
if (xm > 0.001 && ym > 0.001  &&
     zm < LZ-0.001 && zm > 0.001) { // reactor boundary
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
}
if (zm > LZ-0.001 ) { // top
}
if (zm < 0.001) { // bottom
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
```

```
      if (i<n_p_dofs)   bc[Level][idx_dof[k+3*n_nodes]]=0;
}
```

As usual, the planes $x = 0$ and $y = 0$ are symmetry planes. On the reactor boundary we set no tangential and not-crossing conditions. On the bottom, we impose inlet pressure `P_IN` equals to 36000 and velocity parallel to z-axis. On the top we set free outflow boundary conditions.

The boundary conditions in `MGSolver3DT.C` are as follows

```
// boundary conditions on a box
if (zp < 0.001)    bc[Level][dof_u]=0;
```

The only condition imposed is at the inlet: the temperature is fixed at the `T_IN` equal 673.15 value. No heat flux condition is set over the remaining boundary.

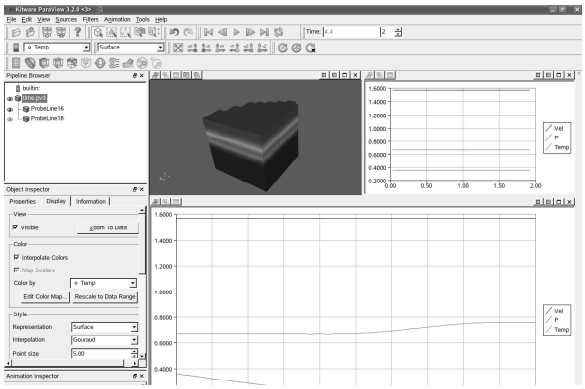


Figure 3.5: Test2. Paraview screen shot.

We assume constant properties as in Table 3.1.2 and introduce sinusoidal power generation along the z-axis in the form

$$\dot{q}''' = W_0 \frac{\pi(H_{out} - H_{in})}{2} \sin\left(\frac{\pi z}{(H_{out} - H_{in})}\right) \tag{3.10}$$

with W_0 constant over all the domain Ω. This implementation guarantees that the total power generated remains the same as the other tests. A paraview snapshot of the solution is in Figure 3.5. The pressure, shown in Figure reffigtest2par, is linear and matches exactly the analytical solution. As expected, the velocity profile, shown in Figure 3.6, is monodimensional and constant with value $1.567 m/sec$. The temperature, shown in Figure 3.6, is constant for the initial part of the reactor where no source generation is present then assumes the standard profile with sinusoidal heat source and finally is constant again on the top of the reactor. The value of temperature is again $758.89 K$.

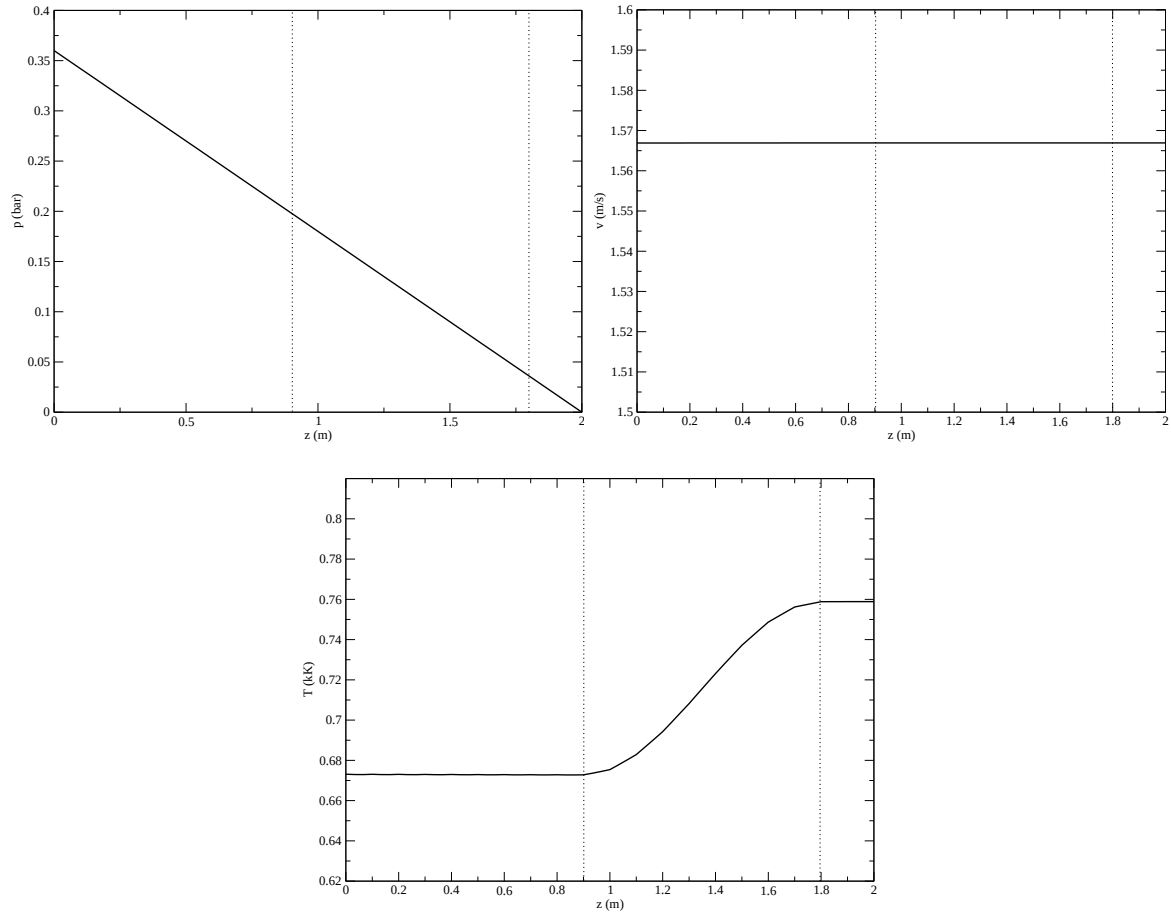


Figure 3.6: Test2. Velocity and pressure distribution (top) and temperature profile along the z-axis (bottom).

3.2.3 Test with control rod assemblies (test3)

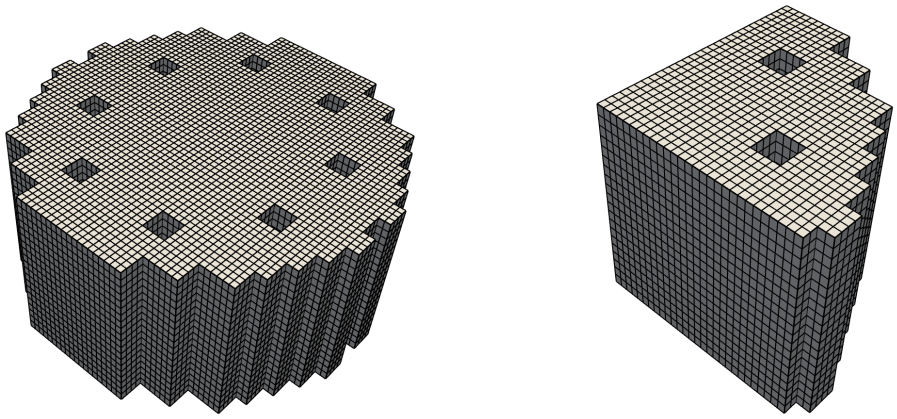


Figure 3.7: Test3: full reactor and computational domain.

This test is a simulation of the reactor model with control rod assemblies and power distribution factors enclosed. No turbulent coefficients and no pressure loss distribution are taken into account. The geometry is shown in Figure 3.7. The core is divides in three zones: 56 fuel assemblies in the inner zone, 62 fuel assemblies in the intermediate zone and the remaining 44 fuel assemblies in the outer zone. The power distribution factors (i.e., the power of a fuel assembly upon the average fuel assembly power) are mapped in Figure 3.3. This map is reported in the file `data_in/data.in`. This file is structured as follows

```
 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
8 0.375 1.375 0.375 1.03 1
9 0.875 1.375 0.375 0.92 1
10 1.375 1.375 0.375 1.04 1
11 1.875 1.375 0.375 1.1 1
12 0.375 1.875 0.375 1.3 1
13 0.875 1.875 0.375 0.89 1
.....................
.....................
```

The first line shows the level and the number of elements. Then each line indicates a fem Hex27 element with the center (x, y, z), the power distribution factor and the pressure loss factor. We advise

to use the appropriate program provided in the directory `contrib/datagen` instead of editing by hand. For details, see Section 2.3.6.

The pressure loss distribution factor is always set to 1. The main options for the `config/config.h` file are

```
//#define DIM2  1
#define BOUNDARY 1
#define NS_EQUATIONS  1
#define T_EQUATIONS 1
#define RESTARTIME 0.
//#define RESTART  750
//#define GENCASE 1
//#define LIBMESHF 1
#define PRINT_STEP 50
```

The main options for the `config/data.h` file are

```
// time
#define ND_TIME_STEP  0.0025
#define TIME          0.0
#define N_TIME_STEPS  2000

// Reference quantities
#define Uref      1.
#define Lref      1.
#define Tref      1000.
#define RHOref    100000.

// Inlet
#define P_IN      36000.
#define T_IN      673.15

// fluid properties
#define RHO0      11367.0
#define MU0       .0022
#define NUT       .0
#define KAPPA0    15.8
#define PRT       0.9
#define CP0       147.3

// Heat source
#define QHEAT     1.2025e+8

//  temperature function
#define RT        1.
```

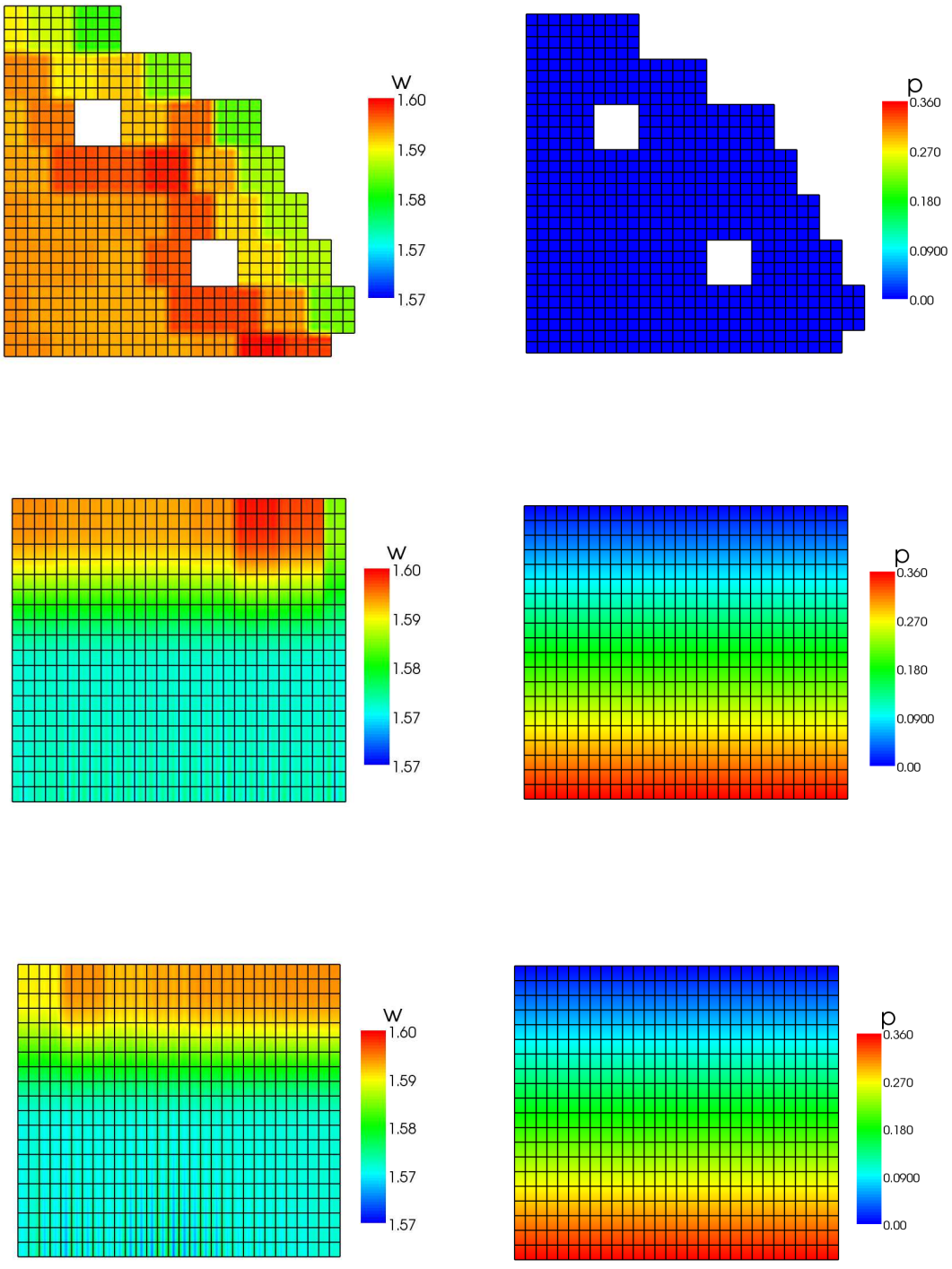


Figure 3.8: Test3. Paraview screen shot of the $z = 2$ (top), $y = 0$ (middle) and $x = 0$ (bottom) planes: velocity (left) and pressure distribution (right).

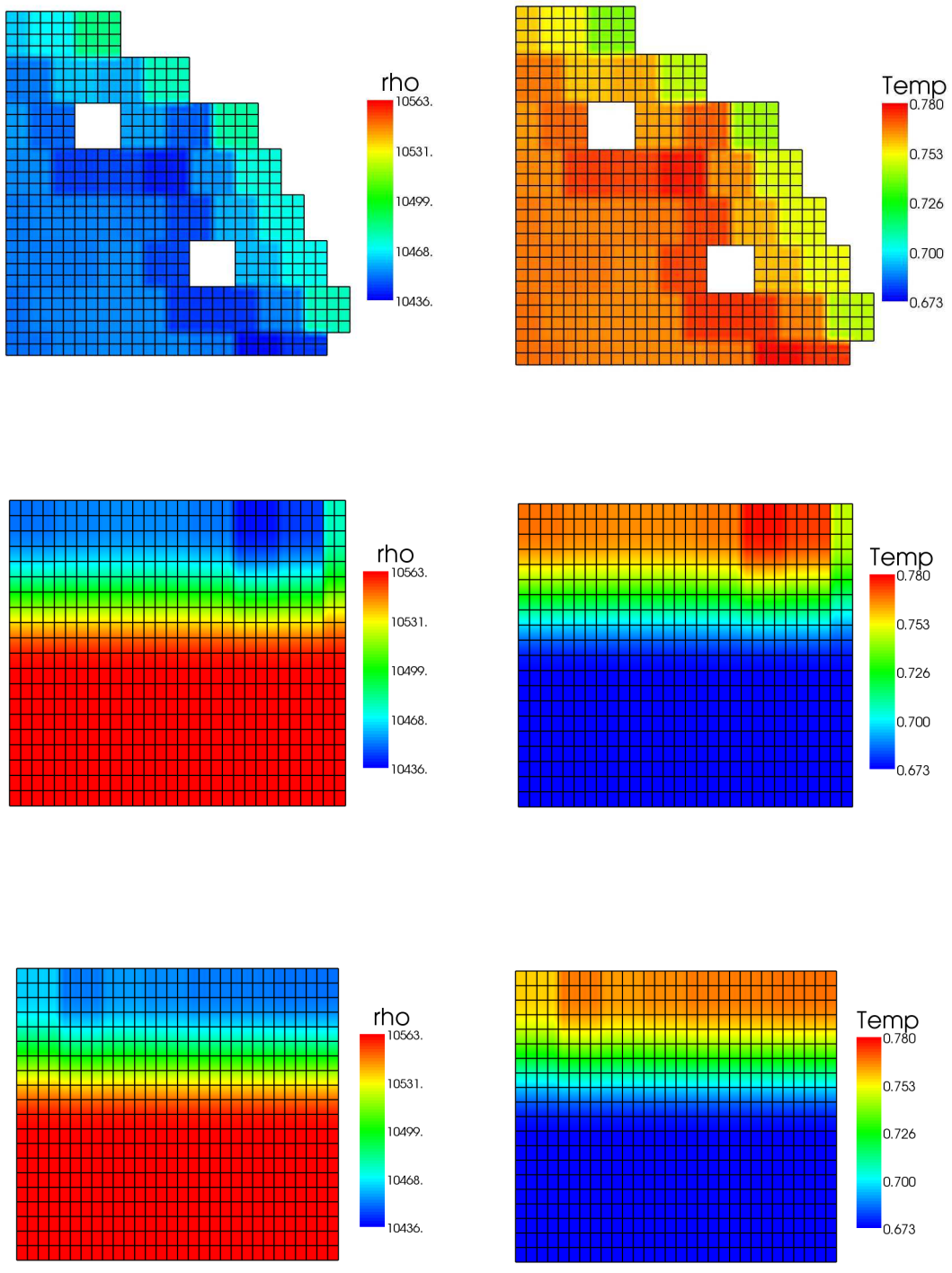


Figure 3.9: Test3. Paraview screen shot of the $z = 2$ (top), $y = 0$ (middle) and $x = 0$ (bottom) planes: density (left) and temperature distribution (right).

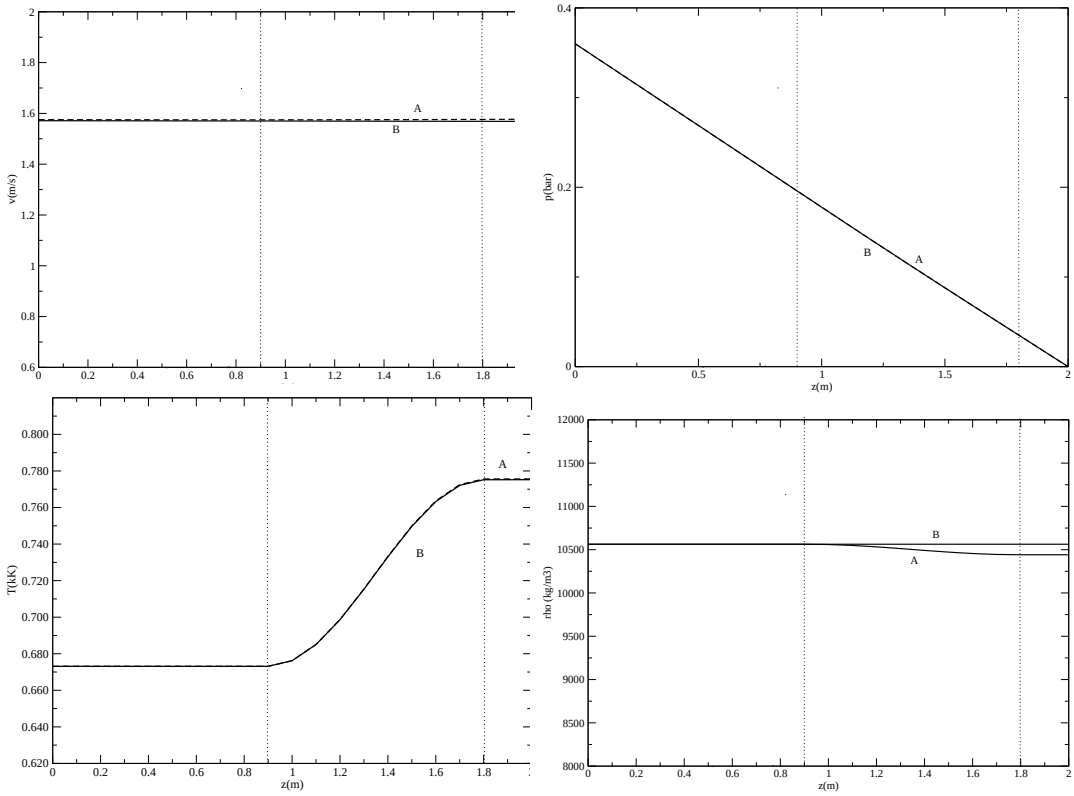


Figure 3.10: Test3 (see section 3.2.3). Paraview screen shot (top), velocity and pressure distribution (top) and temperature and density profile (bottom) along the z-axis at the point $(1.1m, 1.1m)$. Computations with variable ((A) `RT`=1) and constant properties ((B) `RT`=0)

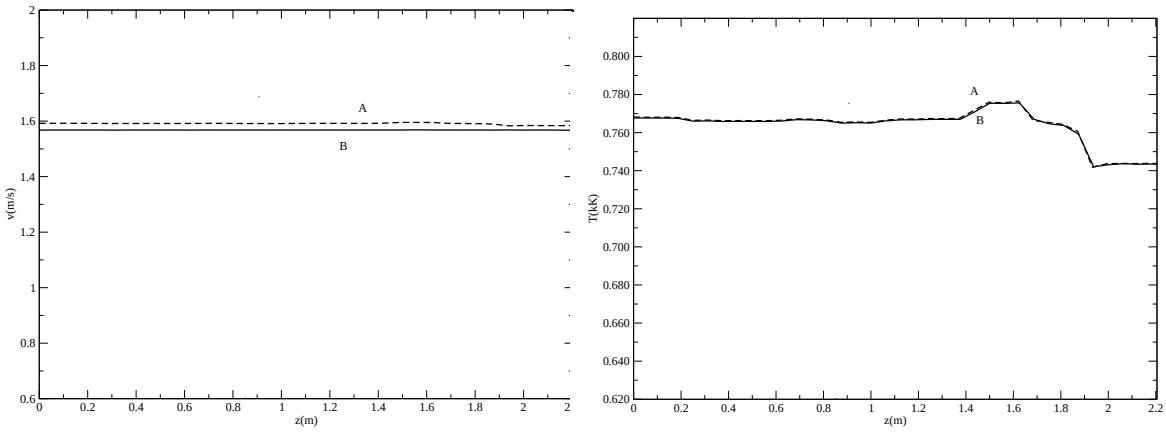


Figure 3.11: Test3. Top view and line $y = x$. Velocity and temperature profile along this line . Computations with variable ((A) `RT`=1) and constant properties ((B) `RT`=0)

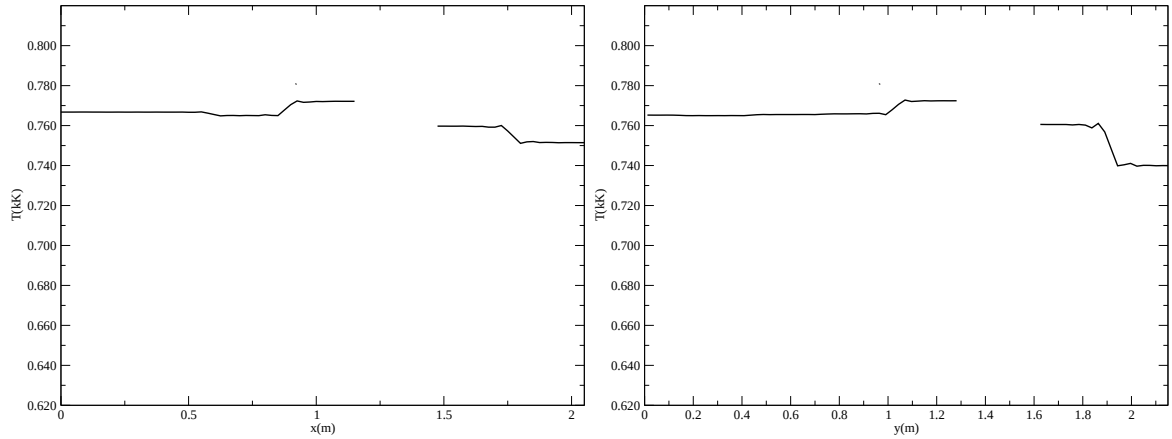


Figure 3.12: Test3. Temperature profile along the line $y = 0.58$, $z = 2$ (left) and the line $x = 0.6$, $z = 2$ (right).

```
// Gravity
#define DIRGX     0.
#define DIRGY     0.
#define DIRGZ     0.
```

	location	Temp	Pw factor
max	A1_6	780	1.17
min	A8_3	737	0.74
	A5_4	776.5	1.15
	A6_1	763.5	1.
	A3_1	766.5	1.04
	A4_1	766.5	1.04
	A7_1	766.5	1.04
	A1_5	767.5	1.04
	A4_4	767	1.04

Table 3.4: Test3. Temprerature at the top of the reactor.

We remark that the power density `QHEAT` is now distributed over 162 assemblies. The parameter `RT` is set to 1, meaning that physical properties depend on temperature. It is necessary to keep a small time step `ND_TIME_STEP` in order to assure convergence. The boundary conditions in the file `MGSolver3DNS.C` are

```
// boundary conditions box
if (xm < 0.001) { // y-z symmetry plane
bc[Level][dof_u]=0;
}
if (ym < 0.001) { // x-z symmetry plane
bc[Level][dof_v]=0;
```

```
}
if (xm > 0.001 && ym > 0.001  &&
     zm < LZ-0.001 && zm > 0.001) { // reactor boundary
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
}
if (zm > LZ-0.001 ) { // top
}
if (zm < 0.001) { // bottom
     bc[Level][dof_u]=0; bc[Level][dof_v]=0;
     if (i<n_p_dofs)  bc[Level][idx_dof[k+3*n_nodes]]=0;
}
```

As usual the planes $x = 0$ and $y = 0$ are symmetry planes. On the reactor boundary we set no tangential and not-crossing conditions. On the bottom, we impose inlet pressure `P_IN` equals to 36000 and velocity parallel to z-axis. On the top we set free outflow boundary conditions.

The boundary conditions in the file `MGSolver3DT.C` are as follows

```
// boundary conditions on a box
if (zp < 0.001)   bc[Level][dof_u]=0;
```

The only condition imposed is at the inlet: the temperature is fixed at the `T_IN` equal 673.15 value. No heat flux condition is set over the remaining boundary. We have performed two computations.
A) Temperature dependent property simulation (`RT`= 1). The results can be seen in the file `test3a.vtu`..
B) Constant property simulation (`RT`= 0). The results can be seen in the file `test3b.vtu`.
Both files can be found in `RM2.tar.gz`.

In Figure 3.8 and Figure 3.9 some paraview snapshots of the top view and $y = 0$ plane are shown. In these figures velocity, pressure, density and temperature distributions are reported. These profiles, along the z-axis at the point $(1.1m, 1.1m)$, can be seen in details in Figure 3.10 A comparison between case (A) and (B) is shown in Figure 3.11, for the line $y = x$ and $z = 2$. We remark that the outlet temperature is not constant but changes following the power factor distribution. Temperature profile along the line $y = 0.58$, $z = 2$ and the line $x = 0.6$, $z = 2$ are shown in Figure 3.12. In Table 3.4 we can find the temperature at the top of the reactor for key assemblies. All the results can be seen by using paraview over the file `test3a.vtu` and `test3b.vtu`.

3.2.4 Test with intra-assembly turbulent viscosity (test4)

This test is a simulation of the reactor model with turbulent viscosity different from zero. We assume a constant turbulent viscosity among assmblies which is imposed from the input file. It is clear that this is no realistic but this may give an idea of the change if a turbulent model is used. This code can be easily adjusted to include a turbulence module coupled with the conservation system but in this version of the code the coupling is not implemented. The main options for the `config/config.h` file are the same as before. The main options for the `config/data.h` file are the same with the exception of the `NUT` parameter which is set to 0.001

In Figure 3.13 the top view (top) $y = 0$ (middle) and $x = 0$ (bottom) plane are shown for density and temperature distributions across the reactor. A comparison between the turbulence case (A) and the

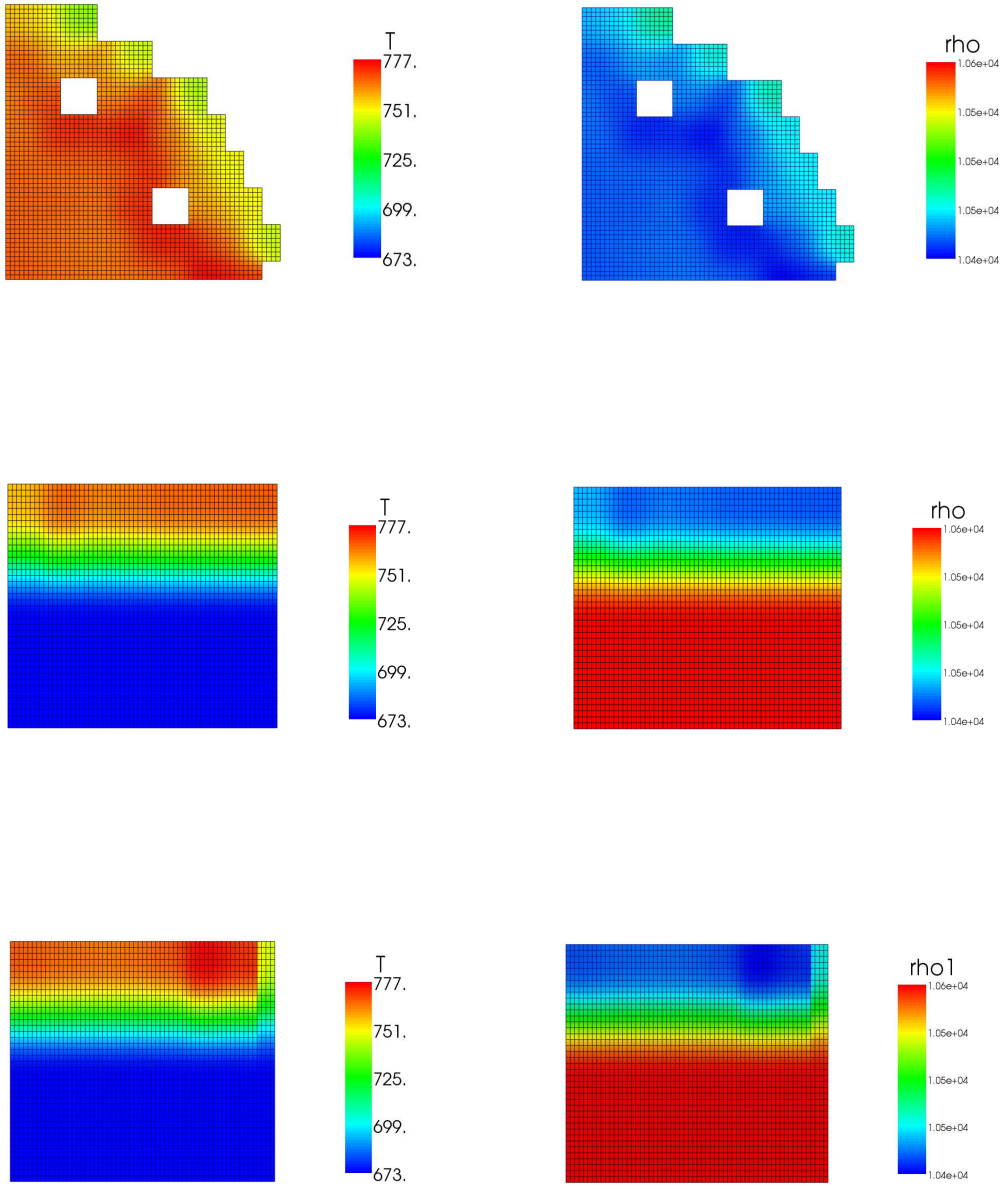


Figure 3.13: Test4. Paraview screen shot of the $z = 2$ (top), $y = 0$ (middle) and $x = 0$ (bottom) planes: temperature (left) and density distribution (right).

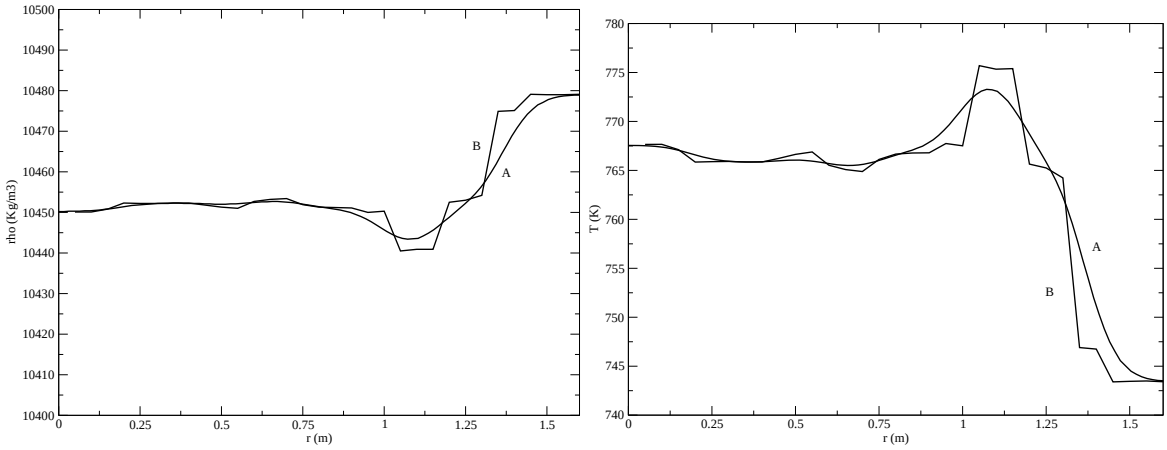


Figure 3.14: Test4. Radial density and temperature profiles along the line $y = x$ at the top of the reactor. Computations with $\nu_t = 0.001$ (A) and $\nu_t = 0$ (B)

non-tubulence case (B), which is defined in the previous section, is shown in Figure 3.14, for the line $y = x$ and $z = 2$. The case A is computed with constant turbulence parameter $\nu_t = 0.001$ and the case B with $\nu_t = 0$. On the right there is the density profile and on the left we can see the distribution of the temperature along the radial line of the reactor. We remark that the outlet temperature and the density are not constant and the profiles follow the power factor distribution. In the case A one can see that the turbulence is smoothing the radial profile since the turbulence increases the heat distribution across the assemblies. Of course the evaluation of the turbulence term may be not correct but can give a feeling of the possible assembly heat exchange. CFD numerical simulations of rod bundle flows or experimental data must be used to define properly the distribution of ν_t across the reactor.

Bibliography

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

[2] C. M. Antonucci et al., EFIT/ELSY, Pressure Drop Performance of the Rod Bundle, Technical report FPN-P9IX-002, ENEA 2002.

[3] GMV visualization software: http://www-xdiv.lanl.gov/XCM/gmv/GMVHome.html

[4] Laspack (linear algebra sparse matrix package): http://www.mgnet.org/mgnet/Codes/laspack/html/laspack.html

[5] Libmesh package: http://libmesh.sourceforge.net/

[6] Paraview visualization software: http://www.paraview.org

[7] VTK library: http://www.vtk.org

Appendix A

Code documentation

A.1 Class index

A.1.1 Reactor model (RM) code File List

Here is a list of all files with brief descriptions:

A.2 Class documentation

A.2.1 MGCase Class Reference

This class controls the input and output data flow. It prints the case file for restarting and visualization.

The documentation for this class is generated from the following files:

- MGCase.h
- MGCase.C

A.2.1.1 Member Data Documentation

- unsigned int _NoLevels

  This is the number of the multigrid levels .
- unsigned int _dim

  This is the space dimension
- const MGMesh * _mgmesh

  This is the mesh pointer
- double ** aux_data

  This is an auxiliary vector for input or output data

Public Member Functions

- MGCase (const MGMesh &mgmesh, const unsigned int NoLevels)

  This is a generic class constructor
- MGCase (const MGMesh &mgmesh, const unsigned int NoLevels, const unsigned int timesteps)

  This is the class constructor for restarting
- ~MGCase ()

  This is the desctructor
- void init_data ()

  This function initializes the data
- double get_data (const int Level, const int iel) const

  This inline function gets data from the data.in file
- void print_xml (const unsigned int timesteps)

  This function prints the time.pvd file
- void restart_xml (std::ifstream &infile, std::ofstream &out, const int icase, const unsigned int t_rest, const unsigned int time_step)

  This function restarts the simulation from the time.pvd file
- void print (MGMesh &mgmesh, const unsigned int flag_print, const unsigned int Level)

  This function prints the mesh
- void print (MGMesh &mgmesh, MGSol &mgsol, MGSolCC &mgcc, MGSolT &mgsolT, MGSolSA &mgsolSA, const unsigned int flag_print, const unsigned int Level)

  This function prints solution in vtk format
- void print_bc (MGMesh &mgmesh, MGSol &mgsol, MGSolCC &mgsolCC, MGSolT &mgsolT, MGSolSA &mgsolSA, const unsigned int flag_print, const unsigned int Level)

  This function prints the boundary and boundary conditions

- void read (MGMesh &mgmesh, MGSol &mgsol, MGSolCC &mgsolCC, MGSolT &mgsolT, MGSolSA &mgsolSA, const unsigned int flag_print, const unsigned int Level)

  This function reads the solution from a vtu file for restarting

A.2.2 MGGauss Class Reference

This class contains shape function values and Gaussian point weights for standard fem integration.

The documentation for this class was generated from the following files:

- MGGauss.h
- MGGauss.C

Public Attributes

- unsigned int _dim

  This is the dimension (2 or 3)
- unsigned int _NoGauss [3]

  This is the number of Gaussian points in 1 2 3D.
- unsigned int _NoShape [3]

  This is the number of shape functions in 1 2 3D.
- double * _weight1D

  These are gaussian weights in 1D
- double * _phi1D_map

  These are the shape functions in 1D
- double * _dphidxez1D_map

  These are the shape derivative functions in 1D
- double * _weight2D

  These are the gaussian weights in 2D
- double * _phi2D_map

  These are the shape functions in 2D
- double * _dphidxez2D_map

  These are the shape derivative functions in 2D
- double * _weight3D

  These are gaussian weights in 3D
- double * _phi3D_map

  These are the shape functions in 3D
- double * _dphidxez3D_map

  These are the shape derivative functions in 3D

Public Member Functions

- MGGauss (const unsigned int _NoGauss[ ], const unsigned int _NoShape[ ])
  This is the constructor.
- void init (const int mode)
  This class allocates the memory data.
- double Jac2D (const unsigned int ng, double x[ ], double y[ ], double InvJacc[ ]) const
  This function computes the Jacobian in 2D space.
- double JacSur2D (const unsigned int ng, double x[ ], double y[ ]) const
  This computes line Jacobian in 2D space.
- double Jac3D (const unsigned int ng, double x[ ], double y[ ], double z[ ], double InvJacc[ ]) const
  This computes the Jacobian in 3D space.
- double JacSur3D (const unsigned int n_gauss, double x[ ], double y[ ], double z[ ]) const
  This computes the surface Jacobian in 3D space.
- ∼MGGauss ()
  This is the destructor.
- void clear ()
  This function is empty.

- void write (const std::string &name)
  This function writes the gaussian points over a file

Private Member Functions

- void read_c3D (std::istream &infile)
  This function reads 3D gaussian points
- void read_c2D (std::istream &infile)
  This function reads 2D gaussian points
- void read_c1D (std::istream &infile)
  This function reads 1D gaussian points
- void write_c3D (std::ostream &infile)
  This function writes the 3D gaussian points

A.2.3 MGMesh Class Reference

This class defines the Multigrid mesh. The number of levels is `NoLevels`. There is a unique structure that stores all vertices for all levels. Each level has its own connection map and element map. There is a further connection map that defines the boundary. The offset vector determines the type of element on the base of the number of nodes.

The documentation for this class was generated from the following files:

- MGMesh.h
- MGMesh.C

Public Attributes

- unsigned int _dim

  This is the dimension (2 or 3)
- unsigned int _NoLevels

  This is the number of multigrid levels.
- int * _NoNodes

  This is the number of mesh nodes (Level).
- int * _NoElements

  This is the number of mesh elements (Level).
- int * _NoElementsB

  This is the number of boundary elements (Level).
- int * _NoOffset

  This is the Offset for boundary elements (Level).
- double * _xyz

  This is the vertex coordinate vector
- int ** _offset

  This is the offset map (type of element)
- int ** _elem_map

  This is the node map (connectivity)

Public Member Functions

- MGMesh (const unsigned int dim_in, const unsigned int NoLevels_in)

  This function is the mesh constructor.
- void init (const std::string &name)

  This function initializes the data mesh allocation
- ~MGMesh ()

  This function is the level Mesh Destructor.
- void clear ()

  This function is the substructure destructor
- void write (const std::string &name, int level)

  This function writes into file
- void print (const unsigned int flag_print, const unsigned int Level)

  This function prints in vtk or gmv format.
- void printb (const unsigned int flag_print, const unsigned int Level)

  This function prints the boundary in vtk format
- void print_nodes (std::ofstream &name, const unsigned int Level, const unsigned int mode)

This function prints node coordinates (_xyz data)

- void print_conn (std::ofstream &name, const unsigned int Level, const unsigned int mode)

  This function prints the node connectivity (_elem_map data)

Private Member Functions

- void read_c (std::ifstream &infile)

  This function reads from standard MGMesh file
- void write_c (std::ostream &infile, int level)

  This function writes to standard MGMesh file

A.2.4 MGSol Class Reference

This class solves the Navier-Stokes equation with different multigrid algorithms. The momentum equation and the conservation of mass equations are discretized by using finite element method. The discrete equations are in the function `GenMatRhs`. The boundary conditions are in the `GenBc` function and the inital solution in the `GenSol` function.

The documentation for this class was generated from the following files:

- MGSolver.h
- MGSolver.C
- MGSolver3DNS.C

Private Attributes

- const MGMesh * _mgmesh

  This is the internal mesh pointer needed from the class constructor

Public Attributes

- unsigned int _NoLevels

  This is the level number
- int * Dim

  This is the dimension
- QMatrix * L

  The matrix L(uw).
- QVector * uw

  The solution vector uw
- QVector * fr

The rhs vector fr

- QVector * uw_old

  This is a solution at the previous step n-1
- int ** bc

  This vector constains the boundary conditions
- int ** _node_dof

  This is the vector for the matrix dof
- int ** _node_matrix_stor

  This the aux matrix storage vector
- Matrix * R

  This is the Restrictor Operator.
- Matrix * P

  This is the Prolungation operator.

Public Member Functions

- MGSol (const MGMesh &mgmesh, const unsigned int NoLevels)

  This function is the constructor
- void init_dof (const unsigned int Level)

  This function initializes dof distribution vector
- void init (const unsigned int Level)

  This function initializes the memory allocation.
- void init_R (const unsigned int Level)

  This function initializes the memory allocation for multigrid Restrictor Operator.
- void init_P (const unsigned int Level)

  This function initializes memory allocation for the multigrid Prolongation Operator.
- ~MGSol ()

  This function is the destructor (level structure).
- void clear ()

  This function is the substructure destructor.
- void GenSol (const unsigned int Level)

  This function computes the initial solution uw (t=0).
- void WriteSol (const unsigned int Level, const std::string &name, const unsigned int nvars)

  This function writes the solution
- void ReadSol (const unsigned int Level, const std::string &name)

  This function reads the solution
- void Sol_addoldsol (double a, const unsigned int Level)

  *This function adds the old solution to the new one (a*uw_old+uw).*
- void GenOldSol (const unsigned int Level)

  This function initializes the memory allocation for uw_old.
- void OldSol_update (const unsigned int Level)

This function copies the solution uw to the old solution uw_old.

- void OldSol_addsol (double a, const unsigned int Level)

  This function adds the old solution to the new one (a∗uw_old+uw).
- void GenMatrix (const MGSolT &mgT, const MGSolCC &mgcc, const MGGauss &mggauss, const MGGauss &mggss1, const unsigned int Level)

  This function assembles the matrix L.
- void GenMatRhs (const double time, const MGCase &mgcase, const MGSolT &mgT, MGSolCC &mgcc, const MGGauss &mggauss, const MGGauss &mggss1, const unsigned int Level, const int mode)

  This function assembles the matrix L and the rhs vector fr.
- void InitGenMatrix (int unsigned mem, const std::string &name, const unsigned int Level)

  This function allocates the entries of the matrix L.
- void ReadMatrix (const unsigned int Level, const std::string &name)

  This function reads the matrix L.
- void WriteMatrix (const unsigned int Level, const std::string &name, const unsigned int vars)

  This funtions writes the matrix L.
- void GenRhs (const double time, const MGSolT &mgT, MGSolCC &mgcc, const MGGauss &mgsg2, const MGGauss &mgs1, const unsigned int Level)

  This function assembles the rhs vector (fr).
- void WriteRhs (const unsigned int Level, const std::string &name, const unsigned int nvars)

  This function writes the rhs vector (fr).
- void ReadRhs (const unsigned int Level, const std::string &name)

  This function reads the rhs vector (fr).
- void MGReadOp ()

  This function reads multigrid operators.
- void InitProl (const unsigned int level)

  This function computes multigrid Prolongation operator.
- void ReadProl (const unsigned int Level, const std::string &name)

  This function reads multigrid Prolongation operator.
- void WriteProl (const unsigned int Level, const std::string &name, const unsigned int vars)

  This function writes multigrid Prolongation operator.
- void InitRestweight (const int Level)

  This function computes the multigrid Prolongation operator.
- void ReadRest (const unsigned int Level, const std::string &name)

  This function reads multigrid Prolongation operator.
- void WriteRest (const unsigned int Level, const std::string &name, const unsigned int vars)

  This function writes multigrid Prolongation operator.
- void GenRes (const unsigned int Level)

  This function generates the residual.
- void GenBc (const unsigned int Level)

  This function sets the boundary conditions

- void MGAssemble (MGSolCC &mgcc, MGGauss &dgauss, MGGauss &dgauss_p, unsigned int nlevels, unsigned int mode)

  This function assembles the matrix.
- void MGTimeStep (const MGCase &mgcase, const MGSolT &mgT, MGSolCC &mgcc, MGGauss &dgauss, MGGauss &dgauss_p, const double time, const int iter)

  This function is the multigrid time step solver (backward Euler).
- void MGSolve (int NoLevels, IterIdType MLSolverId, bool NestedMG, int NoMGIter, int Gamma, IterProcType SmoothProc, int Nu1, int Nu2, PrecondProcType PrecondProc, double Omega, IterProcType SolvProc, int NuC, PrecondProcType PrecondProcC, double OmegaC, int MaxIter, double Eps)

  This function is the multigrid solver.
- void GenRhsB (const double time, MGSolCC &mgcc, MGGauss &mggauss, MGGauss &mggss1, const int Level)

  This function assembles the surface rhs term (top level).
- void print_p (std::ofstream &out, const unsigned int offset, const unsigned int Level)

  This function prints the pressure field
- void print_p_bc (std::ofstream &out, const unsigned int offset, const unsigned int Level)

  This function prints the pressure field on the boundary.
- void print_u (std::ofstream &out, const unsigned int offset, const unsigned int Level, const unsigned int component)

  This function prints one component of the velocity field
- void print_bc (std::ofstream &out, const unsigned int offset, const unsigned int Level, const unsigned int component)

  This function prints one component of the velocity field on the boundary
- void print_uvw (std::ofstream &out, const unsigned int offset, const unsigned int Level)

  This function prints velocity field

A.2.5 MGSolT Class Reference

This class solves the energy equation with different multigrid algorithms. The energy equation is discretized by using finite element method. The discrete equation is defined in the function `GenMatRhs`. The boundary conditions are in the `GenBc` function and the inital solution in the `GenSol` function.

The documentation for this class was generated from the following files:

- MGSolverT.h
- MGSolver3DT.C
- MGSolverT.C

Public Attributes

- unsigned int _NoLevels
  This is the numbers of levels
- size_t * Dim
  This is the dimension
- QMatrix * L
  This is the matrix L(uw).
- QVector * uw
  This is the solution uw
- QVector * fr
  This is the rhs vector (fr)
- QVector * uw_old
  This is the solution at the previous step (n-1)
- int ** bc
  This vector contains the boundary conditions
- int ** _node_dof
  This vector contains the matrix dof
- int ** _node_matrix_stor
  This vector is used to store the auxiliary matrix data
- Matrix * R
  This is the restrictor operator.
- Matrix * P
  This is the Prolungation operator.

Private Attributes

- const MGMesh * _mgmesh
  This is the mesh pointer

Public Member Functions

- MGSolT (const MGMesh &mgmesh, const unsigned int NoLevels)
  This function is the constructor.
- void init_dof (const unsigned int Level)
  This function distributes dof
- void init (const unsigned int Level)
  This function initializes the memory allocation.
- void init_R (const unsigned int Level)
  This function allocates the multigrid Restrictor Operator.
- void init_P (const unsigned int Level)

This function allocates the multigrid Interpolation Operator.

- ~MGSolT ()

  This is the destructor (level structure).
- void clear ()

  This is the substructure destructor.
- void GenSol (const unsigned int Level)

  This function sets the initial solution uw (t=0).
- void WriteSol (const unsigned int Level, const std::string &name, const unsigned int nvars)

  This function writes the solution.
- void ReadSol (const unsigned int Level, const std::string &name)

  This function reads the solution vector.
- void Sol_addoldsol (double a, const unsigned int Level)

  This function adds the old solution to the new one (a∗uw_old+uw).
- void GenOldSol (const unsigned int Level)

  This function allocates memory for the vector uw_old.
- void OldSol_update (const unsigned int Level)

  This function copies the solution vector uw to the old solution uw_old.
- void OldSol_addsol (double a, const unsigned int Level)

  This function adds the old solution to the new solution (a∗uw_old+uw).
- void GenMatrix (const MGSolCC &mgcc, const MGSol &mgs, const MGGauss &mggauss, const unsigned int Level)

  This function assembles the matrix L.
- void GenMatRhs (const double time, const MGCase &mgcase, const MGSol &mgsol, const MGSolCC &mgcc, const MGGauss &mggauss, const unsigned int Level, const int mode)

  This function assembles the matrix and the rhs.
- void InitGenMatrix (int unsigned mem, const std::string &name, const unsigned int Level)

  This function allocates the entries in the matrix L.
- void ReadMatrix (const unsigned int Level, const std::string &name)

  This function reads the matrix L.
- void WriteMatrix (const unsigned int Level, const std::string &name, const unsigned int vars)

  This function writes matrix L.
- void GenRhs (const double time, const MGGauss &mgsg2, const unsigned int Level)

  This function assembles the rhs vector (fr).
- void WriteRhs (const unsigned int Level, const std::string &name, const unsigned int nvars)

  This function writes the rhs vector (fr).
- void ReadRhs (const unsigned int Level, const std::string &name)

  This function reads the rhs vector (fr).
- void MGReadOp ()

  This function reads multigrid operators.
- void InitProl (const unsigned int level)

  This function computes the multigrid Prolongation operator.

- void ReadProl (const unsigned int Level, const std::string &name)

  This function reads the multigrid Prolongation operator.
- void WriteProl (const unsigned int Level, const std::string &name, const unsigned int vars)

  This function writes the multigrid Prolongation operator.
- void InitRestweight (const int Level)

  This function computes the multigrid Prolongation operator.
- void ReadRest (const unsigned int Level, const std::string &name)

  This function reads the multigrid Prolongation operator.
- void WriteRest (const unsigned int Level, const std::string &name, const unsigned int vars)

  This function writes the multigrid Prolongation operator.
- void GenRes (const unsigned int Level)

  This function computes the residual
- void GenBc (const unsigned int Level)

  This function computes boundary conditions
- void MGAssemble (const double time, MGGauss &dgauss, const unsigned int nlevels, const unsigned int mode)

  This function assemble the matrix.
- void MGTimeStep (const double time, const MGCase &mgcase, const MGSolCC &mgcc, const MGSol &mgsol, const MGGauss &dgauss, const unsigned int mode)

  This function computes a multigrid time step (backward Euler).
- void MGSolve (int NoLevels, IterIdType MLSolverId, bool NestedMG, int NoMGIter, int Gamma, IterProcType SmoothProc, int Nu1, int Nu2, PrecondProcType PrecondProc, double Omega, IterProcType SolvProc, int NuC, PrecondProcType PrecondProcC, double OmegaC, int MaxIter, double Eps)

  This function is the multigrid solver.
- void print_u (std::ofstream &out, const unsigned int offset, const unsigned int Level)

  This function prints the solution.
- void print_bc (std::ofstream &out, const unsigned int offset, const unsigned int Level)

  This function prints the solution on the boundary.

A.3 Reactor Model code File documentation

A.3.1 The main file `ex13.C`

This file is the main program. It allocates all the needed classes and manages input and output flow. Each time step is solved inside the temporal loop. All the settings must be specified in the `config.h` file.

```
// C++ include files that we need
#include <iostream>
#include <algorithm>
#include <math.h>
```

```
#include <fstream>
#include <map>

#include "config/config.h"
// ------------------------------
#ifdef LIBMESHF
#include "libmesh.h"
#include "perf_log.h"
#define Perf_logstart_event perf_log.start_event
#define Perf_logstop_event perf_log.stop_event
#else
#define Perf_logstart_event (printf)
#define Perf_logstop_event (printf)
#endif
// ------------------------------

#include <lastypes.h>
#include "MGSolver.h"
#include "MGSolverT.h"
#include "MGSolverSA.h"
#include "MGSolverCC.h"
#include "MGMesh.h"
#include "MGGauss.h"
#include "MGCase.h"

#include "config/data.h"

// ************************************************************
// The main program.
// ************************************************************
int main(int argc, char** argv) {

#ifdef LIBMESHF
  // Initialize libMesh.
  libMesh::init(argc, argv);

  {
    // ------------------------------------------------
    PerfLog perf_log("NSE");
#endif
#ifdef GENCASE
    // ************************************
    // Case Generation
    // ************************************
#ifndef RESTART
    Perf_logstart_event("GenCase");
    GenCase();
```

```
    Perf_logstop_event("GenCase");
#endif
#endif
    // ************************************
    // MGSolver begin
    fprintf(stderr,"\n Start MGSolver \n");
    Perf_logstart_event("Reading");
    // Reading ******************************
    // Data ---------------------------------
    printf(" Reading I Data:: \n");
    // double dt    = ND_TIME_STEP;
    unsigned int t_in=0;
    double time=0.;
    printf("\n Nondimensional Numbers: \n");
    printf(" Re = %e We = %e   Fr = %e \n",
           RHOref*Uref*Lref/MU0,
           RHOref*Uref*Uref*Lref/(SIGMA+1.e-6),
           Uref*Uref/(9.81*Lref));
    printf(" Pe = %e  \n",(RHOref*CP0*Lref*Uref)/KAPPA0);
    fprintf(stderr," - \n *\n");

    // velocity fem -------------------
    unsigned int kng[3],kfem[3];
    // N of gaussian points in 1D,2D,3D
    kng[0]=3;kng[1]=9;kng[2]=27;
    // Fem element type in 1D,2D,3D
    kfem[0]=3;kfem[1]=9;kfem[2]=27;
    // pressure fem --------------------
    unsigned int kng1[3],kfem1[3];
    // N of gaussian points in 1D,2D,3D
    kng1[0]=3;kng1[1]=9;kng1[2]=27;
    // Fem element type in 1D,2D,3D
    kfem1[0]=2;kfem1[1]=4;kfem1[2]=8;

    // Gauss Reading  -------------------------
    fprintf(stderr," Reading II Gauss:: \n");
    MGGauss *dgauss; dgauss=new MGGauss(kng,kfem);
    dgauss->init(27);// hex27
    MGGauss *dgauss_p; dgauss_p=new MGGauss(kng1,kfem1);
    dgauss_p->init(8);// hex8

    printf(" - \n * \n");
    //Mesh --------------------
    fprintf(stderr,"\n Reading III Geometry:: \n");
    fprintf(stderr," Mesh:");
    MGMesh *mgmesh1; mgmesh1=new MGMesh(DIMENSION,gridn+1);
    mgmesh1->init(FILE_MESH_CASE);
```

```
    printf(" - \n");

    // Multigrid ------

    MGSol *mgs=NULL;
#ifdef NS_EQUATIONS

    printf("\n Reading IV  Navier-Stokes Operators:: \n");
    printf(" Multigrid, Matrix, Prol and Restr: \n");
    mgs=new MGSol(*mgmesh1,NoLevels);
    mgs->MGReadOp();
    printf(" - \n * \n");
#endif

    MGSolSA *mgsSA=NULL;
#ifdef TURBULENCE
    printf("\n Reading   Turbulence Operators:: \n");
    // Multigrid ------
    printf(" Multigrid, Matrix, Prol and Restr: \n");
    mgsSA=new MGSolSA(NoLevels);
    mgsSA->MGReadOp(*mgmesh1,0);
    mgsSA->InitVel(*mgmesh1,*mgs,dt);
    printf(" - \n *\n");
#endif
    MGSolT *mgsT=NULL;
#ifdef T_EQUATIONS
    printf("\n Reading V  Energy Operators:: \n");
    // Multigrid ------
    printf(" Multigrid, Matrix, Prol and Restr: \n");
    mgsT=new MGSolT(*mgmesh1,NoLevels);
    mgsT->MGReadOp();
    printf(" - \n * \n");
#endif

    // ************************************
    // Set up time cycle t=0 ***************

    MGSolCC *mgcc=NULL;

#ifdef NS_EQUATIONS
    fprintf(stderr,"\n NS Initializing:: \n");
    fprintf(stderr," Solution: ");
    mgs->GenSol(NoLevels-1);
    mgs->GenOldSol(NoLevels-1);
    fprintf(stderr," -\n");
    fprintf(stderr," Rhs: ");
    fprintf(stderr," -\n");
```

```
#endif

#ifdef TURBULENCE
    fprintf(stderr,"\n Turbulence Initilize:: ");
    fprintf(stderr,"\n Solution: ");
    mgsSA->GenSol(*mgmesh1,NoLevels-1);
    mgsSA->GenOldSol(NoLevels-1);
    fprintf(stderr," - \n");
    fprintf(stderr," rhs: ");
    mgsSA->GenRhs(*dgauss,*mgmesh1,NoLevels-1);
    fprintf(stderr," - \n");
#endif
#ifdef T_EQUATIONS
    printf("\n Energy Initilize:: ");
    printf("\n Solution: ");
    mgsT->GenSol(NoLevels-1);
    mgsT->GenOldSol(NoLevels-1);
    printf(" - \n");
#endif

    printf("\n Case Initilize:: ");
    MGCase *mcase=new MGCase(*mgmesh1,NoLevels,N_TIME_STEPS);
    mcase->init_data();
#ifdef RESTART
    printf("\n Restart %d %g",RESTART,RESTARTIME);
    mcase->read(*mgmesh1,*mgs,*mgcc,*mgsT,*mgsSA,RESTART,NoLevels-1);
    t_in=RESTART;
    time +=RESTARTIME;
#endif
    mcase->print(*mgmesh1,*mgs,*mgcc,*mgsT,*mgsSA,t_in,NoLevels-1);
    mcase->print_bc(*mgmesh1,*mgs,*mgcc,*mgsT,*mgsSA,t_in,NoLevels-1);
    printf(" - \n");

    Perf_logstop_event("Reading");

    // ------------------------------------------------------
    //  time loop
    // ------------------------------------------------------
    for (unsigned int t_step=t_in; t_step< N_TIME_STEPS+t_in; ++t_step) {

      // Let the system of equations know the current time.
      std::cout << "\n\n*** Solving time step " << t_step+1
      << ", time = " << time+ ND_TIME_STEP << " ***" << std::endl;
      // ------------------------------------------------------
      Perf_logstart_event("MG Solver");
#ifdef NS_EQUATIONS
      // solving  V-cyle Multigrid
```

```
      printf("\n Navier-Stokes solution:: \n");
      mgs->MGTimeStep(*mcase,*mgsT,*mgcc,*dgauss,*dgauss_p,time,t_step-t_in);
#endif
#ifdef TURBULENCE
      std::cout  << std::endl << "\n Turbulence solution \n" << std::endl;
      mgsSA->MGTimeStep(*mgmesh1,*dgauss,0);
#endif
#ifdef T_EQUATIONS
      std::cout  << std::endl << "\n Energy Solution \n" << std::endl;
      mgsT->MGTimeStep(time,*mcase,*mgcc,*mgs,*dgauss,0);
#endif
      Perf_logstop_event("MG Solver");
      // ------------------------------------------------------
      // print
      Perf_logstart_event("output");
      if ((t_step+1-t_in)%PRINT_STEP == 0)
        mcase->print(*mgmesh1,*mgs,*mgcc,*mgsT,*mgsSA,t_step+1,NoLevels-1);

      Perf_logstop_event("output");
      time += ND_TIME_STEP;
    }   // end time loop

    // clean ------------------------------
    if (mcase != NULL) delete mcase;
    if (mgcc != NULL) delete mgcc;
    if (mgsT != NULL) delete mgsT;
    if (mgs != NULL) delete mgs;
    if (mgsSA != NULL) delete mgsSA;
    if (mgmesh1 != NULL)   delete mgmesh1;
    if (dgauss != NULL) delete  dgauss;
    if (dgauss_p != NULL) delete dgauss_p;

#ifdef LIBMESHF
  }
  return libMesh::close();
#endif
}
```

A.3.2 Configuration file `config.h`

The configuration file `config.h` is reported below. See Section 2.3.1 for option description.

```
#ifndef _config_cfg
#define _config_cfg

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

```
// BASIC SETTINGS (DIM and Navier-Stokes+Energy
// ********************************************

// space 2D or 3D ***************************
// DIM2= two-dimensional simulation (undef 3D)
//#define DIM2  1
// generate the boundary mesh
#define BOUNDARY 1
// 2d axisymmetric case
//#define AXISYMX 1

// equation *****************************
// for NS simulations
#define NS_EQUATIONS  1
//#define TURBULENCE    1
// ------------------------
// for Energy simulations
#define T_EQUATIONS 1
// --------------------------
 //equation TWO_PHASE
//#define TWO_PHASE  1
#define REFLEV 3
//#define INLEV 0
#ifndef INLEV
#define INLEV (REFLEV-1)
#endif
// ************************************


// restart and generation ****************************
#define RESTARTIME 4.725
#define RESTART  3600

// gencase  ******************************
//#define GENCASE 1
#define LIBMESHF 1


//  print ****************************
//#define OUTGMV 1
#define PRINT_STEP 50
#define PRINT_INFO 1


// dimension for internal cube  generation
#define LX 2.
#define LY 2.
```

```
#define LZ 2.


// ************************************************
// 2D  ( DIM2= defined = two-dimensional simulations )
// ************************************************
#ifdef DIM2 // 2D dim=2 --------------------------------
#define DIMENSION    2
#define NX 32
#define NY 32
#define NZ 0

// Integration ***********************************
#define GEN_GAUSS 1
#define N_GAUSS 9
#define FILE_GAUSS  "fem/shape2D_0909.in"
  #define FILE_GAUSS_P  "fem/shape2D_0904.in"

// FEM Element ******************************
#define ELEM_FEM QUAD9
#define NDOF_FEM  9
#define ORDER_FEM SECOND
#ifdef NS_EQUATIONS
 #define ELEM_P QUAD4
 #define NDOF_P  4
 #define ORDER_P FIRST
#else
 #define NDOF_P  4
#endif
#define NDOF_FEM2D  3
#define NDOF_P2D  2

// ****************************************************
// 3D  ( DIM2= undefined = three-dimensional simulations )
// ****************************************************
#else  // 3D dim=3 -------------------------------
#define DIMENSION    3
#define NX 32
#define NY 32
#define NZ 32

// Integration ***********************************
//#define GEN_GAUSS 1
#define N_GAUSS 27
#define N_GAUSS2D 9
#define FILE_GAUSS  "fem/shape3D_2727.in"
#define FILE_GAUSS_P  "fem/shape3D_2708.in"
```

```
  #define FILE_GAUSS_P2D  "fem/shape2D_0904.in"
  #define FILE_GAUSS2D  "fem/shape2D_0909.in"
// FEM Element ******************************
#define ELEM_FEM HEX27
#define NDOF_FEM  27
#define NDOF_FEM2D  9
#define ORDER_FEM SECOND
#ifdef NS_EQUATIONS
 #define ELEM_P HEX8
 #define NDOF_P  8
#define NDOF_P2D  4
 #define ORDER_P FIRST
#else
 #define NDOF_P  8
#endif


#endif
// *****************************************
// FILES
// *****************************************
#define FILE_MESH_READ "data_in/mesh.msh"
#define FILE_MESH_CASE "data_in/mesh.in"

#ifdef NS_EQUATIONS
 #define FILE_DOFS_CASE "data_in/dofs.in"
 #define FILE_OUT_U00 "output/velocity.gmv.00%d"
 #define FILE_OUT_U0  "output/velocity.gmv.0%d"
 #define FILE_OUT_U   "output/velocity.gmv.%d"
 #define FILE_OUT_P00 "output/pressure.gmv.00%d"
 #define FILE_OUT_P0  "output/pressure.gmv.0%d"
 #define FILE_OUT_P   "output/pressure.gmv.%d"
#endif
 #ifdef T_EQUATIONS
 #define FILE_DOFST_CASE "data_in/dofsT.in"
 #define FILE_OUT_T00 "output/temperature.gmv.00%d"
 #define FILE_OUT_T0  "output/temperature.gmv.0%d"
 #define FILE_OUT_T   "output/temperature.gmv.%d"
#endif



#endif
```

A.3.3 Data file `data.h`

The data file `data.h` is reported below. See Section 2.3.2 for option description.

```
#ifndef __data_h__
#define __data_h__

// parameters ************************************

// time
#define ND_TIME_STEP  0.001
#define TIME          0.0

#define N_TIME_STEPS  4000
#define VOF_SUBSTEP   1

// physics ***************************************

#define Uref      1.
#define Lref      1.
#define Tref     1000.
#define RHOref   100000.
#define PI       3.141592653589793238
#define P_IN     36000.
#define T_IN     673.15

// fluid properties ----------------
// monophase
#define RHO0      11367.0
#define MU0       .0022
#define NUT       .0
#define KAPPA0    15.8
#define PRT       0.9
#define CP0       147.3
#define KOMP0    0.

// 2nd phase
#define RHO1      1.0
#define MU1       0.1
#define KAPPA1    1.0
#define CP1       1.0
#define KOMP1    0.

// surface tension
#define SIGMA   0.0725

// Heat source
```

```
 #define QHEAT 1.20249144e+8
//  temperature function
#define RT 1.

// Gravity
#define DIRGX   0.
#define DIRGY   0.
#define DIRGZ   0.
// *******************************************************
// Multigrid parameters ************************************
// *******************************************************
//#define ILUINIT 10
// Cycle controls ************************************
// grid0
const unsigned int grid0=0;     // level 0
//#define GRID0 0     // level 0
// gridn
const unsigned int gridn=1;      // level n
// "Number of grid levels:
const unsigned int NoLevels=gridn+1;
// "Number of intervals:
// const  size_t MaxNoInt = (size_t) n_int;
// "Maximum number of iterations (cycles)   ");
const int MaxIter=15;
//  Break off accurary for the residual:
const double Eps=1.e-6;

// Cycle type ************************************
//  Solution method  (MLSolverId):
//    1. multigrid                    = MGIterId     |
//    Nested multigrid iterations? (1. yes 0. no)    |  type
//                      (NestedMG=0/1)               | 1. V cycle
//    2. multigrid preconditioned CG  = MGPCGIterId  | 2. W Cycle
//    3. BPX preconditioned CG        = BPXPCGIterId
const IterIdType  MLSolverId = MGIterId;
// Nested multigrid iterations? (1. yes 0. no)
const bool NestedMG = 0;
//  Type of multigrid: ( 1. V cycle 2. W Cycle);
const int Gamma=2;
// number of multigrid iterations within one MGPCG step for case 2 (cycle)
const int NoMGIter=1;

// Restriction/Projection ************************
// Type of restriction operator(1. simple 2. weighted)
const int RestrType=1;

// Smoothing no coarse ***********************************
```

```
//  Number of pre-smoothing iterations
const int Nu1=8;
// Number of post-smoothing iterations
const int Nu2=8;
// Relaxation parameter for smoothing:
const double Omega=0.98;
//  "Smoothing method:" ------------------------------
//  Iterative methods: 0. Vanka
//                     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
const IterProcType SmoothProc = GMRESIter;
//const IterProcType SmoothProc = Explicit;
// Preconditioning for smoothing iterations: (PrecondProcType)
//    Preconditioning: 0. none    = (PrecondProcType)NULL
//                     1. Jacobi  = JacobiPrecond
//                     2. SSOR    = SSORPrecond
//                     3. ILU/ICH = ILUPrecond
//const PrecondProcType PrecondProc = (PrecondProcType)NULL;
const PrecondProcType PrecondProc = ILUPrecond;


// Coarse Grid ***********************************
// Number of iterations on coarsest grid: -------------
const int  NuC=40;
// "Relaxation parameter on coarsest grid: ------------
const double OmegaC=0.98;
// "Solution method on coarsest grid" -----------------
//  Iterative methods: 0. Vanka
//                     a.RungeKutta4
//                     b.explcit
//                     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
const IterProcType SolvProc = GMRESIter;
//const IterProcType SolvProc = Explicit;
// "pointer to Preconditioner (PrecondProcType) on coarsest grid:"
//    Preconditioning: 0. none    = (PrecondProcType)NULL
//                     1. Jacobi  = JacobiPrecond
//                     2. SSOR    = SSORPrecond
//                     3. ILU/ICH = ILUPrecond
const PrecondProcType PrecondProcC = (PrecondProcType)NULL;
//const PrecondProcType PrecondProcC = (PrecondProcType)ILUPrecond;

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

#endif
```