Imperial College London
Department of Earth Science and Engineering

Nuclear Thermal Hydraulics Analysis using Devito's Code Generation Framework

by Kenechukwu Michael Nwegbu

Supervisors: Prof. Christopher C. Pain, Dr. Gerard Gorman and Prof. Paul N. Smith

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy, September 2024

Statement of Originality

I herewith certify that all material in this thesis is a result of the research undertaken by me and any which is not my own work has been properly acknowledged.

Kenechukwu Michael Nwegbu

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

Simulating the fluid flow and heat transfer within nuclear power plants is an important part of nuclear safety and performance analysis. This could be to understand the performance of the reactor during normal operation or to study the reactors phenomena during severe accident scenarios. As nuclear power continues to be part of the global energy supply, new simulation codes are being developed with more accurate modelling capabilities. This has lead to an expansion in computational fluid dynamics (CFD) modelling for nuclear engineering. Despite this, thermal hydraulic codes are still the standard tool for performing simulations of a nuclear reactor core because they have been extensively validated.

A growing alternative to thermal hydraulic codes is the porous media approach within CFD. The reactor core is approximated as a porous media and uses thermal hydraulic correlations to capture the physics of the fluid flow. This flexibility allows porous media-computational fluid dynamics codes to be dual purposed as a thermal hydraulics code and a CFD code.

The aim of this thesis is to develop a porous-media computation fluid dynamics code that uses automatic code-generation to runs efficiently on GPU and CPU architectures. This has been developed with the following capabilities: a 3D numerical solver for the incompressible Navier-Stokes equations, subchannel models for subchannel scale thermal hydraulics analysis and an immersed boundary method for representing complex geometries. The efficient scaling on high performance computing is one of the key objectives of this project. This is why the solver has been written on Devito, a Python-based framework that supports parallel computing on CPU/GPU architectures. The partial differential equations can be written using a high-level syntax. From this syntax, the Devito compiler generates C code that can be optimized for different hardware architectures.

Acknowledgements

I would like to thank everyone who supported me during my PhD. My supervisor, Gerard Gorman, has given me guidance on software development and assistance for using the Devito software. My supervisor, Paul N. Smith, has constantly given me direction and relevant resources for nuclear thermal hydraulics analysis. Claire Heaney has constantly curated, edited and proof-read my work. All of your efforts are greatly appreciated.

The other contributors to the Devito software have also been incredible helpful. In particular, Fabio Luporini, Rhodri Nelson, George Bisbas, Edward Caunt and Zoe Liebowitz. I have enjoyed using the Devito software and working with you throughout this research. I am grateful to the Centre of Doctoral Training in Nuclear Energy Futures for funding this research. Jonathan Tate has tirelessly organized outreach events, international trips and social activities for the cohort of PhD students to participate in.

Then to my friends at Imperial College London, there are too many names to list so I would like to thank everyone collectively! This including the PhD students in the Radiation Transport group, the 1st cohort of the CDT in Nuclear Energy Futures, members of the NORMS group, other researchers in the AMCG and many other acquaintances in the ESE department. The social environment has made working in the office a pleasant experience. Finally, I would like to thank my family for the continuous support and encouragement. It has been a great help to have a large support network.

Contents

Statement of Originality **i**

Copyright Declaration **ii**

Abstract **iii**

Acknowledgements **iii**

1 Introduction **1**

1.1 Background 1

1.2 Motivation 4

1.3 Objectives 7

1.4 Thesis outline 7

1.5 Publications 9

2 Devito's code generation framework for solving partial differential equations 10

2.1 Introduction 10

2.2 Automatic Code Generation 13

2.3 Example problem - Burgers equation 14

2.4 Example problem - Navier-Stokes equations 17

2.5 Verification - lid driven cavity 20

2.6 Conclusions 21

3 Discretizing the porous media model in the Navier-Stokes equations 23

3.1 Introduction 23

3.2 Governing equations 26

3.3 Continuity equation and source term discretization 29

3.3.1 Generalized discretization of source terms 30

3.3.2 Discretization of explicit drag terms with variable porosity 32

3.3.3 Discretization of implicit drag terms with variable porosity 32

3.4 Solution procedure 33

3.4.1 Time integration and spatial discretization 33

3.4.2 Coupled pressure based solver 34

3.5 Numerical test cases 35

3.5.1 Inviscid porous plug test case 35

3.5.2 Viscous porous plug test cases 41

3.5.3 Joseph Beavers test case 42

3.5.4 Unsteady flow past a square block 43

3.6 Conclusions 44

4 Validation of single-phase subchannel models 46

4.1 Introduction 46

4.2 Governing equations and subchannel T/H models 48

4.2.1 Governing equations 48

4.2.2 Subchannel T/H models 48

4.2.3 Heat equation for nuclear fuel rod 51

4.3 Solution procedure 52

4.4 Verification of heat conduction model 52

4.4.1 Verification of steady state solutions 53

4.4.2 Verification of transient solution 59

4.5 Subchannel analysis validation 61

4.5.1 PNL 7×7 sleeve blockage test 61

4.5.2 Westinghouse two 14 × 14 assembly mixing test 64

4.5.3 Pacific Northwest Laboratories 2 × 6 buoyancy test 68

4.6 Conclusions 71

5 Nuclear reactor core simulation on CPU and GPU architectures 73

5.1 Introduction 73

5.2 Parallel computing using Devito 75

5.3 PWR core simulation 76

5.4 HPC cluster parallel 81

5.4.1 CPU nodes - OpenMP/MPI 82

5.4.2 GPU nodes - OpenACC 83

5.5 Conclusions 83

6 Fluid flow in a reactor pressure vessel 84

6.1 Introduction 84

6.2 ROCOM Facility 86

6.3 Governing equations and numerical methods 88

6.3.1 Physics models 88

6.3.2 Geometry simplification 89

6.3.3 Grid generation and immersed boundary methods 90

6.3.4 Boundary conditions 92

6.4 Simulation Setup 92

6.5 Numerical Results 94

6.6 Conclusions 96

7 Conclusions 97

7.1 Summary of Thesis Achievements 97

7.2 Applications 98

7.3 Future Work 99

A Supplementary code from Chapter 2 100

A.1 Python code for Burgers' equation 100

A.2 C code for Burgers' equation 101

A.3 Python code for the incompressible Navier-Stokes equations 103

A.4 C code for the incompressible Navier-Stokes equations 104

List of Tables

3.1 Numerical and analytical solutions for the invisicid porous plug test case 41

3.2 Strouhal number comparison between studies at Re 100 44

4.1 Friction factor correlations 49

4.2 Parameters for the fuel cylinder 54

4.3 Parameters for the fuel cylinder 56

4.4 Parameters for the fuel cylinder 59

4.5 Parameters for the fuel cylinder 60

4.6 Parameters for PNL 7x7 assembly test 62

4.7 Parameters for WH 14 $\times$ 14 assembly test 65

4.8 Parameters of the PNL 2 x 6 assembly 69

5.1 Parameters for fuel assembly 77

5.2 Parameters for reactor core 80

5.3 Young computer nodes used for the performance evaluation 82

6.1 Parameters for ROCOM experiment 93

6.2 Numerical methods 93

List of Figures

2.1 Overview of the Devito architecture from the symbolic expressions to running the code 13

2.2 Computed domain for the porous plug test cases. The white sections are the open fluid regions while the central grey section is the porous region. 20

2.3 The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$ 21

2.4 The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$ 21

3.1 Node locations in the x-direction 29

3.2 Flowchart of the solution strategy where P^{n+1} and $\boldsymbol{u}^{n+1}$ are solved simultaneously using Equations (3.21) and (3.43). 35

3.3 Computed domain for the porous plug test cases. The white sections are the open fluid regions while the central grey section is the porous region. 36

3.4 The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$ 37

3.5 The pressure plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$ 37

3.6 The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.04$ 38

3.7 The pressure plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.04$ 38

3.8 Inviscid flow through a porous plug where $D_a = 10^{-5}$ 39

3.9 Inviscid flow through a porous plug where $D_a = 10^{-5}$ 39

3.10 Inviscid flow through a porous plug where $D_a = 10^{0}$ 40

3.11 Inviscid flow through a porous plug where $D_a = 10^{0}$ 40

3.12 A zoomed in view of the pressure drop where $D_a = 10^{-5}$ 40

3.13 A zoomed in view of the pressure drop where $D_a = 10^{0}$ (right) 41

3.14 Viscous flow through a porous plug where $D_a = 10^{-3}$ 42

3.15 Viscous flow through a porous plug where $D_a = 10^{-3}$ 42

3.16 Computed domain for the Joseph-Beavers test case 42

3.17 Velocity profile for the flow parallel to a porous region 43

3.18 Vortex shedding at Re = 100 44

4.1 Layout of a fuel rod 51

4.2 Flowchart of the solution strategy 52

4.3 Steady state calculation comparing Equation (4.17) to the numerical solution . . 55

4.4 Spatial convergence using difference stencils for the Robin boundary condition . 55

4.5 Steady state calculation comparing Equations 4.26 and 4.27 to the numerical solution 57

4.6 Steady state calculation comparing Equations (4.39) and (4.40) 60

4.7 Transient calculation comparing Equation (4.45) to the numerical solution . . . 61

4.8 Layout of the PNL 7 × 7 assembly . . . 62

4.9 Normalized axial velocity along the centre subchannel with a 70% blockage . . . 63

4.10 Normalized axial velocity comparison between different numerical solvers . . . 63

4.11 The velocity at different levels for PNL 7 × 7 . . . 64

4.12 Layout of the WH 14 x 14 assemblies . . . 65

4.13 A comparison between the present study, the experimental results and CUPID's porous media code . . . 66

4.14 The velocity at each level for the WH 14 x 14 assembly test . . . 67

4.15 Layout of the PNL 2 x 6 assembly . . . 68

4.16 A comparison between the experimental results, resolved CFD results from Liu *et al.* [19], MATRA's subchannel results and the present study . . . 70

4.17 The temperature at different axial positions for the PNL 2 x 6 assembly test . . . 71

5.1 Axial and horizontal power factor within a single fuel assembly . . . 76

5.2 The Power factor of the PWR core. Each 0.215×0.215 assembly in the core has the power distributions shown above in Figure 5.1 . . . 77

5.3 Pellet temperature of the fuel assembly . . . 78

5.4 Cladding temperature of the fuel assembly . . . 78

5.5 Coolant temperature of the fuel assembly . . . 79

5.6 3D plot of the coolant temperature in the fuel assembly . . . 79

5.7 The power density in the PWR core . . . 81

5.8 The pellet centerline temperature in the PWR core . . . 81

5.9 The cladding temperature in the PWR core 81

5.10 The coolant temperature in the PWR core 81

5.11 The speed up for strong scaling 82

6.1 Sketch of the ROCOM facility [107] showing the reactor pressure vessel, and the cold and hot legs 87

6.2 Mixing scalar quantity of salt and water in cold leg 88

6.3 Original CAD geometry of the ROCOM test facility 89

6.4 Simplified CAD geometry of the ROCOM test facility 90

6.5 SDF of the reactor pressure vessel 92

6.6 Velocity distribution in the simulations results 94

6.7 The tracer concentration at the inlet in the experiment and the numerical model 95

6.8 Tracer distribution in the simulation results 96

Chapter 1

Introduction

1.1 Background

Nuclear energy has been an important part of the global energy supply, providing around 10% of the world's electricity production. Throughout its history, there has been a continuous effort to improve reactor designs regarding safety, cost and efficiency. However, developing new reactor technologies can be a long and expensive process. This is why modelling and simulation has been an essential part of nuclear engineering. It helps nuclear engineers understand the behaviour of nuclear systems alongside the experimental testing.

Currently, there is a renewed interest in nuclear energy due to more focus on energy security and reducing greenhouse emissions. Generation IV reactors - designed to be the successor of older technologies - have began commercial operation with new high-temperature gas reactors. Small modular reactors - reactors designed for factory fabrication - are being developed in many countries globally. Nuclear microreactors - reactors ranging from 1MW to 20 MW - are being designed to provide power in remote environments. To meet the needs of these new reactor designs, there is ongoing research to develop new simulation software. Areas of interest includes fuel performance, material properties and nuclear thermal hydraulics. In this project, the field of interest is simulation methods for nuclear thermal hydraulics.

Nuclear thermal hydraulics is the study of the fluid dynamics and heat transfer within nuclear reactors. It is an essential discipline for nuclear reactor safety and performance optimization. For analysing nuclear power plants (NPP), thermal hydraulics is separated into three scales of size depending on the behaviour being investigated: system scale, component scale and mesoscale analysis. System scale analysis evaluates the performance of the entire nuclear system. For a pressurized water reactor (PWR), this could be the entire primary circuit where heat is removed from the reactor core, or the secondary circuit where steam is generated to turns a turbine which produces electricity. Component scale evaluates the phenomena in specific reactor components. This could be a single fuel assembly, the entire reactor core, or a steam generator. Mesoscale analysis evaluates the complicated fluid behaviour at a fine scale. One example would be the formation of bubbles on a fuel element as heat is removed.

It would be a challenging task to develop a simulation code that is suitable for all three scales of size. Instead, codes have been developed for each scale. System scale analysis is performed by system codes. These solve the Euler equations and use one-dimensional (1D) empirical correlations to represent the thermal hydraulic characteristics. Many system codes such as RELAP 5, ATHLET, TRACE and CATHARE have been widely used to simulate nuclear power plants [1]. However, the 1D models have significant limitations when three dimensional (3D) effects play an important role. A simple channel flow test case by Schultz *et al.* [2] shows one of these limitations. In this test case, RELAP-3D produces inaccurate results for the velocity profile of the fluid because the Euler equations neglects the stress tensor for viscous flow.

System scale analysis produces the boundary conditions such as the inlet temperature and velocity, which are used for component scale analysis. Component scale analysis is then conducted using subchannel codes which predict the thermal safety margins. In subchannel codes, the viscid momentum equation is only solved in the axial direction of fluid flow. The transverse directions neglect viscous and convective forces. Subchannel codes such as COBRA-en[3], COBRA-TF [4] [5], MATRA [6] and SUBCHANFLOW [7] have also been extensively validated against several internationally recognized benchmarks. A big advantage of these codes is that they have been derived from legacy COBRA codes and then undergone a long development

process. COBRA-en is an upgraded version of COBRA-3C/MIT, COBRA-TF is derived from the original COBRA/TRAC version, MATRA is based on a mainframe version of COBRA-IV-I and SUBCHANFLOW is also derived from the COBRA family. While this is a step above system codes, they also have limited capabilities for modelling 3D fluid phenomena. One example of this is a study by Chen *et al.* [8] comparing COBRA-TF with ANSYS Fluent. COBRA-TF was noticeable less accurate for simulating the cross flow between mixing vanes. Both system and subchannel codes are classified as thermal hydraulics codes and are the industry standard for nuclear safety and performance analysis.

Mesoscale analysis is performed by computational fluid dynamics (CFD) codes, which are well suited for modelling a complex 3D flow field. Examples of the application of CFD include understanding pressurized thermal shock, fuel rod degradation, radiation damage, or turbulent flow in the plenum [9]. CFD yields a more accurate representation of viscous fluid flow, as the Navier-Stokes equations solve convection and viscous effects in all spatial directions. However, representing many complex components in nuclear reactors (grid spacers, fuel rods, mixing vanes) is not always practical within CFD simulations. In a steam generator, for example, resolving a sufficient mesh to represent more than 10000 tubes is not suitable for performing timely simulations.

Coupling simulation codes that resolve different scales of size is a useful tool for zooming in on a particular section [10]. This uses the advantage of each method, as the fluid phenomena can be observed at both the smaller scale and the macro-scale simultaneously. There are numerous examples of multi-scale coupling for thermal hydraulics analysis. A few examples of system code - subchannel code coupling include RELAP5 - COBRA [11], ATHLET - COBRA-TF [12], and TRACE - SUBCHANFLOW [13]. Examples of system code - CFD code coupling include RELAP5-3D - ANSYS CFX [14], RELAP5-3D - ANSYS Fluent [15] and TRACE - STAR-CCM+. Then, there are also studies on subchannel - CFD coupling such as SUBCHANFLOW - TrioCFD [16]. A major drawback of multiscale coupling is that it often requires a complicated coupling framework between the two codes. Firstly, each code will have a different Application Programming Interface (API) for exchanging data. If the code is proprietary, this adds an extra layer of difficulty in accessing the API. On top of that, each code will also have a different

grid mesh that divides the geometry into subdivisions. If there is a vast difference in mesh size, such as the different between a fine CFD mesh and a coarse system code mesh, a robust interpolation method is necessary. Furthermore, care must be taken with the coupling methods for solving the equations at each time step. The simplest method, explicit coupling transfers the data between the codes once at the beginning or at the end of each time step. Semi-implicit and implicit coupling methods transfer the data multiple times at each time step. This is more robust than explicit coupling, but the implementation is more complex. Coupling methods needs to be chosen that are sufficiently easy to implement and produce accurate results. In comparison, a multi-scale simulation code avoids a lot of the complexity of code-coupling. This is a code that can resolve different scales of size depending on the numerical model that is used. Although this loses the advantages of code-coupling, which leverages the extensive models from two separate numerical codes, the simpler solution strategy is a major advantage. This is why despite the improvements in coupling methods, there is still active research into multi-scale numerical codes.

1.2 Motivation

Porous-media CFD codes are multi-scale numerical codes that are used for nuclear thermal hydraulics. They use the porous-media approach, which approximates the fluid flow through porous structures using volume-averaged properties. Using the steam generator again as an example, instead of representing the solid structure of 10000 tubes, it is more efficient to represent the whole region as a porous media. Each cell in the mesh will have volume-averaged properties. Then, additional source terms are applied in the governing equations to represent the macroscopic effects of the steam generator tubes to the fluid flow. These source terms could be empirical correlations from subchannel codes which also use the porous media approach. Consequently, the porous media CFD code can be used for subchannel analysis in the porous regions of the steam generator. In the open regions, where there are no solid structures, this equation simplifies to the standard Navier-Stokes equations. This example shows how porous-media CFD codes can do both subchannel and CFD analysis in a nuclear system.

Many studies have used porous-media models to simulate different reactor designs. Examples of this include the flow simulation in a sodium fast reactor fuel assembly [17], a VVER-100 reactor core [18], a PWR reactor core [19] and the thermal hydraulics inside the entire reactor pressure vessel [20]. Many codes have also been implemented with two-phase flow models, which can be used to predict the void fraction in rod bundles[21]. A few 3D two-phase porous media codes have been developed consistently with numerous studies on their code validation. CUPID, PORFLO and TWOPORFLOW are examples of codes that have been actively developed during the last decade. CUPID has been used for several applications including the flow in fuel rod bundles, a PWR core [22], a steam generator [23] and a reactor pressure vessel [24]. PORFLO has been used in several applications including the fluid flow in an condenser[25], boiling water reactor (BWR) fuel bundles[26], a steam generator[27] and a reactor pressure vessel [28]. TWOPORFLOW has been validated with BWR experiments [29] and liquid cooled metal reactor experiments [30]. There are also many open source CFD codes that have been extended with porous media models for nuclear thermal hydraulics. GenFOAM is an open source multiphysics solver developed on the OpenFOAM open source CFD software [31]. This solver was validated for single-phase heat transfer [32] and sodium boiling experiments using a Eulerian two-phase flow model [33]. CorTAF is another porous-media code developed on OpenFOAM. This has been used for both subchannel and CFD analysis of the flow within fuel assemblies [19]. An alternative to OpenFOAM is MOOSE [34], an open source finite element framework for multiphysics simulations. MOOSE has been used for thermal hydraulics analysis, with one example being Pronghorn, a porous media code built on this framework [35]. Alongside the development of new porous media codes, there is also research on Coarse-Grid-CFD (CGCFD) solvers [36] [37]. These are CFD simulations that are performed on a coarse mesh to reduce the computing costs. The advantage here is that they can capture more details of the flow profile compared to porous media models. But, there are some restrictions on where coarse grid CFD methods are applicable. In fuel assemblies with complex structures (such as mixing vanes and spacer grids) a very refined mesh would be needed to accurately represent the geometry. In such cases, it is more efficient to use a porous media model and incorporate the complex sub-structures using source terms. As an example, SubChCFD is a coarse grid

CFD solver developed on the open source CFD code Saturn. For simulation with complex substructures, a porous media model was used in the solution procedure of the CFD solver. This showed good agreement to the resolved CFD simulations that were performed using the same code [38]. The studies presented here demonstrate the flexibility of porous media codes for thermal hydraulics analysis. They have been used for CFD analysis, subchannel analysis and even coupled with system codes for system scale analysis [15], [39].

In terms of computing resources, the porous-media CFD approach is a reasonable compromise between thermal hydraulic codes and full resolved CFD. However, it still requires considerable computing resources to perform large-scale simulations. Parallel computing is essential for allowing large scale simulations to be performed efficiently. Simulations can be run on multiple threads and multiple processor simultaneously which accelerates the compute time. Both Gen-FOAM and CUPID [22] use the Message Parsing Interface (MPI) framework for parallel computing on Central processing units (CPUs). Similarly, a few subchannel codes use both MPI and OpenMP [40], [41] [42]. But, there is less support for running on Graphic Processing units (GPUs) for many of these numerical codes. Although many of the stated CFD solvers such as Open-FOAM and ANSYS Fluent support GPU accelerated solvers. It is still a work in progress to redevelop the entire code as a native GPU solver. Given the immense performance of GPUs for numerical simulations, it would be useful to have GPU based porous media solver. One way of doing this is using a domain-specific language that already supports CPU/GPU architectures. This makes the development easier without needing to focus on the performance optimization. There are a few Python-based domain specific languages (DSL) that already support heterogeneous architectures. These have been applied to numerically solving partial differential equations for multiple different fluid dynamics applications [43, 44] .

Devito is a DSL for automatically generating finite difference stencils [45]. Similar to other packages, its a Python-based framework that allows mathematical equations to be written using a symbolic language. From these equations, the Devito compiler generates C code that can be optimized to whichever architecture the code is being used on (CPUs, GPUs, a cluster etc). For writing a porous media code, a Python based language significantly reduces the code base and likeliness of coding errors compared to a low level language. Also, this enables

performing large scale simulations of nuclear systems on HPC clusters.

1.3 Objectives

The aim of this research is to develop a porous media CFD solver on Devito that scales efficiently on CPU/GPU architectures. A porous media model within a CFD code enables subchannel analysis and CFD analysis of nuclear systems. Additionally, the efficient performance on heterogeneous architectures enables complicated nuclear systems to be simulated rapidly. This capability can be shown for component scale analysis to demonstrate the potential for system scale analysis. Within this research aim, there are four main objectives:

- Develop an accurate discretization of the volume-averaged Navier-Stokes equations in a porous medium.
- Implement single phase sub-channel models and validate the solver using thermal hydraulic benchmarks.
- Demonstrate the solvers scaling performance on CPU/GPU architectures
- Simulate the fluid flow within a reactor pressure vessel.

1.4 Thesis outline

In the second chapter, an overview is given on the Devito framework and how it is used to solve partial differential equations. Then, example problem scripts are shown including a pressure-based CFD solver. In the third chapter, robust formulations are implemented for solving the volume-averaged Navier-Stokes equations in a porous medium. These formulations are validated against steady state porous media test cases. The fourth chapter explains subchannel thermal hydraulic (T/H) analysis and shows how subchannel models are implemented into a CFD code. The fifth chapter explains the advantages of using Devito for parallel computing. This is demonstrated by simulating a pressurized water reactor core on different hardware

architectures. The final chapter shows the simulation results for the Rossendorf Coolant Mixing Model (ROCOM) experiments, which is for single-phase flow within a reactor pressure vessel.

1.5 Publications

Journal publications in preparation:

Kenechukwu M. Nwegbu, Claire E. Heaney, Alan V. Jones, Gerard J. Gorman, Christopher C. Pain and Paul N. Smith, *A porous media code using automatic code generation on a variety of computing architectures*, 2025.

Conference proceedings:

Kenechukwu M. Nwegbu, Christopher C. Pain, Gerard J Gorman and Paul N. Smith, *A coupled pressure based solver for simulating variable porosity flows on coarse grids*, 76th Annual Meeting of the Division of Fluid Dynamics, 2023.

Kenechukwu M. Nwegbu, Christopher C. Pain, Gerard J Gorman and Paul N. Smith, *A Porous Media Code for Reactor Pressure Vessel Analysis using Automatic Code Generation*, ANSWERS Seminar 2023.

Kenechukwu M. Nwegbu, Claire E. Heaney, Alan V. Jones, Gerard J. Gorman, Christopher C. Pain and Paul N. Smith, *Modelling flows in nuclear fuel rod bundles using automatic code generation*, 5th Annual Modelling in Nuclear Science and Engineering Seminar, 2022.

Kenechukwu M. Nwegbu, Claire E. Heaney, Alan V. Jones, Gerard J. Gorman, Christopher C. Pain and Paul N. Smith, *Nuclear thermal hydraulics modelling using automatic code generation*, ECCOMAS Congress (MS151A), 2022.

Chapter 2

Devito's code generation framework for solving partial differential equations

In this chapter, an overview of the domain-specific language Devito is given and example test cases for fluid dynamics problems are shown. Domain-specific languages allow mathematical equations to be written symbolically in a programming script. These languages have been widely used for fluid dynamics problems in science and engineering fields such as aerospace engineering, weather forecasting, chemical engineering and nuclear engineering. Devito is a domain-specific language that has mainly been used for geophysics problems. The flexibility of the framework for defining any partial differential equation means that it is also suitable for computational fluid dynamics. To demonstrate this, a problem script is shown for solving the incompressible Navier-Stokes equations in two dimensions. This is verified against the lid-driven cavity benchmark.

2.1 Introduction

Partial differential equations (PDEs) are used to model a variety of physical systems. This includes heat transfer, mass transfer, fluid dynamics, structural mechanics, and sound propagation. In many problems, there might not be an analytical solution that satisfies the equations.

This is why numerical techniques for solving PDEs are one of the major research areas in computational science. Continuous research is underway in search of new numerical techniques for ever more challenging science and engineering problems.

Computational fluid dynamics (CFD) in particular has had extensive research into novel numerical methods. There are a vast range of applications of CFD, from the air flow around wind turbines to the heat transfer in heat exchangers. Running CFD simulations similar to these examples is very computationally expensive. They require large amounts of memory and lots of complex calculations on large amounts of data. To run these simulations in a reasonable time, it is necessary to prioritize high performance and execution speed. Compiled languages such as Fortran, C and C++ are preferred for writing CFD codes because of their execution speed and extensive scientific libraries.

Computer hardware is constantly changing as more powerful central processing units (CPUs) are continually being released. The biggest change has been the substantial growth of graphic processing units (GPUs) for high performance computing. GPUs have exceptional capabilities for parallel computing, which enables many tasks to be performed simultaneously. A single GPU can drastically reduce simulation times compared to multiple CPUs. To enhance the performance of CFD codes it is necessary to port them to the latest hardware architectures. But, porting onto new architectures can be a time consuming process. Each processor has a unique design and different support for software frameworks which makes it challenging for developers. The development process for a new CFD code is then split between implementing new physics models for fluid dynamics, developing numerical methods to solve the equations and improving the performance for running on the latest hardware.

Domain-specific languages (DSL) are one way of reducing the complexity of developing a new CFD code. The mathematics can be written in a high-level abstraction which closely resembles the governing equations. The code for the performance optimization and execution is a separate layer which the user does not have to interact with. This makes it much easier to focus on the governing equations and the numerical methods. The extensive usage of the C++ finite volume code OpenFOAM [31] shows the benefit of using a DSL. In OpenFOAM the

differential operators for divergence gradient and curl are written as `div`, `grad`, and `curl` to define equations. New OpenFOAM-based solvers can quickly be developed using this syntax.

An even higher level of abstraction is to write a DSL in an interpreted language. Interpreted languages such as Python are easy to use and portable to difference machines. Python also has the benefit of a wide user base and many scientific libraries such as NumPy, SciPy and SymPy. A few software libraries combine Python-based DSLs with automatic code generation. The equations which are written in Python, then code-generation packages convert this to C/C++ code. This leverages the easy use of an interpreted language with the performance of a complied language. A popular open source library that uses this approach is the FEniCS project [46] for solving PDEs with the finite element method. This uses the unified form language (UFL), which is a domain-specific language to represent the weak formulations of PDEs. FEnICS has been used for many fluid dynamics applications, including aerodynamics [47], tidal turbines [48] and porous media models in heat exchangers [49]. Firedrake [43] is another software library that uses the unified form language (UFL) to express PDEs. It is mainly used for geophysics applications such as atmospheric modelling [50], ocean modelling [51] and glacial flow modelling [52]. PyFR [53] uses the flux reconstruction approach to solve PDEs. It has been used for aerospace applications [54] and hydrodynamic problems such as the fluid flow around a submarine [55]. OpenSBLI [44] uses finite difference schemes to solve PDEs. The main application is simulating shock wave boundary layer interactions in high Mach number flows [56].

In this work, an overview will be given on using Devito to solve PDEs. Similarly to OpenSBLI, Devito uses finite-difference schemes to discretize the governing equations and uses a code-generation library to produce C code. The main application has been seismic imaging techniques such as full wave form inversion. Devito supports fully explicit time integration schemes which allows these simulations run quickly on multi-core platforms. In contrast, simulation methods for incompressible fluid flow often need implicit schemes. This is especially the case for porous-media models and nuclear thermal hydraulics, where pressure-based methods are preferred. In pressure-based solvers, an implicit pressure equation is solved using iterative solvers. The Jacobi iterative solver has been implemented within Devito for solving the incompressible Navier-

Stokes equations.

In Section 2.2, a high-level overview of the Devito framework is given. Then in Section 2.3, the Devito API is shown using Burgers equation as an example problem. Section 2.4 shows how iterative loops can be created within Devito. This is used to solve the incompressible Navier-Stokes equations. Finally, in Section 2.4, the incompressible Navier-Stokes solver is verified against the lid-driven cavity test case.

2.2 Automatic Code Generation

Devito is a Python library for numerically solving PDEs. The mathematical equations can be represented symbolically, as the Devito DSL uses SymPy for symbolic mathematics. This allows Devito to represent any PDE include non linear equations, coupled systems and steady state equations. Then, the symbolic expressions are discretized using finite difference methods. While the main use case is for writing PDE solvers, the DSL also contains functions for linear algebra operations and interpolation. Figure 2.1 shows a high-level overview of the Devito architecture.

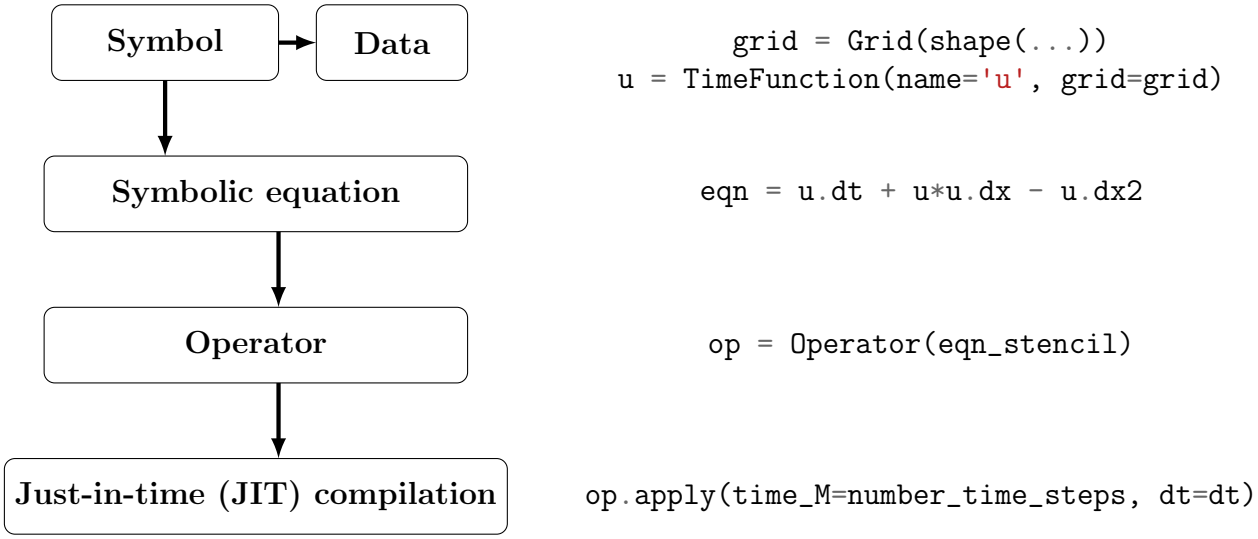


Figure 2.1: Overview of the Devito architecture from the symbolic expressions to running the code

The Devito `Operator` has three fundamental tasks: generation of C code from the symbolic expressions, just in time compilation and execution. The generation of C code is a multi-stage

compilation process. At each stage, the input goes through a series of operations to produce an output for the next stage. The code at each stage can be seen as an intermediate representation until the final stage. In the final stage, a CGen tree is used to produce C code. A detailed explanation of the code generation process in the Devito operator is given by Luporini *et al.* [45]. As a guide, bullet points of the main stages are given below

- Equations lowering
- Local analysis
- Clustering
- Symbolic optimization
- Iteration/Expression Tree (IET)
- Synthesis
- JIT compilation

The C code is JIT compiled to a shared object. This is done once on the first call to carry out execution. JIT compile and execution are initiated by calling on `Operator.apply`.

2.3 Example problem - Burgers equation

Burgers equation is a non-linear PDE for convection and diffusion. It is a simplification of the Navier-Stokes equations by removing the pressure gradient term. This makes it ideal for analysing numerical schemes and fluid flow properties with a simple model. The vector form of the equation is given in 2.1.

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = \nu \nabla^2 \boldsymbol{u} \tag{2.1}$$

In Devito, the 2D Burgers equation will be implemented. The equations for the velocity components u and v are given below.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.3}$$

The first step in the implementation is to define a computational grid that will be used for the numerical simulations. This is done using a `Grid` object where the `shape` is the number of grid points and the `extent` is the physical size of the grid.

```
import devito as dv

width = 1
nx = 41
ny = 41

grid = dv.Grid(shape=(nx, ny), extent=(width, width))
```

A `TimeFunction` is a `Function` that is both spatially varying and time-dependent. These objects are defined on the grid where they will have the same spatial dimensions. In this case, the `TimeFunction` are the u and v velocity components.

```
# Constants
nu = .1

# Variables are automatically initalized at 0.
u = dv.TimeFunction(name='u', grid=grid, space_order=2)
v = dv.TimeFunction(name='v', grid=grid, space_order=2)
```

Devito supports shorthand notation for writing derivative.`u.dt` is the time derivative, `u.dx` is the spatial derivative and `u.laplace` is the Laplacian. These terms are all discretized suing finite difference stencils.

```
# Burgers equation in 2 dimensions
eq_u = dv.Eq(u.dt + u*u.dx + v*u.dy,  nu*(u.laplace),
 ↪  subdomain=grid.interior)
eq_v = dv.Eq(v.dt + u*v.dx + v*v.dy,  nu*(v.laplace),
 ↪  subdomain=grid.interior)
```

The equations are rearranged to form an expression for $\boldsymbol{u}^{n+1}$, which is updated at every time step. First, the time derivative $\frac{\partial \boldsymbol{u}}{\partial t}$ is approximated as $\frac{\boldsymbol{u}^{t+1}-\boldsymbol{u}^{t}}{\Delta t}$. Then, the whole equation is rearranged to form a stencil for $\boldsymbol{u}^{n+1}$.

```
# Let SymPy rearrange our stencils to form the update expressions
stencil_u = dv.solve(eq_u , u.forward)
stencil_v = dv.solve(eq_v , v.forward)

update_u = dv.Eq(u.forward, stencil_u)
update_v = dv.Eq(v.forward, stencil_v)
```

Boundary conditions can be specified in the same way as equations in the interior domain. These are then lowered in the operator to form indexed expressions. Alternatively, the indexed expression can be defined directly using the low-level API as shown below. `t` is used to index the previous time step `u(t, x, y)` and the current time step `u(t + dt, x, y)`. The symbols `x` and `y` are used for indexing the spatial dimensions. After using these indexes, all of these expressions for $\boldsymbol{u}^{n+1}$ can now be passed to the `Operator`. These are solved for a specified time step `dt` and number of time steps `time_M`.

```
t = grid.stepping_dim
x, y = grid.dimensions
# Create Dirichlet BC expressions using the low-level API
bc_u  = [dv.Eq(u[t+1, 0, y], 0)]
bc_u += [dv.Eq(u[t+1, nx-1, y], 0)]
bc_u += [dv.Eq(u[t+1, x, 0], 0)]
bc_u += [dv.Eq(u[t+1, x, ny-1], 1)]  # u=1 for y=2
bc_v  = [dv.Eq(v[t+1, 0, y], 0)]
bc_v += [dv.Eq(v[t+1, nx-1, y], 0)]
bc_v += [dv.Eq(v[t+1, x, ny-1], 0)]
bc_v += [dv.Eq(v[t+1, x, 0], 0)]

# Time stepping dimensions
nt = 1000
delta_t = .001

op = dv.Operator([update_u, update_v] + bc_u + bc_v )
op(time_m=0, time_M=nt, dt=delta_t)
```

The full python script and C code are provided in the appendix.

2.4 Example problem - Navier-Stokes equations

In this section the 2D incompressible Navier-Stokes equations are implemented in Devito using a pressure-based solver. The vector form of the equations are given as

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} \tag{2.4}$$

$$(\nabla \cdot \boldsymbol{u}) = 0 \tag{2.5}$$

$$\tag{2.6}$$

As shown above, the continuity equation has no time dependence. The momentum equation can be substituted into the continuity equation to form an elliptical pressure equation. The equations for the velocity components and pressure can now be written as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.7}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.8}$$

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \rho\left[\frac{\partial}{\partial t}\left(\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\right) - \left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)\right] \tag{2.9}$$

The first step in the Devito implementation is identical to the previous example, where a `Grid` object and `TimeFunctions` are defined.

```
import devito as dv

width = 1
nx = 41
ny = 41

grid = dv.Grid(shape=(nx, ny), extent=(width, width))

#Constants
```

```
nu = .1

#Variables are automatically initalized at 0.
u = dv.TimeFunction(name='u', grid=grid, space_order=2)
v = dv.TimeFunction(name='v', grid=grid, space_order=2)
```

In the previous example, only one time loop was needed. The index used for specifying a time loop can be accessed from the `Grid` object.

```
# Time stepping dimensions
time = grid.time_dim
```

In this example, a pressure equation is solved implicitly, which needs an additional time loop. The symbol `i` has been defined for pseudo time stepping (as there is no actual time advancement) within the main time loop. The symbol `i_1` is for indexing the variable at the current time step value. The variable `p` can now be defined using this index.

```
# Iterative time dimension (within each time step)
i = dv.TimeDimension(name='i', parent=time)

# Iterative stepping dimension (within each time step)
i_1 = dv.SteppingDimension(name='i_1', parent=i)

#Variable p, defined for the iterative dimension, is initalized at 0.
p = dv.TimeFunction(name='p',  grid=grid, time_dim=i_1, space_order=2)
```

The partial differential equations from 2.9 are written using the symbolic syntax. `u.dxc` is an abbreviation for using the central differencing finite difference stencil.

```
# First order derivatives will be handled with p.dxc
eq_u = dv.Eq((u.dt + u*u.dx + v*u.dy), -p.dxc + nu*(u.laplace),
 ↪  subdomain=grid.interior)
eq_v = dv.Eq((v.dt + u*v.dx + v*v.dy), -p.dyc + nu*(v.laplace),
 ↪  subdomain=grid.interior)

delta_t = .001
eq_p = dv.Eq(p.laplace,
 ↪  1./delta_t*(u.dxc+v.dyc)-(u.dxc*u.dxc)-2*(u.dyc*v.dxc)-(v.dyc*v.dyc),
 ↪  subdomain=grid.interior)
```

These equations are then rearranged to form stencils for `u`, `v` and `p`. The Jacobi iterative method has been used to solve for `p` within the implicit time loop.

```
# NOTE: Pressure has no time dependence so we solve for the other pressure
↪  buffer.
stencil_u = dv.solve(eq_u , u.forward)
stencil_v = dv.solve(eq_v , v.forward)
stencil_p = dv.solve(eq_p, p)

update_u = dv.Eq(u.forward, stencil_u)
update_v = dv.Eq(v.forward, stencil_v)

idims_i = (time, i,) + tuple(grid.dimensions)
update_p = dv.Eq(p.forward, stencil_p, implicit_dims=idims_i,)
```

Just as with the previous example, the low level API has been used for defined the boundary conditions. These equations are then passed to the `Operator`. In this solver, additional arguments are needed in the `Operator` for the implicit time loop.

```
t = grid.stepping_dim
x, y = grid.dimensions
# Boundary Conds. u=v=0 for all sides
bc_u  = [dv.Eq(u[t+1, 0, y], 0)]
bc_u += [dv.Eq(u[t+1, nx-1, y], 0)]
bc_u += [dv.Eq(u[t+1, x, 0], 0)]
bc_u += [dv.Eq(u[t+1, x, ny-1], 1)]  # except u=1 for y=2
bc_v  = [dv.Eq(v[t+1, 0, y], 0)]
bc_v += [dv.Eq(v[t+1, nx-1, y], 0)]
bc_v += [dv.Eq(v[t+1, x, ny-1], 0)]
bc_v += [dv.Eq(v[t+1, x, 0], 0)]

bc_p  = [dv.Eq(p[i_1+1, 0, y],p[i_1, 1,y], implicit_dims=idims_i,)]  # dpn/dx
↪  = 0 for x=0.
bc_p += [dv.Eq(p[i_1+1,nx-1, y],p[i_1,nx-2, y], implicit_dims=idims_i,)]  #
↪  dpn/dx = 0 for x=2.
bc_p += [dv.Eq(p[i_1+1, x, 0],p[i_1,x ,1], implicit_dims=idims_i,)]  # dpn/dy
↪  = 0 at y=0
bc_p += [dv.Eq(p[i_1+1, x, ny-1],p[i_1, x, ny-2], implicit_dims=idims_i,)]  #
↪  pn=0 for y=2
bc_p += [dv.Eq(p[i_1+1, 0, 0], 0, implicit_dims=idims_i,)]

nt = 1000
nit = 50
```

```
op = dv.Operator([update_p] + bc_p + [update_u, update_v] + bc_u + bc_v )
op(time_m=0, time_M=nt-1, i_m=0, i_M = nit, dt=delta_t)
```

The full python script and C code are provided in the appendix.

2.5 Verification - lid driven cavity

The lid-driven cavity is a well known test case for verifying CFD codes. The domain consists of a square cavity with three rigid walls and a moving lid which drives the fluid flow. The rigid walls have no slip boundary conditions while the moving lid has a velocity of 1 $m\ s^{-1}$.

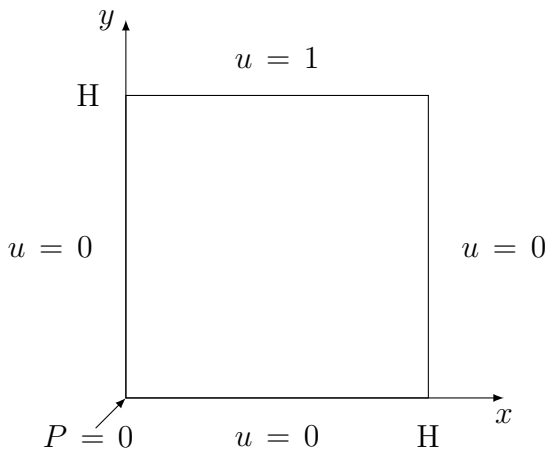


Figure 2.2: Computed domain for the porous plug test cases. The white sections are the open fluid regions while the central grey section is the porous region.

In the numerical model for this test case, a 40×40 cartesian mesh has been used. The equations have been discretized using second order finite difference schemes. The results of this are shown in the 2.3 below along with the 2D contour in 2.4. From 2.3 it is evident there is good agreement between the present study and the numerical solution from. More advanced numerical schemes can be implemented using this solver on more difficult problems.

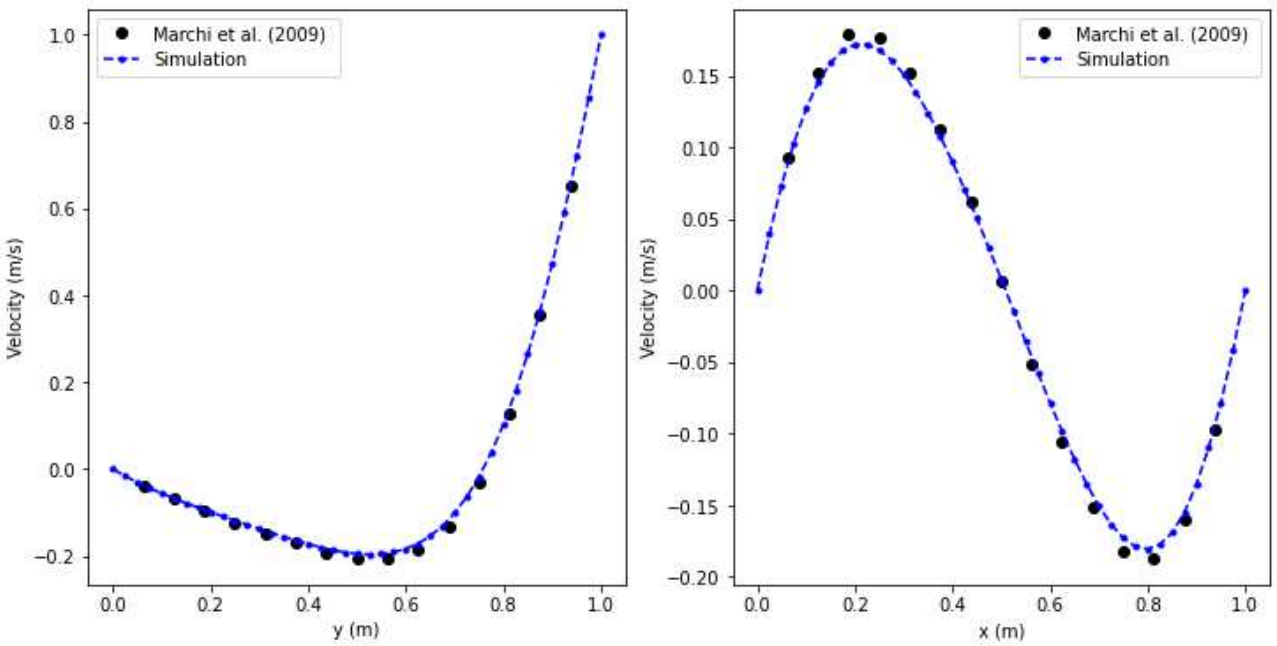


Figure 2.3: The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$

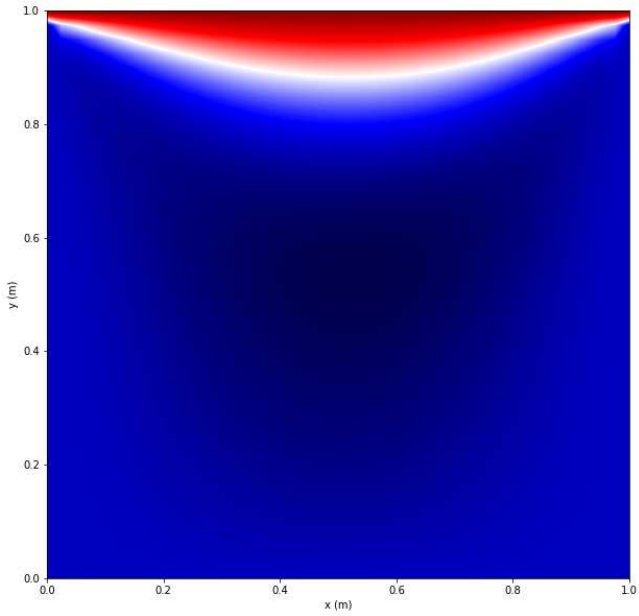


Figure 2.4: The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$

2.6 Conclusions

In this chapter an overview has been given for solving differential equations using Devito. The problem script for Burgers equation shows the implementation for explicit time stepping, while

the Navier-Stokes solver shows how to create additional iterative loops. Using this functionality, it is possible to implement more complicated iterative solvers.

Many practical problems for fluid dynamics include unsteady flows, high Reynolds numbers and large drag forces. This can lead to pressure-velocity decoupling if the implementation presented here is used. In the next chapter, a robust formulations will be shown for implemented a porous media model in a pressure-based solver. These formulations diminish the numerical instabilities caused by large drag forces and transient flows.

Chapter 3

Discretizing the porous media model in the Navier-Stokes equations

In this chapter, novel formulations for discretizing the volume-averaged Navier-Stokes equations in a porous medium are presented. These porous media models are used to model fluid flow in physical systems such as oil and gas reservoirs, catalytical converters and nuclear fuel assemblies. In many of these systems, the porosity and drag forces will vary considerably. This can cause numerical instabilities at discontinuities in porosity and permeability. To rectify this, the chapter begins by analysing the volume-averaged momentum equation to determine what causes instabilities at discontinuities in porosity. Then, an explicit discretization method is implemented that diminishes oscillations for explicit drag forces. Followed this, an implicit discretization method is implemented that diminishes oscillations for implicit drag forces. These methods are verified against analytical solutions for porous media flows.

3.1 Introduction

The fluid flow through a porous medium occurs in many natural and engineering systems. Petroleum reservoirs, aquifiers, nuclear reactor cores and fuel cells are a few examples of systems that can be modelled as heterogeneous porous media. To understand how the fluid behaves,

computational fluid dynamics simulations are an important tool. This is mainly viewed from two scales of analysis: the pore scale and the Darcy scale. For pore scale simulations, the fluid flow between individual solid particles is modelled using microscopic transport equations. This requires an accurate representation of the solid geometry in the numerical simulation. If a complicated solid geometry is used, this would require a high computational cost. The alternative approach for large simulations is the porous media model at the Darcy scale. This uses a much coarser approximation of the solid geometry by volume-averaging the liquid and solid regions.

In many engineering applications, the flow can be assumed to be incompressible. Pressure-based solvers are are the preferred method for these problems, where the continuity equation is solved in terms of pressure. However, these solvers are susceptible to pressure-velocity decoupling, especially with large discontinuities in the flow field. For finite-difference and finite-volume schemes, many studies have explored the treatment of discontinuities in porous media flows using a pressure based solver. Previously, the staggered grid arrangement was a favoured method for resolving pressure-velocity decoupling [57]. The pressure gradient is defined across the cell face in both the momentum and continuity equations using a first-order stencil. This ensures the pressure gradient is consistent across the coupled equations and avoids the checkerboard problem. Staggered grids are robust and stable methods even with large body forces. However, they can be difficult to implement on unstructured meshes. This is a major drawback for modern finite-volume CFD codes where unstructured meshes can be used to efficiently represent complex geometries or complicated flow fields. Collocated grids can handle this flexibility, which is why they are the focus of recent work.

Many of the recent methods on collocated grids are based on the Rhie-Chow interpolation for pressure velocity decoupling [58]. In the Rhie-Chow interpolation, the pressure gradient is defined across the cell face using a first-order stencil, replicating the staggered grid formulation. While the original formulation is suitable for steady state simulation, it has limitations for transient simulations or large discontinuities which cause numerical oscillations. Improvements have been developed in more recent work [59], [60], [61], in particular by Zhang *et al.* [62] where many of these formulations were summarized. In that study, the explicit source terms in the

momentum equation are reformulated using linear interpolation and gradient reconstruction. This formulation showed good agreement with the verified solutions for low permeabilities. However, for the test cases with variations in porosity, numerical errors were present at the discontinuities. Bartholomew *et al.* [63] used a similar approach for the explicit source terms and also demonstrated a method for discontinuous densities. While both of these explicit methods are suitable for moderate source terms, large source terms require implicit treatment to maintain numerical stability. One method for discretizing implicit source terms used a redistribution method to smooth out oscillations [64, 65]. Another study solved the implicit source terms using a multigrid solver [66]. However, the exact method for simulating sharp porosity discontinuities without generating numerical errors has not been presented in the literature.

In this work, discretization schemes for the volume-averaged momentum equation are presented, which diminish the numerical errors caused by discontinuities in porosity and permeability. The first part of this work is to compare the volume-averaged momentum equation from different studies. This is motivated by previous work, which shows that unphysical numerical oscillations can occur even when there are no drag terms. The present work aims to explain those findings by comparing the numerical results from different momentum equations. The next part of this work presents two formulations for the porous media drag terms. These are discretized on a collocated grid and use the Rhie-Chow interpolation method to avoid pressure-velocity decoupling. For source terms in a pressure-based solver, previous work has shown that the source term discretization must be consistent with the discretization of the pressure gradient for the scheme to be force-balanced [63]. This concept was applied in this work and then extended to variable porosity flows. No interface treatment is required at the porous-fluid boundary, a single formulation is used throughout the whole domain. The same approach has been used for the implicit formulation for discretizing the drag terms.

Section 3.2 shows the governing equations, Section 3.3 shows the explicit and implicit discretization schemes for the drag terms, then Section 3.5 shows the verification results. This verification is performed using test cases for 1D porous media flow with analytical solutions. The final test case is the flow past a square block, which shows the numerical model is valid

for unsteady flows.

3.2 Governing equations

The fluid flow through a porous solid can be modeled at the pore scale using microscopic transport equations. The microscopic momentum equation is given as

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u})\right) = -\nabla p + \mu \nabla^2 \boldsymbol{u} \tag{3.1}$$

where $\boldsymbol{u}$ is the velocity, ρ is the density, p is the pressure and μ is the viscosity. This is a useful approach if localized flow variables are needed, but it has a high computational cost for many industrial applications. Instead, macroscopic equations are normally used, which uses a much coarser representation of the physical system. In this method, the porous medium is divided into control volumes much larger than the solid particles. Given a control volume of size V, let V_f be the space occupied by the fluid. Then, the phase-averaged quantities are defined as the fluid quantities averaged over the entire control volume.

$$\langle \psi \rangle = \frac{1}{V} \int_{V_f} \psi \, dV \tag{3.2}$$

The phase-averaged velocity is known as the superficial velocity or Darcy velocity.

$$\langle \boldsymbol{u} \rangle = \boldsymbol{u_D} \tag{3.3}$$

At low Reynolds number ($Re \ < 1$) for a steady state flow, the pressure drop is proportional to the Darcy velocity as expressed by Darcy's law:

$$-\nabla p = \frac{\mu}{K} \boldsymbol{u_D} \tag{3.4}$$

where K is the permeability. At higher Reynolds numbers for a steady state flow, the pressure drop is proportional to the velocity squared. An inertial drag term can be added to Darcy's

law to represent these effects. A derivation for this is presented in Hsu and Cheng [67]:

$$-\nabla p = \frac{\mu}{K}\boldsymbol{u_D} + \frac{C_F}{\sqrt{K}}\rho \boldsymbol{u_D}^2 \tag{3.5}$$

where C_F is the quadratic drag or Forchheimer coefficient. These equations are useful for modelling the steady state behavior at the macroscopic scale, but this approximation loses many details of the flow. Boundary layer effects, inertia effects and unsteady flow behavior are some characteristics that are not fully represented. This is why there has been continuous work to derive the equivalent macroscopic momentum equation for a porous medium [68, 69]. It is useful to have a set of governing equations that can represent both the open region and the porous region. The derivation process starts by applying averaging theorems [70] to the microscopic momentum equations:

$$\left\langle \frac{\partial \psi}{\partial t} \right\rangle = \frac{\partial \langle \psi \rangle}{\partial t} - \frac{1}{V}\int_{A_f} \psi \boldsymbol{w} \cdot \boldsymbol{n}\, dA \tag{3.6}$$

$$\langle \nabla \psi \rangle = \nabla \langle \psi \rangle + \frac{1}{V}\int_{A_f} \psi \boldsymbol{n}\, dA \tag{3.7}$$

$$\langle \nabla \cdot \psi \rangle = \nabla \cdot \langle \psi \rangle + \frac{1}{V}\int_{A_f} \psi \cdot \boldsymbol{n}\, dA, \tag{3.8}$$

where $\boldsymbol{n}$ represents the unit normal vector of the fluid and $\boldsymbol{w}$ represents the velocity of the region A_f. Applying these theorems to Equation (3.1) gives the following equation

$$\rho\left(\frac{\partial \langle \boldsymbol{u} \rangle}{\partial t} + \nabla \cdot (\langle \boldsymbol{u}\boldsymbol{u} \rangle) + \frac{1}{V}\int_{A_f} \boldsymbol{u}(\boldsymbol{u}-\boldsymbol{w}) \cdot \boldsymbol{n}\, dA\right) = -\nabla \langle p \rangle + \mu \nabla^2 \langle \boldsymbol{u} \rangle + \frac{1}{V}\int_{A_f} (-p\boldsymbol{I} + \mu \nabla \boldsymbol{u}) \cdot \boldsymbol{n}\, dA \tag{3.9}$$

On the left-hand side $\boldsymbol{u} = \boldsymbol{w}$, so this term will be zero. On the right-hand side, the third term is the drag force caused by the relative motion between the fluid and the solid. The derivation of the drag force leads to the Darcy-Forchheimer model, as shown in Hsu and Cheng [67]. From here, the next stage of the derivation process is to write the momentum equation in terms of the intrinsic phase-averaged velocity. The definition of an intrinsic phase-averaged quantity is given by

$$\langle \psi \rangle^f = \frac{1}{V_f}\int_{V_f} \psi\, dV \tag{3.10}$$

. This allows the phase-averaged velocity to be related to the intrinsic phase-averaged velocity:

$$\langle \boldsymbol{u} \rangle = \epsilon \langle \boldsymbol{u} \rangle^f \tag{3.11}$$

$$= \boldsymbol{u_D} \tag{3.12}$$

where the porosity $\epsilon = \frac{V_f}{V}$. The convective term in Equation (3.9), which contains the average of a product, $\langle \boldsymbol{uu} \rangle$ can now changed to the product of averages. The approximation given in [70] is shown below:

$$\langle \boldsymbol{uu} \rangle = \epsilon \langle \boldsymbol{uu} \rangle^f \tag{3.13}$$

$$= \epsilon \langle \boldsymbol{u} \rangle^f \langle \boldsymbol{u} \rangle^f + \epsilon \langle \boldsymbol{u}^{'} \boldsymbol{u}^{'} \rangle^f \tag{3.14}$$

As $\epsilon \to 0$ or $\epsilon \to 1$, the term $\epsilon \langle \boldsymbol{u}^{'} \boldsymbol{u}^{'} \rangle^f$ can be ignored. Following this, the derived macroscopic momentum equation is given below:

$$\rho \left(\frac{\partial \left(\epsilon \langle \boldsymbol{u} \rangle^f \right)}{\partial t} + \nabla \cdot \left(\epsilon \langle \boldsymbol{u} \rangle^f \langle \boldsymbol{u} \rangle^f \right) \right) = -\epsilon \nabla \langle p \rangle^f + \mu \nabla^2 \epsilon \langle \boldsymbol{u} \rangle^f + \boldsymbol{F} \tag{3.15}$$

where $\boldsymbol{F}$ is the Darcy-Forchheimer model $\epsilon \left[\frac{\mu}{K} \langle \boldsymbol{u} \rangle + \rho \frac{C_F}{\sqrt{K}} \langle \boldsymbol{u} \rangle | \langle \boldsymbol{u} \rangle | \right]$. This form of the macroscopic momentum equation has been used in many studies [62, 64, 66, 70]. It can also be rewritten in terms of the Darcy velocity (u_D) and dividing by ϵ [67, 71, 72].

$$\rho \left(\frac{\partial \boldsymbol{u_D}}{\partial t} + \frac{1}{\epsilon} \nabla \cdot (\boldsymbol{u_D} \boldsymbol{u_D}) \right) = -\nabla \langle p \rangle + \mu \nabla^2 \boldsymbol{u_D} + \boldsymbol{F} \tag{3.16}$$

A different form of the macroscopic momentum equation can be derived using a different approximation for the convective terms. In this method, the average of a product of two quantities $\langle \boldsymbol{uu} \rangle$ is replaced by the product of two average quantities:

$$\langle \boldsymbol{uu} \rangle = \langle \boldsymbol{u} \rangle \langle \boldsymbol{u} \rangle = \epsilon \langle \boldsymbol{u} \rangle^f \epsilon \langle \boldsymbol{u} \rangle^f \tag{3.17}$$

The momentum equation using this derivation is presented by Kaviany [73]:

$$\rho\left(\frac{\partial\left(\epsilon\langle\boldsymbol{u}\rangle^f\right)}{\partial t}+\nabla\cdot\left(\epsilon\langle\boldsymbol{u}\rangle^f\epsilon\langle\boldsymbol{u}\rangle^f\right)\right)=-\epsilon\nabla\langle p\rangle^f+\mu\nabla^2\epsilon\langle\boldsymbol{u}\rangle^f+\boldsymbol{F} \tag{3.18}$$

The subtle difference in the convection term can have a large influence on the numerical solution at discontinuities. This is shown in the results section of this chapter. In this study, Equation (3.18) and the volume averaged continuity equation form the set of governing equations:

$$\nabla\cdot(\epsilon\boldsymbol{u}) = 0 \tag{3.19}$$

$$\rho\left(\frac{\partial\left(\epsilon\langle\boldsymbol{u}\rangle^f\right)}{\partial t}+\nabla\cdot\left(\epsilon\langle\boldsymbol{u}\rangle^f\epsilon\langle\boldsymbol{u}\rangle^f\right)\right) = -\epsilon\nabla\langle p\rangle^f+\mu\nabla^2\epsilon\langle\boldsymbol{u}\rangle^f+\boldsymbol{F}. \tag{3.20}$$

3.3 Continuity equation and source term discretization

In pressure-based solvers, the pressure in the momentum equation is treated implicitly. The momentum equation is then substituted into the continuity equation to form a pressure Poisson equation. Alternative methods are possible: one example is to substitute the momentum equation into the energy equation [74] which has been used for simulating high Mach-number flows. Nevertheless, the former method has been used here. The continuity equation at time level $n+1$ is given by

$$\frac{\partial\left(\epsilon u\right)}{\partial x}_{e}=0 \tag{3.21}$$

where the face-centered porosity and velocity are interpolated using the cell-centered values. A 1D analysis of this interpolation shows how these properties can be derived:

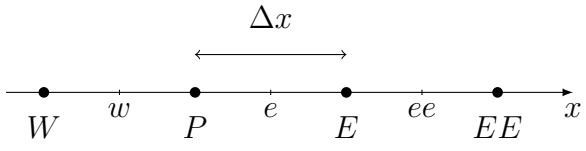


Figure 3.1: Node locations in the x-direction

$$\epsilon_e = \frac{\epsilon_P + \epsilon_E}{2} \tag{3.22}$$

$$u_e = \frac{1}{a_e}\frac{(a_P u_P + a_E u_E)}{2} \tag{3.23}$$

Where a_e is the diagonal coefficient of u_e:

$$a_e = \frac{(a_P + a_E)}{2}. \tag{3.24}$$

The interpolation shown above can cause pressure velocity decoupling and a checkerboarding pressure solution. This is caused by the cell-centered pressure gradients from u_P and u_E that have non-overlapping stencils, leading to the checkerboarding effect. Instead, the Rhie-Chow interpolation replaces these cell centered pressure gradients with a face-centered pressure gradient. Using this and including porosity, the face-centered velocity can be redefined:

$$u_e = \frac{1}{a_e}\left(\frac{(a_P u_P + a_E u_E)}{2} - \left[\epsilon_e \frac{\partial p}{\partial x}_e - \frac{1}{2}\left(\epsilon_P \frac{\partial p}{\partial x}_P + \epsilon_E \frac{\partial p}{\partial x}_E\right)\right]\right). \tag{3.25}$$

3.3.1 Generalized discretization of source terms

Source terms in the momentum equation can come from temperature effects in buoyancy driven flows, body forces due to gravity or electromagnetic fields, inter-facial forces in multi-phase flows and drag terms in porous media flows. These source terms can have a significant effect on the pressure, which can cause nonphysical oscillations at discontinuities. To rectify this, previous studies have shown that the source term discretization and the pressure gradient in the momentum equation need to be equivalent to each other [62, 63]. The source term discretization was then formulated using the source terms in the continuity equation. This is because the pressure gradient in the momentum equation is obtained from the continuity equation. Therefore, the source terms in the momentum equation need to be formulated using the interpolated face-centered source terms from the continuity equation. This can be demonstrated by analysing the porous media drag terms in 1-dimension. The 1D macroscopic momentum equation with

source terms is given below.

$$\rho\left(\frac{\partial\left(\epsilon u\right)}{\partial t}_{E}+\epsilon_E u_E\frac{\partial\left(\epsilon u\right)}{\partial x}_{E}\right)=-\epsilon_E\frac{\partial p}{\partial x}_{E}+\mu\frac{\partial^2\left(\epsilon u\right)}{\partial x^2}_{E}+\epsilon_E S_E. \tag{3.26}$$

where ϵ_E, S_E, and u_E is the porosity, source term and velocity at point E in Figure 3.1. Equation (3.26) has then been simplified to an inviscid steady state equation where the porosity is 1. For these particular cases, the pressure gradient should be equivalent to the drag forces in the source terms:

$$\frac{\partial p}{\partial x}_{E}=S_E u_E. \tag{3.27}$$

The analysis presented here was previously covered by Bartholomew *et al.* [63] and Zhang *et al.* [62] for the discretization of explicit source terms:

$$\bar{S_E} = \frac{1}{2}\left(\frac{S_P u_P+S_E u_E}{2}+\frac{S_E u_E+S_{EE}u_{EE}}{2}\right) \tag{3.28}$$

$$\bar{S}_e = \frac{S_P u_P+S_E u_E}{2} \tag{3.29}$$

where $\bar{S_E}$ is the new cell-centered source term and and $\bar{S}_e$ is the face-centered source term. The face velocity u_e in the continuity equation is now given by

$$u_e=\frac{1}{a_e}\left(\frac{\left(a_P u_P+a_E u_E\right)}{2}-\left[\frac{\partial p}{\partial x}_{e}-\frac{1}{2}\left(\frac{\partial p}{\partial x}_{P}+\frac{\partial p}{\partial x}_{E}\right)+\bar{S}_e-\frac{1}{2}\left(\bar{S_P}+\bar{S_E}\right)\right]\right) \tag{3.30}$$

where a_e is defined in Equation (3.24). This replaces the cell-centered source terms with the face-centered source term. Equation (3.26) can now be rewritten using the reformulated cell-centered source term.

$$\rho\left(\frac{\partial u}{\partial t}_{E}+u_E\frac{\partial u}{\partial x}_{E}\right)=-\frac{\partial p}{\partial x}_{E}+\mu\frac{\partial^2 u}{\partial x^2}_{E}+\bar{S_E} \tag{3.31}$$

In this section, the Darcy-Forchheimer model has been used, where $S=\left[\frac{\mu\epsilon}{K}\boldsymbol{u}+\rho\frac{C_D\epsilon^2}{\sqrt{K}}\boldsymbol{u}|\boldsymbol{u}|\right]$. However, the reformulated source term $\bar{S_E}$ could be derived for any type of force used in the momentum equation.

3.3.2 Discretization of explicit drag terms with variable porosity

The discretization shown in this section extends the method from the previous section to include variations in porosity for the explicit drag terms. This still follows the same approach as the previous source term method: reformulate the cell-centered source terms in the momentum equation using the face-centered source terms from the continuity equation.

$$\bar{S_E}^{exp} = \frac{1}{2}\left(\frac{S_P + S_E}{2} + \frac{S_E + S_{EE}}{2}\right) \tag{3.32}$$

$$\bar{S_e}^{exp} = \frac{S_P + S_E}{2} \tag{3.33}$$

where $\bar{S_E}^{exp}$ is the new cell-centered source term and and $\bar{S_e}^{exp}$ is the face-centered source term. The face velocity u_e is now given by

$$u_e = \frac{1}{a_e}\left(\frac{(a_P u_P + a_E u_E)}{2} - \left[\epsilon_e \frac{\partial p}{\partial x}_e - \frac{1}{2}\left(\epsilon_P \frac{\partial p}{\partial x}_P + \epsilon_E \frac{\partial p}{\partial x}_E\right) + \epsilon_e \bar{S_e}^{exp} - \frac{1}{2}\left(\epsilon_P \bar{S_P}^{exp} + \epsilon_E \bar{S_E}^{exp}\right)\right]\right) \tag{3.34}$$

where a_e is defined in Equation (3.24).

$$\rho\left(\frac{\partial(\epsilon u)}{\partial t}_E + \epsilon_E u_E \frac{\partial(\epsilon u)}{\partial x}_E\right) = -\epsilon_E \frac{\partial p}{\partial x}_E + \mu \frac{\partial^2(\epsilon u)}{\partial x^2}_E + \epsilon_E \bar{S_E}^{exp}. \tag{3.35}$$

3.3.3 Discretization of implicit drag terms with variable porosity

Extremely large drag terms will still cause numerical instability, even with an accurate explicit formulation. In these circumstances, an implicit formulation of drag terms is needed to make the numerical model more stable. The 1D macroscopic momentum equation is given below.

$$\rho\left(\frac{\partial \epsilon_E u_E}{\partial t} + \epsilon_E u_E \frac{\partial \epsilon_E u_E}{\partial x}\right) = -\epsilon \frac{\partial p}{\partial x} + \mu \frac{\partial^2 \epsilon_E u_E}{\partial x^2} + \epsilon_E S_E^{imp} u_E. \tag{3.36}$$

In this equation, there is a slight difference in how the source term is represented compared to the explicit methods. The velocity vector in the Darcy-Forchheimer model needs to be treated

implicitly. This means the source term S^{imp} is now defined as $\left[\frac{\mu\epsilon}{K} + \rho\frac{C_D\epsilon^2}{\sqrt{K}}|\boldsymbol{u}|\right]$, with a factor of $\boldsymbol{u}^{n+1}$ removed. This velocity component $\boldsymbol{u}^{n+1}$ is solved implicitly as a diagonal coefficient. The formulation of this new implicit source term still uses the same approach as the explicit formulation. However, there is also a slight difference in the porosity interpolation:

$$\bar{S_E}^{imp} = \frac{1}{2}\left(\frac{S_P/\epsilon_P + S_E/\epsilon_E}{2} + \frac{S_E/\epsilon_E + S_{EE}/\epsilon_{EE}}{2}\right) \tag{3.37}$$

$$\bar{S_e}^{imp} = \frac{S_P/\epsilon_P + S_E/\epsilon_E}{2} \tag{3.38}$$

where $\bar{S_E}^{imp}$ is the cell-centered implicit source term and and $\bar{S_e}^{imp}$ is the face-centered implicit source term. The face velocity u_e and diagonal coefficient a_e are now given by

$$u_e = \frac{1}{a_e}\left(\frac{(a_P u_P + a_E u_E)}{2} - \left[\epsilon_e \frac{\partial p}{\partial x}_e - \frac{1}{2}\left(\epsilon_P \frac{\partial p}{\partial x}_P + \epsilon_E \frac{\partial p}{\partial x}_E\right)\right]\right) \tag{3.39}$$

$$a_e = \frac{(a_P + a_E)}{2} + \epsilon_e^2 \bar{S_e}^{imp} - \frac{1}{2}\left(\epsilon_P^2 \bar{S_P}^{imp} + \epsilon_E^2 \bar{S_E}^{imp}\right). \tag{3.40}$$

The macroscopic momentum equation with the cell-centered implicit source term is given below:

$$\rho\left(\frac{\partial \epsilon_E u_E}{\partial t} + \epsilon_E u_E \frac{\partial \epsilon_E u_E}{\partial x}\right) = -\epsilon\frac{\partial p}{\partial x} + \mu\frac{\partial^2 \epsilon_E u_E}{\partial x^2} + \epsilon_E^2 \bar{S_E}^{imp} u_E \tag{3.41}$$

3.4 Solution procedure

3.4.1 Time integration and spatial discretization

A first order Implicit-Explicit (IMEX) scheme has been used for the time integration. The non-linear advection terms are treated explicitly while the linear viscous terms are implicit. This yields a time step restriction that can be approximated by the Courant–Friedrichs–Lewy condition.

$$\Delta t\left(\sum_{i=1}^{n} \frac{u_i}{\Delta x_i}\right) \leqslant C_{max} \tag{3.42}$$

Δt is the time step size, Δx is the interval, u is the magnitude of the velocity, n is the number of dimensions and C_{max} is the Courant number. The value of the Courant number in the CFL condition depends on the spatial discretization and the time integration method. In this numerical solver, the advection terms have been discretized using either upwind or central difference schemes. These discretization schemes can be represented symbolically using Devito's DSL for finite difference operators that enable the user to specify the spatial discretization order.

3.4.2 Coupled pressure based solver

The system of equations is solved using a coupled solver for pressure and velocity. There are several other pressure based numerical procedures for solving incompressible flow which normally involve operator splitting such as SIMPLE, PISO and projection methods. These numerical procedures could also be used to solve the discretized equations shown in the previous section. Instead, a simple approach is used here which solves the entire system in one step so that there is no operator splitting error. Also, if the advection term is treated implicitly, it can be extended to solve compressible flow problems using a fully coupled implicit solver [75]. The discretized momentum equation can be written as

$$\rho\left(\epsilon\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^{n}}{\Delta t}+\nabla\cdot(\epsilon\boldsymbol{u}^{n}\otimes\epsilon\boldsymbol{u}^{n})\right)=-\epsilon\nabla P^{n+1}+\mu\nabla^{2}\epsilon\boldsymbol{u}^{n+1}+\epsilon\bar{S}^{exp}+\epsilon^{2}\bar{S}^{imp}\boldsymbol{u}^{n+1} \tag{3.43}$$

While the solution procedure is shown below

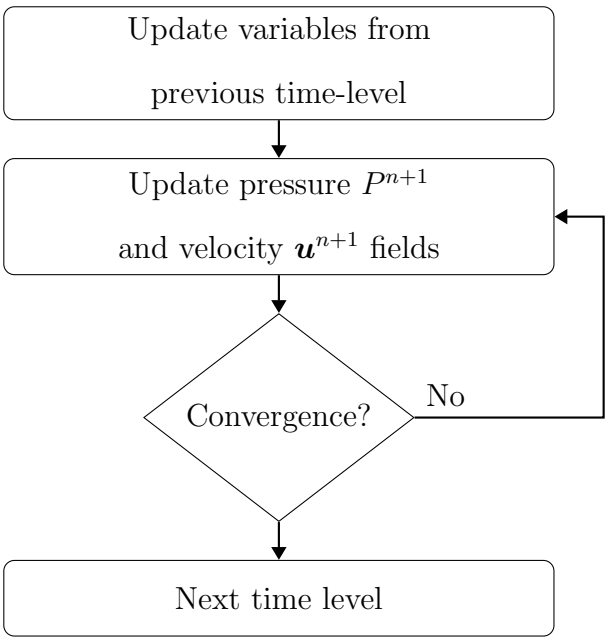


Figure 3.2: Flowchart of the solution strategy where P^{n+1} and $\boldsymbol{u}^{n+1}$ are solved simultaneously using Equations (3.21) and (3.43).

3.5 Numerical test cases

The proposed discretization schemes have been verified against a number of test cases. This covers both viscous and inviscid flow to assess the robustness and accuracy of this method. A structured Cartesian grid has been used for all the numerical simulations. The results from these steady state and transient incompressible flow problems have been compared with exact solutions or other studies.

3.5.1 Inviscid porous plug test case

The first test case is a numerical experiment which simulates inviscid incompressible flow through a porous region [62]. This is an idealized problem in which the porous medium exerts no drag resistance. The domain is shown in Figure 3.3, for a 2D channel of length 6H and width H. At the inlet where x= 0, a uniform velocity of 50 m/s is specified and the density is 1.1614 kg/m^3. In the central region, which is shaded grey, the porosity is 0.4 while the rest of

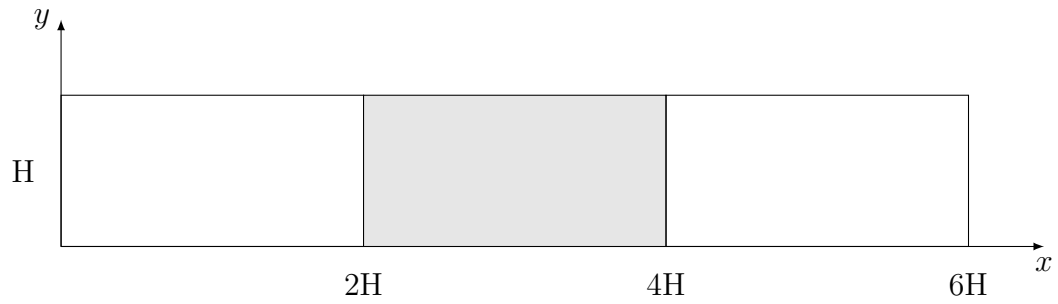


Figure 3.3: Computed domain for the porous plug test cases. The white sections are the open fluid regions while the central grey section is the porous region.

the domain has a porosity of 1. Slip wall conditions are used for the left and right wall boundaries, while a pressure boundary condition is used for the outlet. In this study, this numerical experiment has been used to compare Equations (3.15), (3.16) and (3.18) with the drag force $\boldsymbol{F}$ set to zero. This will show how changing the convective term influences the numerical results at porous-fluid interfaces.

Zhang *et al.* [62] explained that there are two key contributions that affect the pressure gradient: the dynamic effects from the fluid flow and non-dynamic effects from the body forces. The dynamics effects are caused by changes in the velocity, while the non-dynamics effects are caused by the drag terms or other body forces. In this test case, there are no drag forces or no-slip boundary conditions, so the non-dynamic effects are zero. The dynamic effects depend on the momentum transfer at the porous-fluid interface. If there is no momentum loss at the porous-fluid interface, the dynamic effects on the pressure gradient should also be zero. Instead, the results in Figure 13 from Zhang *et al.* [62] showed large oscillations at the porous-fluid interfaces. The explanation given was that the Rhie-Chow interpolation error becomes significant at large discontinuities in velocity. However, the results in this study show that the convective term has a larger impact than the Rhie-Chow interpolation error.

In the present study, a uniform mesh of 76 cells in the x direction and 6 cells in the y direction were used for the computed domain. The Courant number was 0.1, while the convective term was discretized using an upwind scheme. The results are shown in Figures 3.4 and 3.5, where the model using Equation (3.15) mirrors the results obtained by Zhang *et al.* [62]. In contrast, the models using Equations (3.16) and (3.18) have a negligible pressure drop and no numerical

oscillations at the discontinuities. The velocity also has no numerical oscillations and gives the expected analytical value. This can be tested further by reducing ϵ to 0.04 in the porous region. The results are shown in Figures 3.6 and 3.7, where only Equation (3.18) was able to generate stable results. From these results, Equation (3.18) is most suitable for simulating sharp discontinuities at porous-fluid interfaces, while only generating negligible oscillations. This is relevant for Darcy flows ($Re < 1$) where the inertial effects from the convective terms have a negligible impact on the pressure gradient. Equation (3.18) replicates the expected analytical solutions while including the convective terms in the momentum equation.

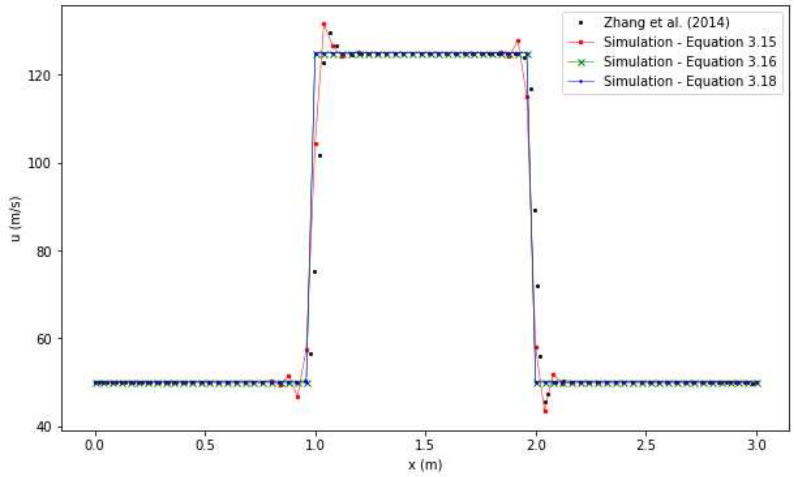


Figure 3.4: The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$

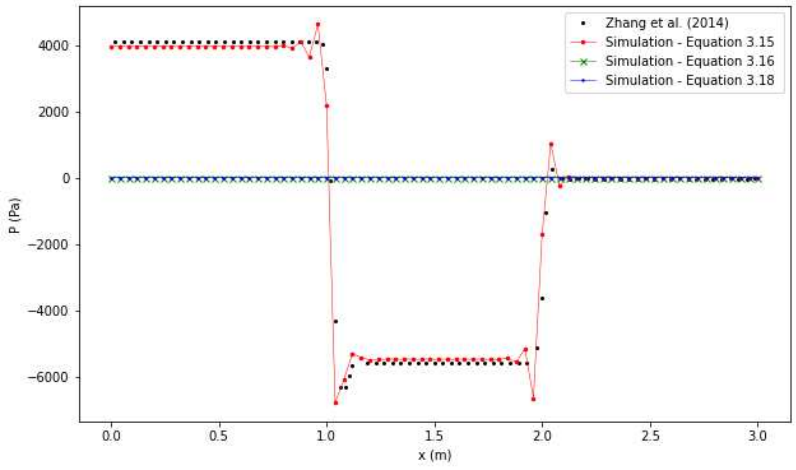


Figure 3.5: The pressure plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.4$

The next test cases are for verifying the flow through a porous media with drag forces. A drag force is specified within the porous region by setting the Forchheimer coefficient C_F to 0.55

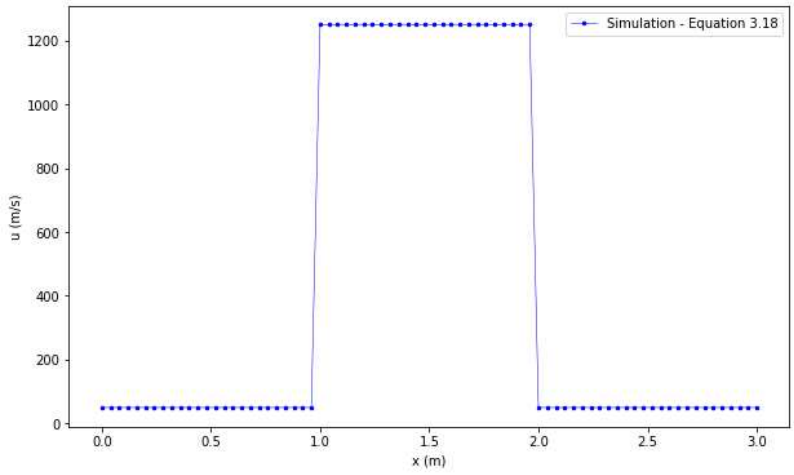


Figure 3.6: The velocity plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.04$

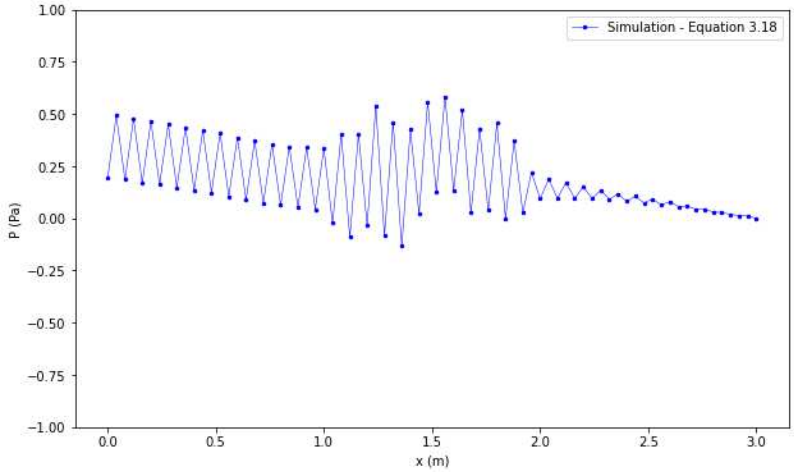


Figure 3.7: The pressure plot for an inviscid flow with no drag forces in the porous region where $\epsilon = 0.04$

and varying the Darcy number, where $D_a = \frac{K}{H^2}$. Here the velocity is 1 m/s, and the density is 1.1614 kg/m^3. Both the implicit and explicit methods have been used to discretize the drag terms in these test cases. The results using Darcy numbers of 10^{-5} and 10^0 are shown in Figures 3.8, 3.9, 3.10 and 3.11. There are no numerical oscillations at the porous-fluid discontinuities in both the pressure and velocity field. This is zoomed in for clarity in Figures 3.12 and 3.13. An analytical solution of the pressure gradient can be calculated by using the Darcy-Forchheimer model with no viscous drag.

$$\epsilon \frac{\Delta P}{\Delta x} = \rho \frac{\epsilon^2 C_D}{\sqrt{K}} |u| u \tag{3.44}$$

Table 3.1 shows the analytical pressure gradient and the numerical results. There is a small

error of about $1.5 \times 10^{-3}\%$ when $Da = 10^{-5}$ and a 0.25% error when $Da = 10^{0}$.

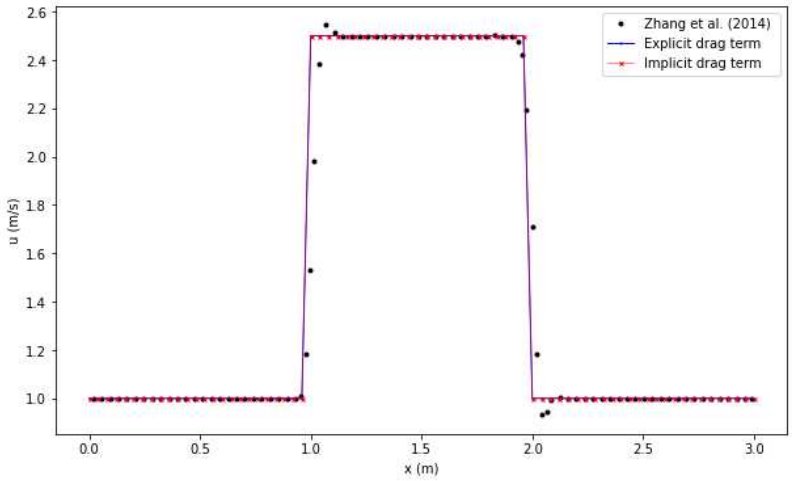


Figure 3.8: Inviscid flow through a porous plug where $D_a = 10^{-5}$

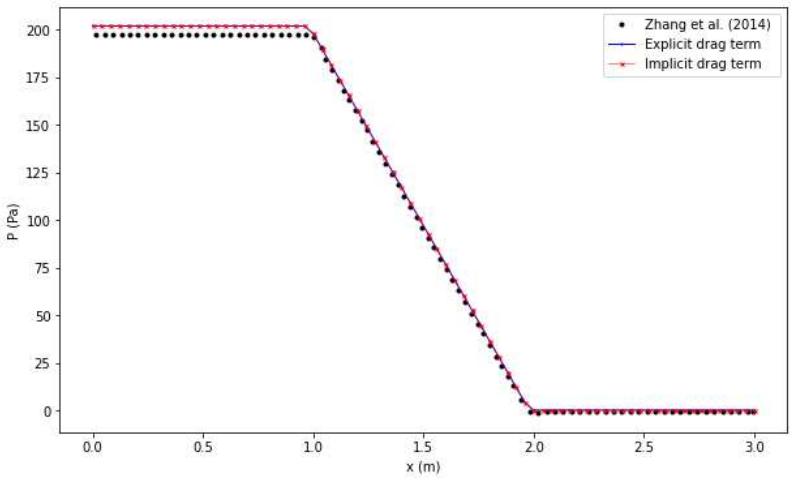


Figure 3.9: Inviscid flow through a porous plug where $D_a = 10^{-5}$

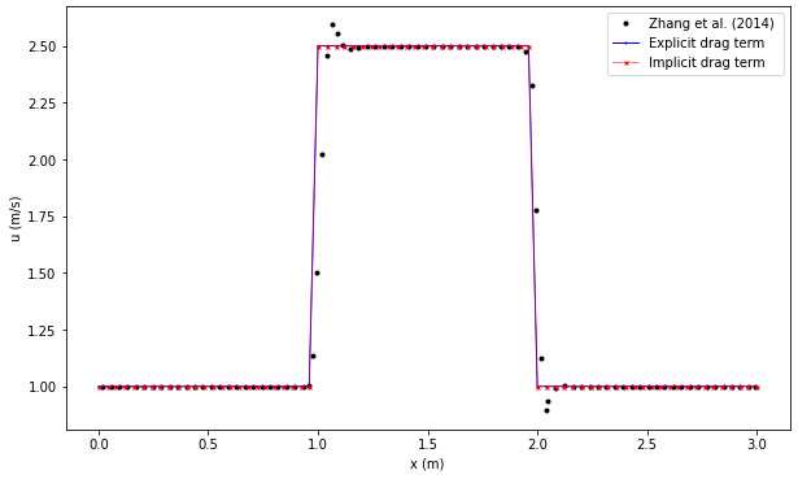


Figure 3.10: Inviscid flow through a porous plug where $D_a = 10^0$

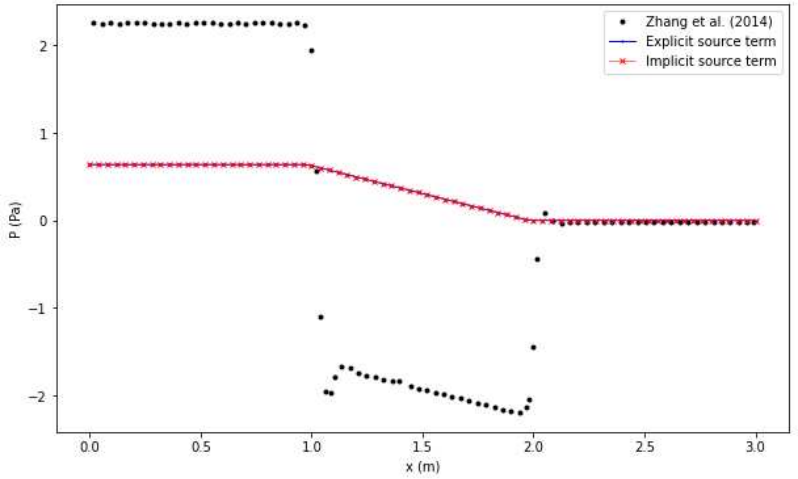


Figure 3.11: Inviscid flow through a porous plug where $D_a = 10^0$

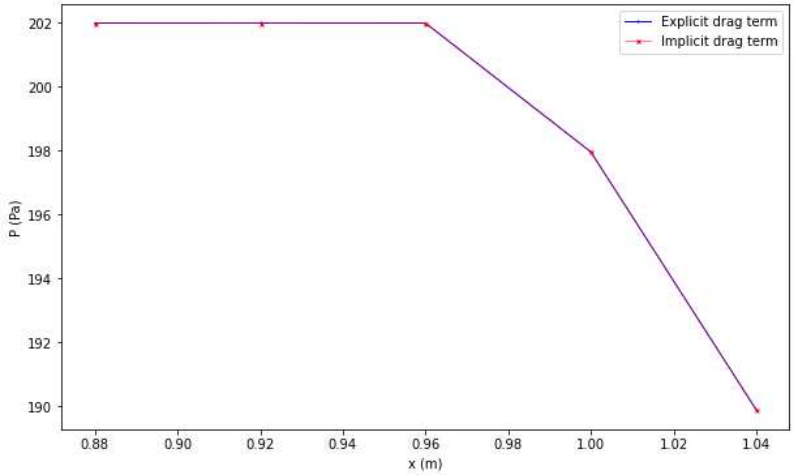


Figure 3.12: A zoomed in view of the pressure drop where $D_a = 10^{-5}$

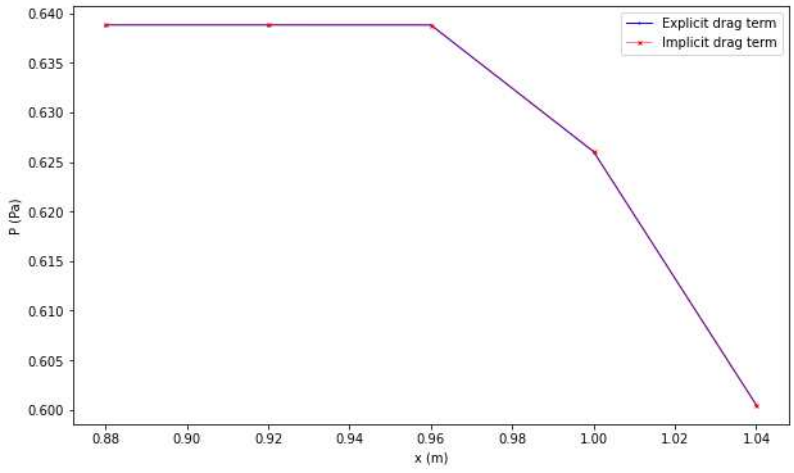


Figure 3.13: A zoomed in view of the pressure drop where $D_a = 10^0$ (right)

Table 3.1: Numerical and analytical solutions for the invisicid porous plug test case

Darcy number	**Analytical solution**	**Numerical solution**
10^{-5}	201.997 Pa	Explicit = 202.000 Pa
		Implicit = 201.996 Pa
10^0	0.6377 Pa	Explicit = 0.638864 Pa
		Implicit = 0.638908 Pa

3.5.2 Viscous porous plug test cases

This porous plug test case taken from Costa *et al.* [76] simulates the viscous flow through a porous region. Similarly to the previous test cases, this involves a 2D channel with a porous region in the center. At the inlet, the flow is fully developed with a parabolic velocity profile. The centerline velocity is 1.5 m/s and the mean superficial velocity U is 1 m/s. The density is 1 kg/m^3, the viscosity is 1 Pas and the Reynolds number is 1. The porosity and Darcy numbers are 0.7 and 10^{-3} respectively. No-slip wall boundary conditions are applied at the left and right walls. The computed domain consists of 161 cells in the x direction and 41 cells in the y direction. The results are shown in Figures 3.15 and 3.14 where there are no oscillations at porous-fluid boundaries. However, the velocity and pressure gradient in the porous region Costa *et al.* [76] are underpredicted for using the initial mesh. This is consistent with other numerical studies [64, 66] compared to the reference mesh of 204 800 cells.

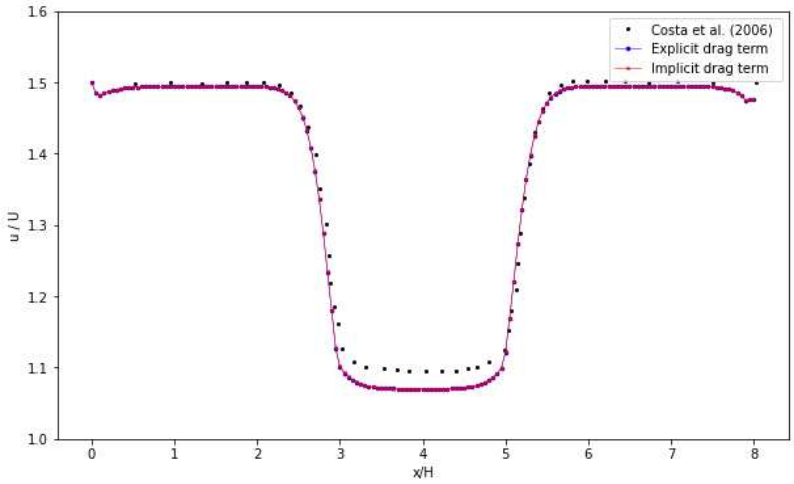


Figure 3.14: Viscous flow through a porous plug where $D_a = 10^{-3}$

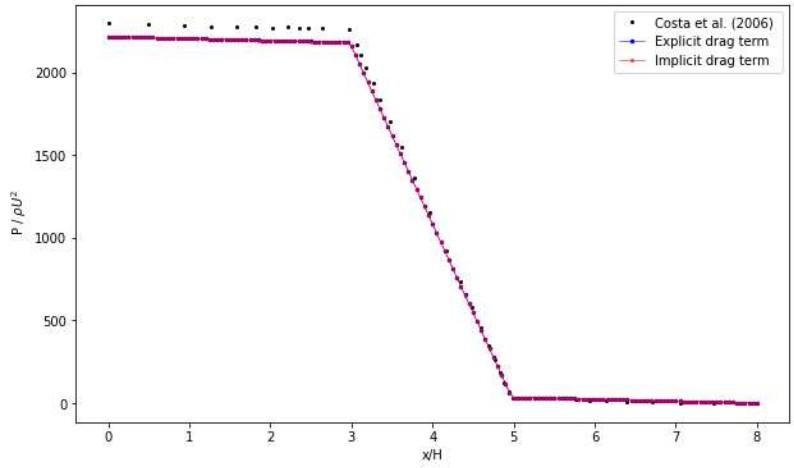


Figure 3.15: Viscous flow through a porous plug where $D_a = 10^{-3}$

3.5.3 Joseph Beavers test case

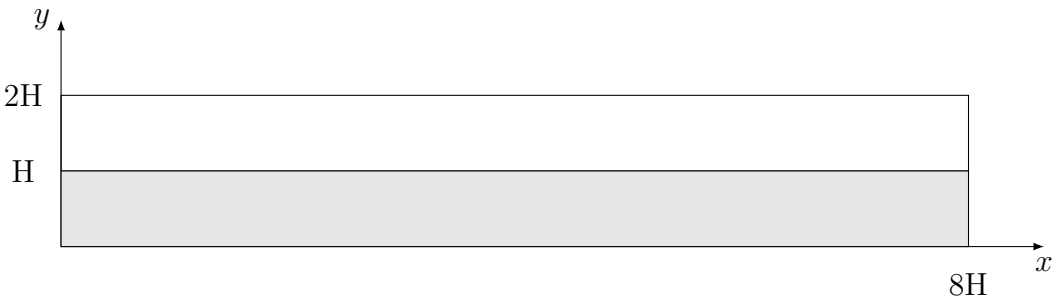


Figure 3.16: Computed domain for the Joseph-Beavers test case

The Joseph Beavers test case is also taken from Costa *et al.* [76], which simulates fluid flow parallel to a porous region. This domain is shown in Figure 3.17 where the porous region is

shaded grey while the open region is in white. At the inlet of the open region, the Reynolds number is 1 and the velocity is uniform. In the porous region, the porosity is 0.7 and the Darcy number is 10^{-2}. No slip wall boundary conditions are applied on the left and right walls. A uniform mesh with 81 cells in the x direction and 61 cells in the y direction have been used. The results are shown in Figure 3.17 where there is good agreement with the reference solution.

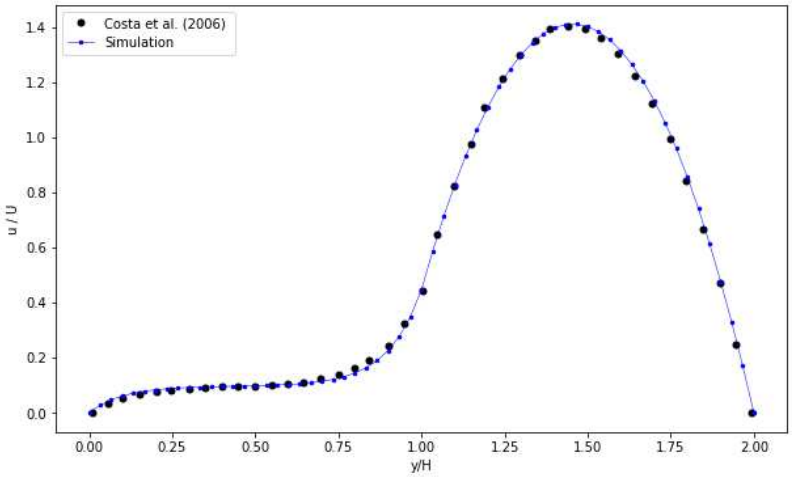
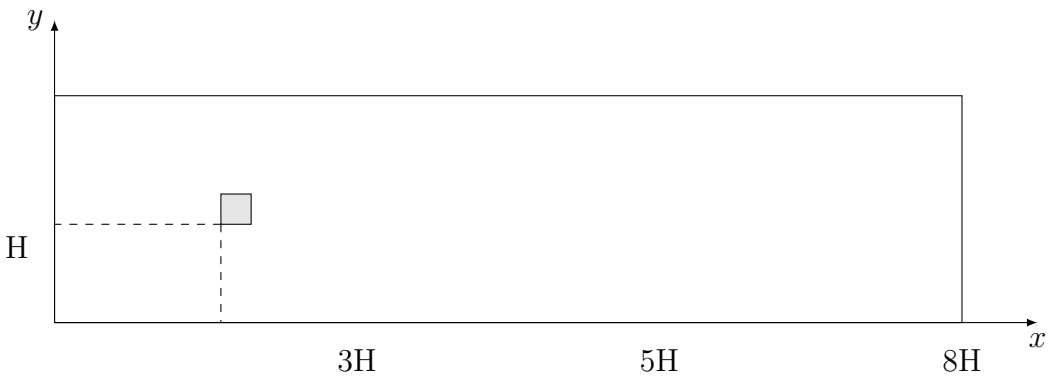


Figure 3.17: Velocity profile for the flow parallel to a porous region

3.5.4 Unsteady flow past a square block



Fluid flow past a square cylinder at moderate Reynolds numbers is a transient problem that produces vortex shedding behind the cylinder. The domain used is shown above and is taken from Ray and Kumar [77]. The inlet boundary is a uniform velocity while the left and right walls use slip wall boundary conditions. At the outlet, a convective boundary condition is used which reduces the effects of the boundary on the vortices in the inner domain. The Reynolds

number based on the square cylinder's side length is 100. This object is represented as a porous region with a large drag force, where the porosity is 0.7 and the Darcy number is 10^{-15}. For such a large drag force the implicit method has been used. There are 801 cells in the x direction and 276 cells in the y direction. The results in Table 3.2 show good agreement with other studies for the Strouhal Number and drag coefficient C_D of the square block. This shows the numerical scheme is suitable for unsteady flows and extremely large drag forces.

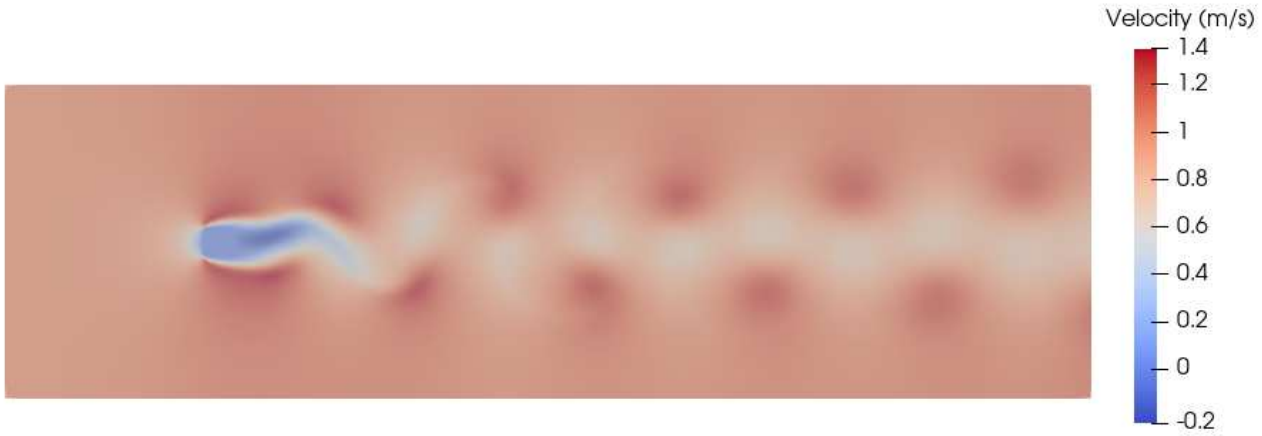


Figure 3.18: Vortex shedding at Re = 100

Table 3.2: Strouhal number comparison between studies at Re 100

Reference	**Strouhal Number**	**C**
Davis and Moore [78]	0.148-0.153	1.64
Jue [79]	0.154	1.60
Ray and Kumar [77]	0.148	1.61
Present study	0.151	1.62

3.6 Conclusions

In this chapter, a numerical scheme has been presented for modelling the volume-averaged Navier Stokes equations in a porous medium. The main aim is to demonstrate how to model sharp discontinuities at porous-fluid interfaces without generating numerical oscillations. The first numerical experiment compared different derivations of the volume-averaged momentum equation. This showed that the convective term can have a bigger influence on the numerical errors than the Rhie-Chow interpolation error. In these results, the semi-heuristic momentum

equation from Kaviany [73] produced the least oscillations. This model would be most appropriate for creeping and moderate Reynolds number flows where the convective term has a negligible impact on the pressure gradient.

The next test cases verified the discretization of the drag forces. Two novel formulations for discretizing have been shown: one explicit method and one implicit method. Both of these methods use a unified formulation across the entire porous-fluid domain. No porosity smoothing or artificial diffusion is needed at porous-fluid interfaces. Instead, the discretization of the drag terms in the momentum equation is exactly force balanced with the pressure gradient. These methods were verified using test cases for steady state inviscid and viscous flow with low permeabilites in the porous region. The results showed good agreement with the analytical solutions, displaying no oscillations at porous-fluid discontinuities even at low Darcy numbers. Further validation was conducted for the transient vortex shedding past a block, which showed good agreement with the literature.

The advantages of these schemes are that they replicate the robustness of a staggered grid with flexibility of a collocated grid arrangement. A future study could be to use these schemes on unstructured meshes in FV CFD codes. In the following chapter, the drag term discretizations schemes shown here are used in a 3D porous-media CFD solver. The application for this solver is subchannel analysis in nuclear fuel assemblies.

Chapter 4

Validation of single-phase subchannel models

In this chapter, a porous-media CFD code has been validated against experimental thermal hydraulic benchmarks. This code uses the numerical methods developed in the previous chapter within a 3D CFD solver. The solver has been implemented with empirical correlations for the heat transfer, pressure drop and turbulence in a fuel rod bundle. Also, a cylindrical heat equation has been implemented in this code. This represents the radial heat conduction in each fuel rod. The heat equation has been verified against analytical solutions while the subchannel models have been validated against experimental rod bundle tests.

4.1 Introduction

Nuclear thermal hydraulics is an essential discipline for improving the safety and performance of a nuclear reactor. It is used to understand how a reactor operates with different geometric design, operation conditions or during transient scenarios. However, performing experiments for all of these scenarios would be too time-consuming and expensive. This is why computer codes are necessary to complement the experimental thermal hydraulic tests. Subchannel codes are the industry standard for the thermal hydraulic analysis of a reactor core. In these methods, the

core is approximated as a porous media, then empirical correlations in the governing equations are used to represent the momentum and energy transfer between the fluid and the solid. Many subchannel codes, such as the COBRA family of subchannel codes have been validated extensively on a wide range of thermal hydraulic problems. This includes the turbulent mixing in nuclear rod bundles [80], the average fuel pellet temperature, the critical heat flux and the departure from nuclear boiling ratio (DNBR) [81] .

More recently, improvements in computing power has enabled higher fidelity simulations to become practical. CFD codes are now being used more frequently for detailed 3D simulations of the fluid flow. But, it is still too computationally expensive to do detailed 3D simulations of the nuclear reactor core in a reasonable time. Instead, some CFD codes use the porous-media approach to approximate complex geometries. This allows subchannel-scale analysis to be performed for a fuel assembly or the entire reactor core. Examples of these codes include CUPID, GenFOAM, CorTaF and PORFLO. These codes have also been validated against experimental thermal hydraulic benchmarks of light water reactors. The benefit of being able to simulate different scales of analysis is why many porous media codes are being actively developed.

In this work, a porous-media CFD code been implemented with single-phase subchannel thermal hydraulic models. Section 4.2 shows the governing equations and subchannel models for this solver. The subchannel models include correlations for single-phase heat transfer, pressure drop and turbulence that are implemented into the governing equations. The energy equation is coupled to a cylindrical heat equation which represents the heat conduction within each fuel rod. Section 4.3 shows the solution procedure for solving the system of equations. In Section 4.4, the heat equation model has been verified against steady state and transient analytical solutions for radial heat conduction. In Section 4.5 the subchannel models have been validated against the PNL 7x7, WH 14X14 and PNNL 2X6 experimental rod bundle tests. These results have also been compared to other numerical codes: the subchannel code MATRA, the porous-media code CorTaF and the course-grid CFD code SubchannelCFD.

4.2 Governing equations and subchannel T/H models

4.2.1 Governing equations

The governing equations represent the flow of an incompressible fluid through a porous media with heat transfer between the solid and the fluid. These have been discretized in 3D on a structured cartesian grid. Equations (4.1), (4.2) and (4.3) given below are the mass, momentum and energy equations.

$$\nabla \cdot (\epsilon \boldsymbol{u}) = 0 \tag{4.1}$$

$$\rho\left(\frac{\partial \epsilon \boldsymbol{u}}{\partial t} + (\epsilon \boldsymbol{u} \cdot \nabla)\,\boldsymbol{u}\right) = -\epsilon \nabla P + \epsilon \nabla \cdot \tau + \epsilon S_M \tag{4.2}$$

$$\rho C_p\left(\frac{\partial \epsilon T}{\partial t} + \epsilon \boldsymbol{u} \cdot \nabla T\right) = \epsilon \nabla \cdot (k \nabla T) + \epsilon q_E''' \tag{4.3}$$

ρ is the density, ϵ is the porosity, P is the pressure, τ is the viscous stress, S_M is the source term and $\boldsymbol{u}$ is the velocity. In the energy equation, C_p is the heat capacity of the fluid, T is the temperature, k is the thermal conductivity and q_E is the heat transfer source term.

4.2.2 Subchannel T/H models

Pressure drop model

The source term in Equation (4.2) represents the drag resistance from a solid object to the fluid flowing past it. This will cause a pressure drop in the fluid. The general form of this equation is

$$S_M = \frac{f}{D_h}\frac{\rho}{2}\,\boldsymbol{u}|\boldsymbol{u}| + \frac{K''}{\Delta z"}\frac{\rho}{2}\boldsymbol{u_0}|\boldsymbol{u_0}|, \tag{4.4}$$

where f is the friction factor, D_h is the hydraulic diameter, K'' is the form loss coefficient, $\Delta z"$ is the axial length of a solid obstruction and $\boldsymbol{u_0}$ is the inlet velocity. The coefficient

$\frac{f}{D_h}$ in the first term is the frictional drag resistance from the fuel rods. f is derived from experimental pressure loss data as shown in Equation (4.5), then an appropriate correlation is used to relate the friction factor to the Reynolds number. Table 4.1 shows some friction factor formula applicable to smooth pipes. For other pipe geometries or pipes with surface roughness, other formulas are often more appropriate.

$$f = \frac{2\Delta P D_h}{\rho \boldsymbol{u}^2 L} \tag{4.5}$$

where L is the length of the fuel rod.

Table 4.1: Friction factor correlations

Reynolds Number	**Correlation**
$Re \leqslant 2000$	$f = 64Re^{-1}$
$2000 < Re \leqslant 30000$	$f = 0.316Re^{-0.25}$
$Re > 30000$	$f = 0.184Re^{-0.2}$

The form loss coefficient ($K^{''}$) is due to internal blockages such as spacer grids or fuel rod swelling blockages which have a length of $\Delta z^{"}$. These are calculated by first comparing the experimental pressure drop of the flow without a blockage to the the flow with a blockage. Then, the minor head loss formula is rearranged for the loss coefficient. This is shown below

$$K^{''} = \frac{2\Delta P}{\rho \boldsymbol{u_0}^2 \Delta z^{"}} \tag{4.6}$$

Turbulence model

Turbulence between the subchannels can be represented using a turbulent viscosity μ_t [29]. The viscosity in Equation (4.2) can be written as

$$\mu = \mu_l + \mu_t \tag{4.7}$$

where μ_l is the dynamic viscosity of the fluid. This is analogous to the mixing length model where μ_t is also used to model turbulence. In this subchannel model, the characteristic length

is the subchannel gap size:

$$\mu_t = \beta s G \,, \tag{4.8}$$

where β is the mixing coefficient, s is the subchannel gap size and G is the mass flux. Reynolds analogy has then been used to relate the turbulent momentum transfer to the turbulent heat transfer. This approximates the Prandtl number to be 1. The total thermal conductivity can now be expressed in terms of the fluid's intensive property and the turbulent term.

$$k = k_l + k_t \quad \text{where } k_t = \mu_t C_p \,. \tag{4.9}$$

Heat transfer model

The source term in the energy equation represents convective heat transfer from the surface of the fuel rod to the coolant channel:

$$q_E''' = \frac{S}{V_f} q_{conv}'' \tag{4.10}$$

$$q_{conv}'' = h(T_{fo} - T) \tag{4.11}$$

S is the heat transfer area of the fuel, V is the volume of the coolant channel, h is the convective heat transfer coefficient and T_{fo} is the temperature of the fuel surface. The heat transfer coefficient can then be calculated from the Nusselt number, while the Nusselt number can be calculated using the Dittus-Boettler correlation:

$$h = \frac{Nuk}{D_h} \tag{4.12}$$

$$Nu = 0.023 Re^{0.8} Pr^{0.4}. \tag{4.13}$$

This is a simple correlation for fully developed turbulent flow in smooth pipes. But, it is less accurate when there is a large temperature difference between the solid surface and bulk fluid or if the calculation is for rough pipes.

4.2.3 Heat equation for nuclear fuel rod

The heat equation is written in cylindrical coordinates to represent radial heat conduction in a cylindrical fuel rod. This equation is given by

$$\rho_f C_{p,f} \frac{\partial T_f}{\partial t} = k_f \left(\frac{\partial^2 T_f}{\partial r^2} + \frac{1}{r} \frac{\partial T_f}{\partial r} + \frac{\partial^2 T_f}{\partial z^2} \right) + q'''_G \tag{4.14}$$

where $C_{p,f}$ is the fuel heat capacity, T_f is the fuel's temperature, k_f is the fuel's thermal conductivity, r is the radius and q'''_G is the heat generation. A single heat equation is used for both the pellet and the cladding. Then, appropriate boundary conditions are used at the boundary nodes of the different materials to couple the heat transfer. A diagram of a cylindrical fuel rod is shown below

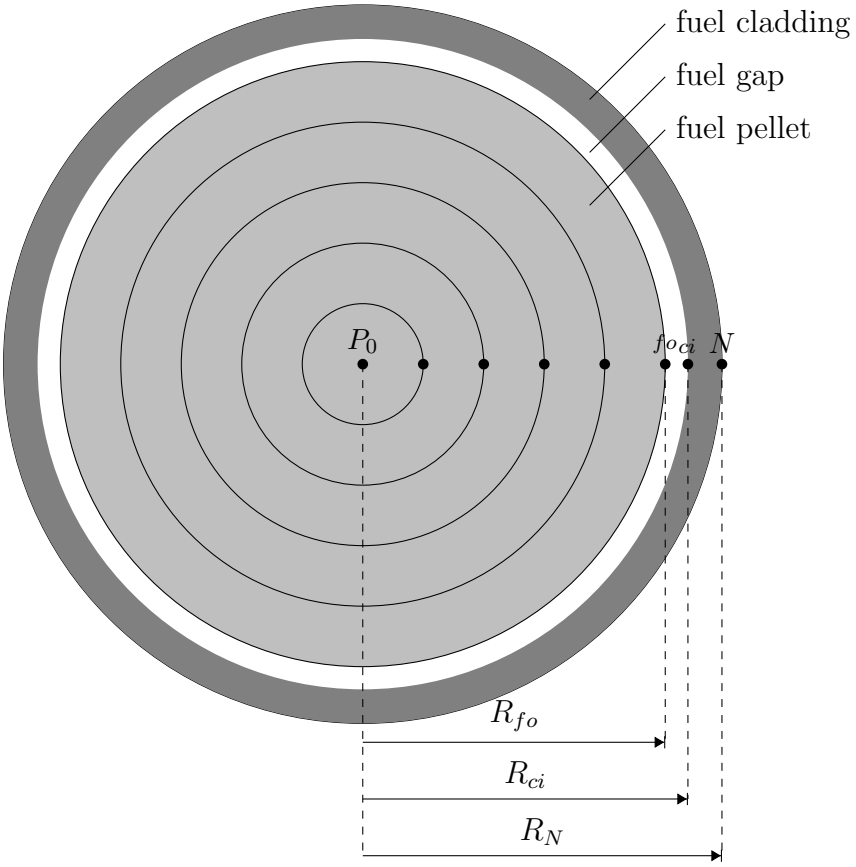


Figure 4.1: Layout of a fuel rod

4.3 Solution procedure

The partial differential equations have been discretized using the finite difference method. The source term in the momentum equation, Equation (4.2) has been discretized using the schemes outlined in Chapter 3 but extended to 3D. These discretized equations are then solved using a pressure based solver. This is shown in Figure 4.2.

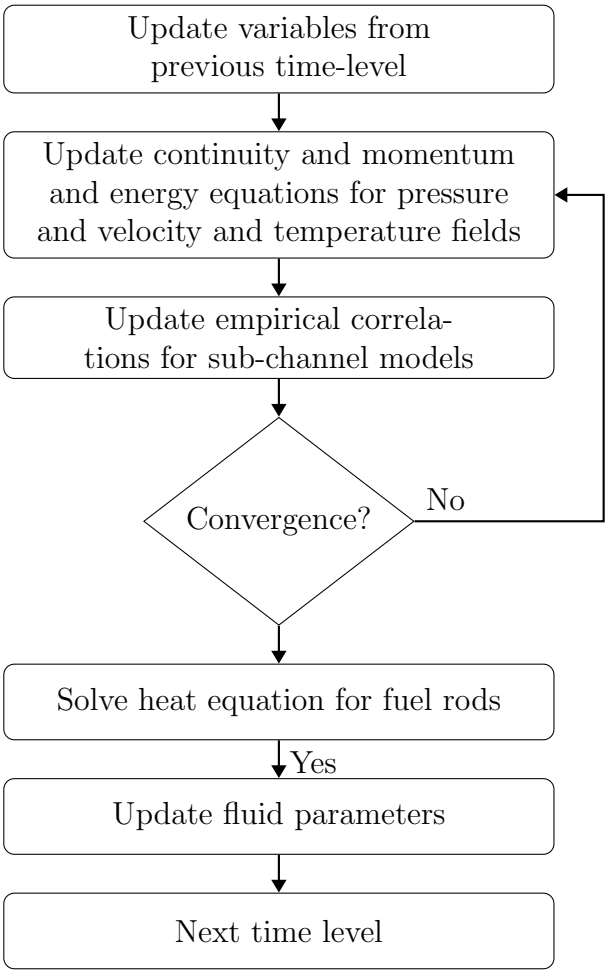


Figure 4.2: Flowchart of the solution strategy

4.4 Verification of heat conduction model

The discretization scheme for Equation (4.14) has been verified by comparing its numerical solutions with analytical solutions. These analytical solutions included key assumptions to simplify the model:

- The fuel rod is an infinitely long cylinder.
- Each material has constant thermal properties.

Boundary conditions at the interface between the different materials were applied in each test case.

4.4.1 Verification of steady state solutions

These steady state test cases are solved without the transient term in Equation (4.14). In the first test case, the fuel rod is represented as a heated cylinder surrounded by a flowing coolant. The cylinder has a uniform heat generation and constant thermal properties, while the coolant has a constant temperature. At the centre of the cylinder, Neumann boundary conditions are used to represent a zero temperature gradient. On the cylinders outer surface, a Robin boundary condition is used for the convective heat flux between the coolant and the solid. These are given below:

$$\left.\frac{\partial T_f}{\partial r}\right|_{r=0} = 0 \tag{4.15}$$

$$-k_f \left.\frac{\partial T_f}{\partial r}\right|_{r=R_N} = h(T_N - T_\infty). \tag{4.16}$$

An analytical solution can be derived by integrating Equation (4.14) with the boundary conditions Equations (4.15) and (4.16). This derivation has been shown in Thatoi *et al.* [82]. The analytical solution for the temperature profile is given below

$$T_f = \frac{q'''_G R_N^2}{4k}\left(1 - \frac{r^2}{R_N^2}\right) + \frac{q'''_G R_N}{2h} + T_\infty. \tag{4.17}$$

Equation (4.14) has been discretized using a second-order finite difference stencil. This stencil can be rewritten as an energy balance across each node. For the interior node, the energy balance is given by

$$k_f 2\pi\left(r_n - \frac{\Delta r}{2}\right)\frac{T_{f,n-1} - T_{f,n}}{\Delta r} + k_f 2\pi\left(r_n + \frac{\Delta r}{2}\right)\frac{T_{f,n+1} - T_{f,n}}{\Delta r} + \frac{q'''_{G,n}}{2} 2\pi\left(\frac{3r_n + r_{n+1}}{4}\Delta r\right) + \frac{q'''_{G,n}}{2} 2\pi\left(\frac{3r_n + r_{n-1}}{4}\Delta r\right) = 0 \tag{4.18}$$

where n is the node. Then, the boundary condition at $r = 0$ has an energy balance given by

$$k_f 2\pi\left(\frac{\Delta r}{2}\right)\frac{T_{f,1} - T_{f,0}}{\Delta r} + q'''_{G,0}\pi\left(\frac{\Delta r}{2}\right)^2 = 0 \tag{4.19}$$

while the Robin boundary condition at the exterior node is given by

$$k_f 2\pi\left(r_N - \frac{\Delta r}{2}\right)\frac{T_{f,N-1} - T_{f,N}}{\Delta r} + h2\pi r_N\left(T_\infty - T_{f,N}\right) + \frac{q'''_{G,N}}{2} 2\pi\left(\frac{3r_N + r_{N-1}}{4}\Delta r\right) = 0 \tag{4.20}$$

where N is the exterior boundary node. Table 4.2 displays the parameters used for this test case.

Table 4.2: Parameters for the fuel cylinder

Parameter	**Value**
Radius of fuel R_N	$5 \times 10^{-3}\,\mathrm{m}$
Heat generation in the fuel q'''_G	$5 \times 10^{7}\,\mathrm{W/m^3}$
Thermal conductivity of fuel k	$6\,\mathrm{W/(mK)}$
Surface heat transfer coefficient h	$1000\,\mathrm{W/(m^2K)}$
Coolant temperature T_∞	$300\,\mathrm{K}$

The results are shown in Figure 4.3 with the analytical solution, Equation (4.17). There is excellent agreement between the two results due to the accurate discretization of the boundary conditions [83]. This can be shown more clearly in a convergence plot of the percentage error. Figure 4.4 shows the percentage error does not decrease as the mesh size decreases, which is characteristic of schemes with no spatial discretization error. For comparison, the boundary condition from Incropera *et al.* [84] has also been used:

$$k_f 2\pi\left(r_N - \frac{\Delta r}{2}\right)\frac{T_{f,N-1} - T_{f,N}}{\Delta r} + h2\pi r_N\left(T_\infty - T_{f,N}\right) + q'''_{G,N} 2\pi r_N \Delta r = 0. \tag{4.21}$$

This shows second order convergence in Figure 4.4 even though there is only a slight difference

between the two stencils.

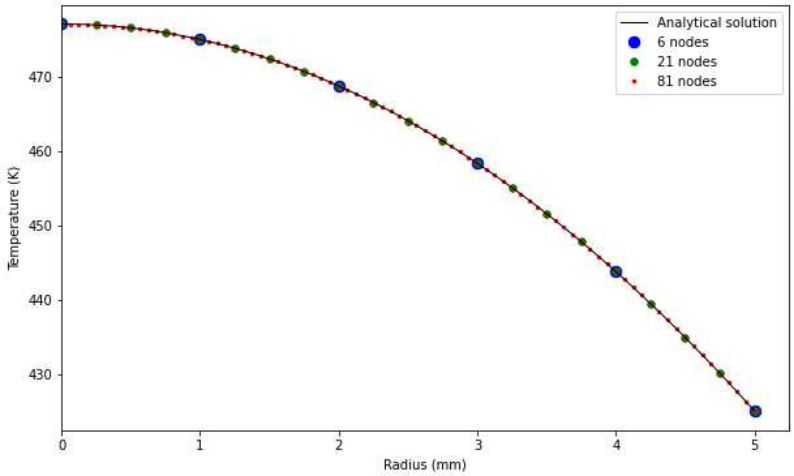


Figure 4.3: Steady state calculation comparing Equation (4.17) to the numerical solution

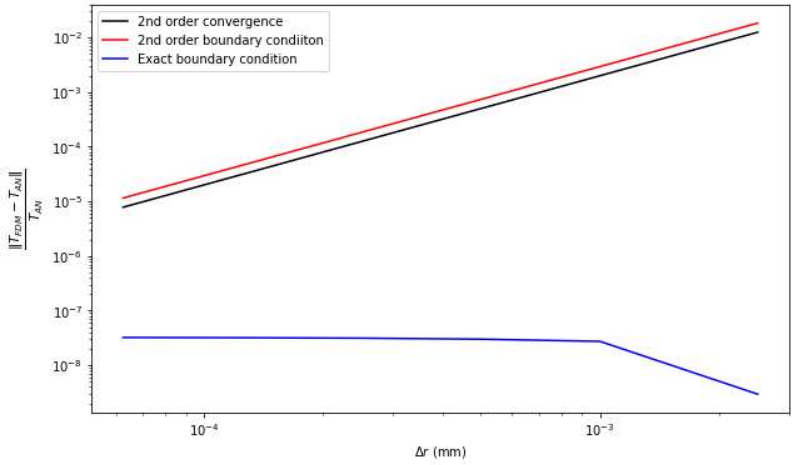


Figure 4.4: Spatial convergence using difference stencils for the Robin boundary condition

In the next test case, the fuel rod is represented as a fuel pellet surrounded by a cladding material. This outer surface of the cladding is exposed to a flowing coolant that has a constant temperature. The fuel pellet has constant thermal properties and a uniform heat generation. The cladding has constant thermal properties and no heat generation. At the pellet-cladding boundary, there is no temperature gradient and the heat flux is equivalent on each side. The

full list of boundary conditions are given below

$$\left.\frac{\partial T_f}{\partial r}\right|_{r=0} = 0 \tag{4.22}$$

$$k_f \left.\frac{\partial T_f}{\partial r}\right|_{r=R_{fo}} = k_c \left.\frac{\partial T_c}{\partial r}\right|_{r=R_{ci}} \tag{4.23}$$

$$T_{fo} = T_{ci} \tag{4.24}$$

$$-k_c \left.\frac{\partial T_c}{\partial r}\right|_{r=R_N} = h(T_N - T_\infty). \tag{4.25}$$

Integrating Equation (4.14) and the above list of boundary conditions results in the temperature profiles for the fuel pellet and the cladding [85] results in

$$T_f = \frac{q'''_G R_{fo}^2}{4k}\left(1 - \frac{r^2}{R_{fo}^2}\right) + \frac{q'''_G R_{fo}^2}{2k_c}\ln\frac{R_N}{R_{ci}} + \frac{q'''_G R_{fo}^2}{2hR_N} + T_\infty \tag{4.26}$$

$$T_c = \frac{q'''_G R_{fo}^2}{2k_c}\ln\frac{R_N}{r} + \frac{q'''_G R_{fo}^2}{2hR_N} + T_\infty \tag{4.27}$$

The energy balance for a fuel-cladding interface is given by

$$\begin{aligned} k_f 2\pi\left(r_{ci} - \frac{\Delta r}{2}\right)\frac{T_{f,ci-1} - T_{f,ci}}{\Delta r} + k_c 2\pi\left(r_{ci} + \frac{\Delta r}{2}\right)\frac{T_{f,ci+1} - T_{f,ci}}{\Delta r} \\ + \frac{q'''_{G,ci}}{2} 2\pi\left(\frac{3r_{ci} + r_{ci-1}}{4}\Delta r\right) = 0 \end{aligned} \tag{4.28}$$

where ci is the node at the boundary. The pellet centreline boundary and the exterior boundary are given in Equations (4.19) and (4.20). Table 4.3 displays the parameters used for this test case.

Table 4.3: Parameters for the fuel cylinder

Parameter	**Value**
Outer radius of cladding R_N	5×10^{-3} m
Inner radius of cladding R_{ci}	4×10^{-3} m
Outer radius of fuel R_{fo}	4×10^{-3} m
Heat generation in the fuel q'''_G	5×10^7 W/m^3
Thermal conductivity of cladding k_c	21 W/(mK)
Thermal conductivity of fuel k_f	6 W/(mK)
Surface heat transfer coefficient h	1000 W/(m^{2}K)
Coolant temperature T_∞	300 K

Figure 4.5 shows good agreement between the numerical and analytical solution. The discretized equation accurately represented the model, which had different thermal properties for each material.

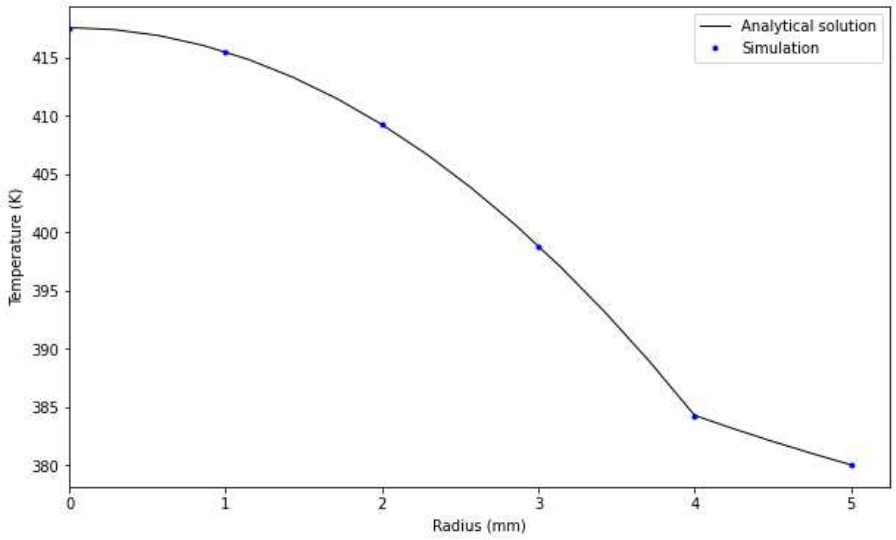


Figure 4.5: Steady state calculation comparing Equations 4.26 and 4.27 to the numerical solution

The previous test cases have simplified the geometry of a fuel rod. In reality, there is a gas gap between the fuel pellet and the cladding where the gaseous fission products build up during operation. This causes the thermal conductivity of the gap to change over the lifespan of the core. A gap conductance coefficient is defined which is the summation of three heat paths: gas conductance, solid conductance and direct thermal radiation.

$$h_{gap} = h_g + h_s + h_r \tag{4.29}$$

where h_g is the gas conductance, h_s is the solid conductance, and h_r is the radiation. There are common models for each of these heat paths that can account for variations in thermal conductivity, gap distance and surface roughness [86]. Instead, a constant value has been used in this heat conduction model. The heat transfer between the fuel pellet and the interior cladding surface can then be modelled using Robin boundary conditions. The list of boundary conditions are given below:

$$\left.\frac{\partial T_f}{\partial r}\right|_{r=0} = 0 \tag{4.30}$$

$$-k_f \left.\frac{\partial T_f}{\partial r}\right|_{r=R_{fo}} = h_{gap}(T_{fo} - T_{ci}) \tag{4.31}$$

$$-\frac{R_{ci}}{R_{fo}} k_c \left.\frac{\partial T_c}{\partial r}\right|_{r=R_{ci}} = h_{gap}(T_{fo} - T_{ci}) \tag{4.32}$$

$$-k_c \left.\frac{\partial T_c}{\partial r}\right|_{r=R_N} = h(T_N - T_\infty). \tag{4.33}$$

Using the above boundary conditions and Equation (4.14), an analytical temperature profile can be derived [87]. An intermediate step within this process is to define the temperature at each boundary:

$$T_N = \frac{q'''_G R_{fo}^2}{2hR_N} + T_\infty \tag{4.34}$$

$$T_{ci} = \frac{q'''_G R_{fo}^2}{2k_c} \ln\frac{R_N}{R_{ci}} + T_N \tag{4.35}$$

$$T_{fo} = \frac{q'''_G R_{fo}}{2h_{gap}} + T_{ci} \tag{4.36}$$

$$T_o = \frac{q'''_G R_{fo}^2}{4k_f} + T_{fo}. \tag{4.37}$$

Combining these terms together, the temperature profiles of the fuel pellet and cladding are obtained

$$T_f = T_o - \frac{q'''_G r^2}{4k_f} \tag{4.38}$$

$$= \frac{q'''_G R_{fo}^2}{4k}\left(1 - \frac{r^2}{R_{fo}^2}\right) + \frac{q'''_G R_{fo}}{2h_{gap}} + \frac{q'''_G R_{fo}^2}{2k_c}\ln\frac{R_N}{R_{ci}} + \frac{q'''_G R_{fo}^2}{2hR_N} + T_\infty \tag{4.39}$$

$$T_c = \frac{q'''_G R_{fo}^2}{2k_c}\ln\frac{R_N}{r} + \frac{q'''_G R_{fo}^2}{2hR_N} + T_\infty \tag{4.40}$$

where k_c is the cladding thermal conductivity. The discretized energy balance for Equation

(4.31) is given by

$$k_f 2\pi\left(r_{fo}-\frac{\Delta r}{2}\right)\frac{T_{f,fo-1}-T_{f,fo}}{\Delta r}+h_{gap}2\pi r_{fo}\left(T_{f,ci}-T_{f,fo}\right) + \frac{q'''_{G,fo}}{2}2\pi\left(\frac{3r_{fo}+r_{fo-1}}{4}\Delta r\right)=0 \tag{4.41}$$

where fo is the boundary node on the outer surface of the fuel. The discretized energy balance for Equation (4.32) is given by

$$k_c 2\pi\left(r_{ci}+\frac{\Delta r}{2}\right)\frac{T_{f,ci+1}-T_{f,ci}}{\Delta r}+h_{gap}2\pi r_{fo}\left(T_{f,ci}-T_{f,fo}\right)=0 \tag{4.42}$$

where ci is the boundary node on the inner surface of the cladding. Table 4.4 displays the parameters for the gap conductance test case.

Table 4.4: Parameters for the fuel cylinder

Parameter	**Value**
Outer radius of cladding R_N	$4.9\times10^{-3}\,\mathrm{m}$
Inner radius of cladding R_{ci}	$4.1\times10^{-3}\,\mathrm{m}$
Fuel gap size	$0.1\times10^{-3}\,\mathrm{m}$
Outer radius of fuel R_{fo}	$4\times10^{-3}\,\mathrm{m}$
Heat generation in the fuel q'''_G	$5\times10^{7}\,\mathrm{W/m^3}$
Thermal conductivity of cladding k_c	$21\,\mathrm{W/(mK)}$
Thermal conductivity of fuel k_f	$6\,\mathrm{W/(mK)}$
Surface heat transfer coefficient h	$1000\,\mathrm{W/(m^2K)}$
Gap heat transfer coefficient h_{gap}	$6000\,\mathrm{W/(m^2K)}$
Coolant temperature T_∞	$300\,\mathrm{K}$

The results in Figure 4.6 show good agreement between the analytical and numerical model.

4.4.2 Verification of transient solution

This transient test case represents a heated cylinder with a coolant flowing past it. No heat generation is present in the cylinder and the temperature at the surface is identical to the coolant temperature. The boundary conditions and analytical temperature profile [88] are

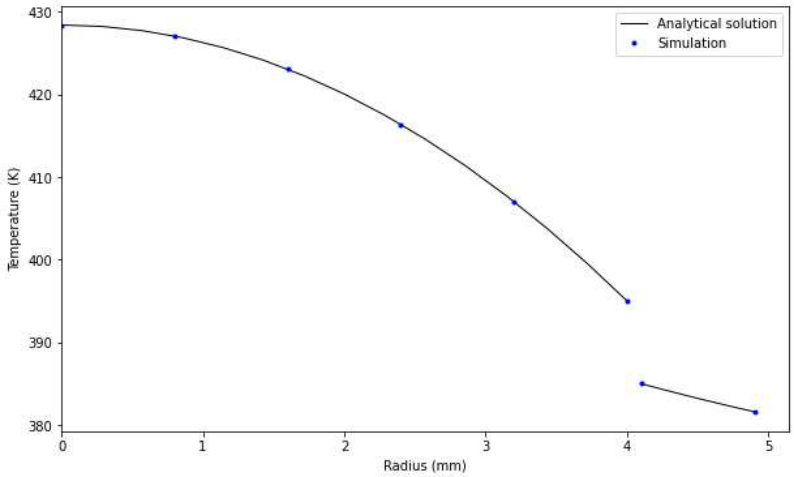


Figure 4.6: Steady state calculation comparing Equations (4.39) and (4.40) to the numerical solution

given below

$$\left.\frac{\partial T_f}{\partial r}\right|_{r=0} = 0 \tag{4.43}$$

$$T_N = T_\infty \tag{4.44}$$

$$T_f = T_\infty + (T_o - T_\infty)2\sum_{n=1}^{\infty}\frac{1}{\lambda_n}\exp\lambda_n^2\frac{\alpha t}{R_N^2}\frac{J_0(\lambda_n\frac{r}{R_N})}{J_1(\lambda_n)}, \tag{4.45}$$

where J_0 is a Bessel function of the zeroth order, J_1 is a Bessel function of the first order and λ_n are the roots of J_0. T_o is the initial temperature at the centre of the fuel and α is the the thermal diffusivity coefficient, $\frac{k_f}{\rho_f C_{p,f}}$.

Table 4.5 shows the parameters for the transient heat conduction test case.

Table 4.5: Parameters for the fuel cylinder

Parameter	**Value**
Radius of fuel R_N	4.75×10^{-3} m
Thermal conductivity of fuel k_f	6 W/(mK)
Diffusion coefficient α	1×10^{-5}
Heat capacity of fuel $C_{p,f}$	500 W/(mK)
Initial temperature T_o	350 K
Coolant temperature T_∞	300 K

A first order forward Euler scheme has been used for the time integration. The time step size is 0.01 s and the grid size is 0.01 m. Using these parameters, Figure 4.7 shows there is good agreement with the analytical solution, though a higher order time integration would improve

the accuracy of these results.

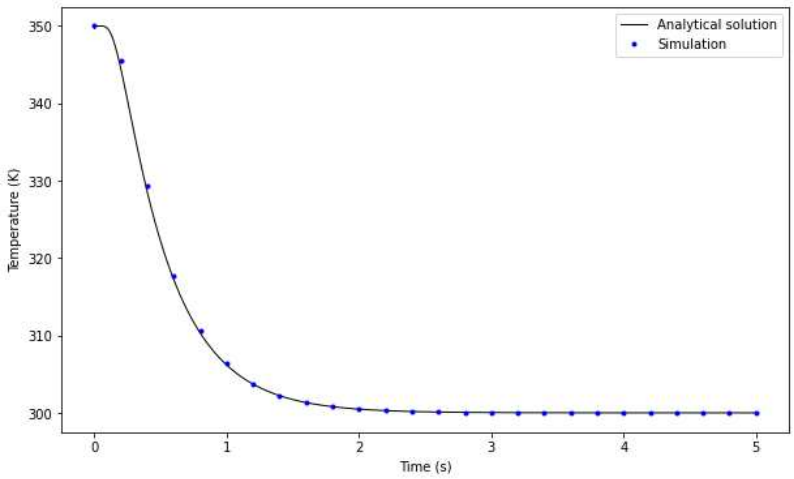


Figure 4.7: Transient calculation comparing Equation (4.45) to the numerical solution

4.5 Subchannel analysis validation

4.5.1 PNL 7×7 sleeve blockage test

The Pacific North Laboratory (PNL) assembly test [89] investigated how a sleeve blockage in a fuel rod bundle affected the surrounding turbulent flow. Additionally, this test case was used for validating the subchannel code COBRA-IIIC. The experimental setup consisted of a 7×7 rod bundle with rods of diameter 0.01 m and a pitch of 0.0137 m. The sleeve blockage around the 9 central rods reduced the area of the central subchannels by 70%, as shown in Figure 4.8. The test was performed at 0.12 MPa and 302.6 K with an inlet Reynolds number of 2.95×10^4. In the numerical model, a structured grid was used of 101 cells in the axial direction, 8 cells in the x-direction and 8 in the y-direction (6400 cells). The sub-channel models are given in Table 4.6, which also includes the loss coefficients for the spacer grids.

Figure 4.5.1, shows the results from two different meshes compared to the experimental data. The porous-media code predicted the fluid velocity within a 20% error range. The mean error using all the experimental data points was 7.8%. In Figure 4.10 the simulation results are compared to MATRA subchannel code and a resolved CFD simulation from Liu *et al.* [38].

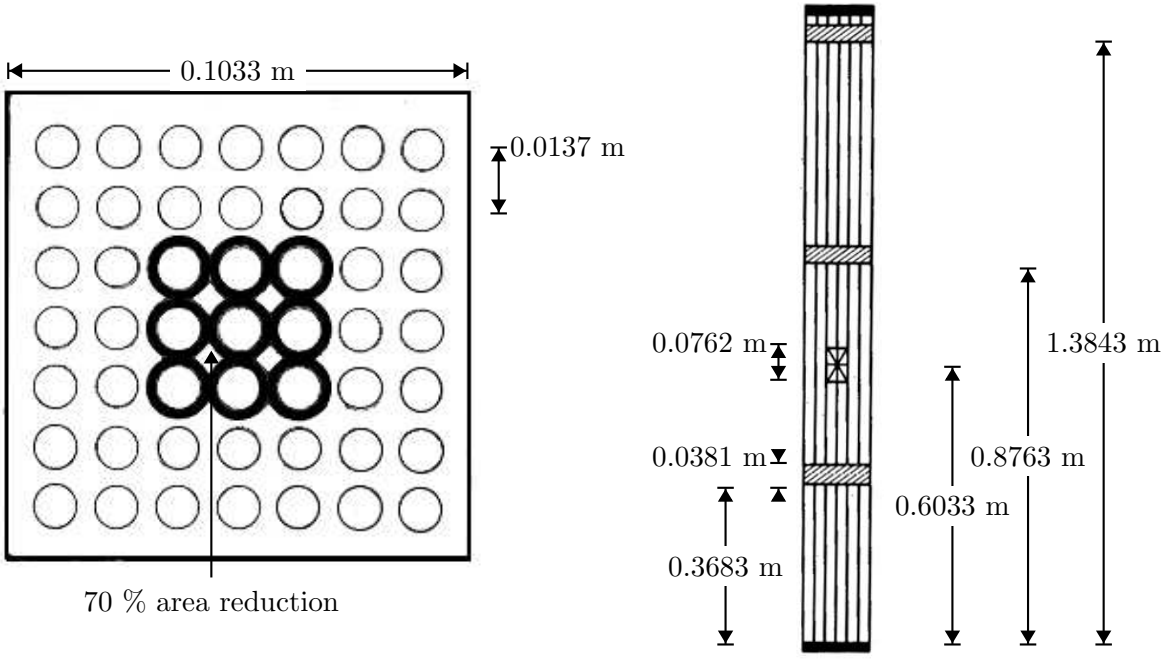


Figure 4.8: Layout of the PNL 7 × 7 assembly

Table 4.6: Parameters for PNL 7x7 assembly test

Parameter	**Value**
Fuel assembly parameters	
Number of fuel rods	49
Fuel rod diameter	0.01 m
Pitch	0.0137 m
Cross-sectional area	0.1033 m × 0.1033 m
Fuel rod length	1.3843 m
Operation Pressure	0.12 MPa
Operation Temperature	302.6 K
Subchannel T/H models	
Friction loss	$f = 0.34Re^{-0.25}$
Spacer loss coefficient	$K''_{axial} = 1.14$
Blockage loss coefficient	$K''_{axial} = 0.5$
Turbulent mixing	$\beta = 0.02$

This resolved CFD results closely match the Porous-media CFD results, especially around the blockage. This further validates the source terms in Equation 4.4for 1-dimensional pressure losses. Figure 4.11 displays a 3D plot of the velocity in the axial direction. The central subchannels show a sharper change in the velocity than the exterior subchannels.

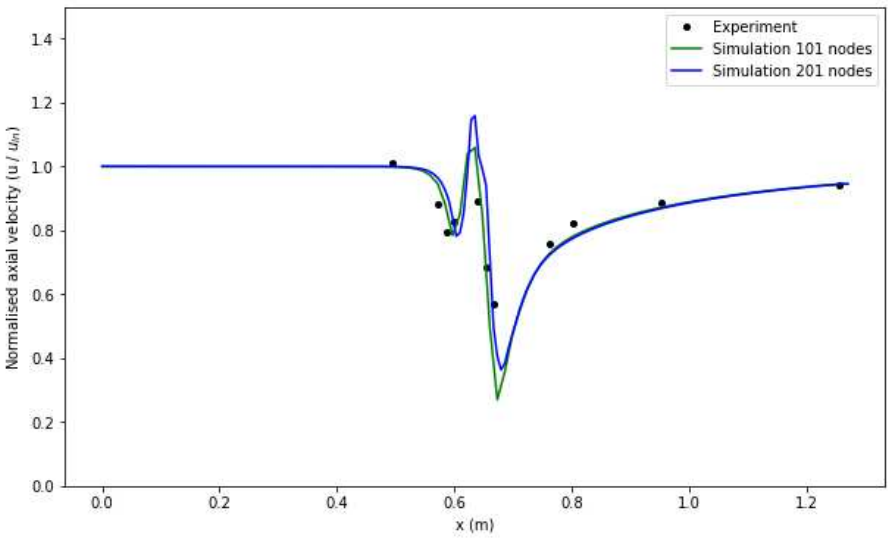


Figure 4.9: Normalized axial velocity along the centre subchannel with a 70% blockage

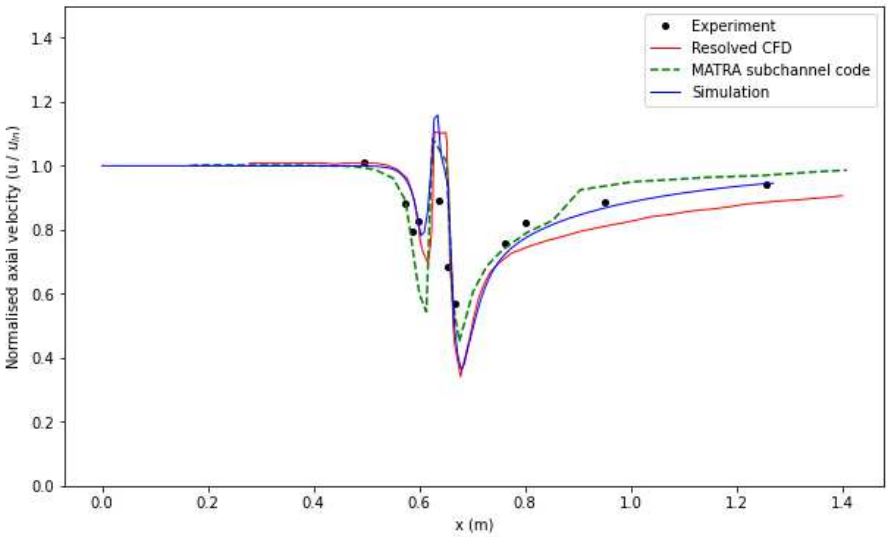


Figure 4.10: Normalized axial velocity comparison between different numerical solvers

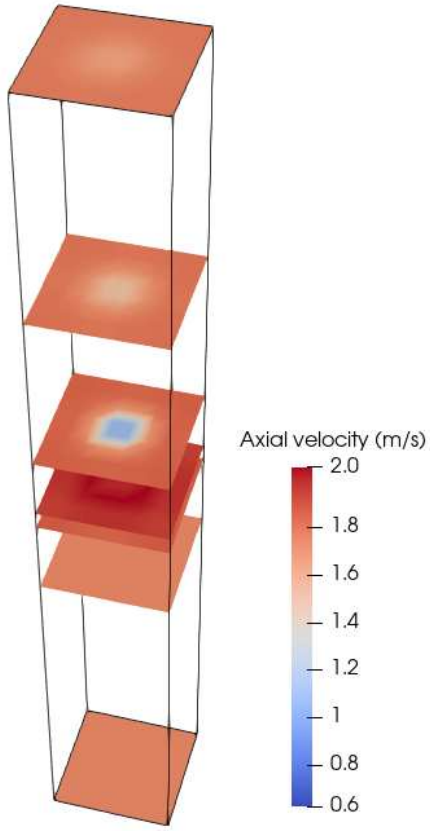


Figure 4.11: The velocity at different levels for PNL 7 × 7

4.5.2 Westinghouse two 14 × 14 assembly mixing test

The Westinghouse (WH) 14 × 14 assembly test [90] was carried out to investigate the flow redistribution between two fuel assemblies separated by a water gap. Each assembly measured 0.3893 m × 0.1938 m, and had an axial height of 0.9652 m as shown in Figure 4.12. The rod diameter was 0.0108 m and the pitch to diameter ratio was 1.28. The inlet velocities of each assembly were 3.52 m/s and 1.76 m/s, while the inlet velocity of the water gap was 2.64 m/s. The numerical model has used a structured mesh, consisting of 38 cells in the axial direction, 15 cells in the x-direction and 29 in the y-direction (16,530 cells). The subchannel models in this simulation are given in Table 4.7.

The results in Figure 4.13 show the axial velocity at six different horizontal cross-sections. The mean error for the axial velocity between the porous media code and the experimental data was

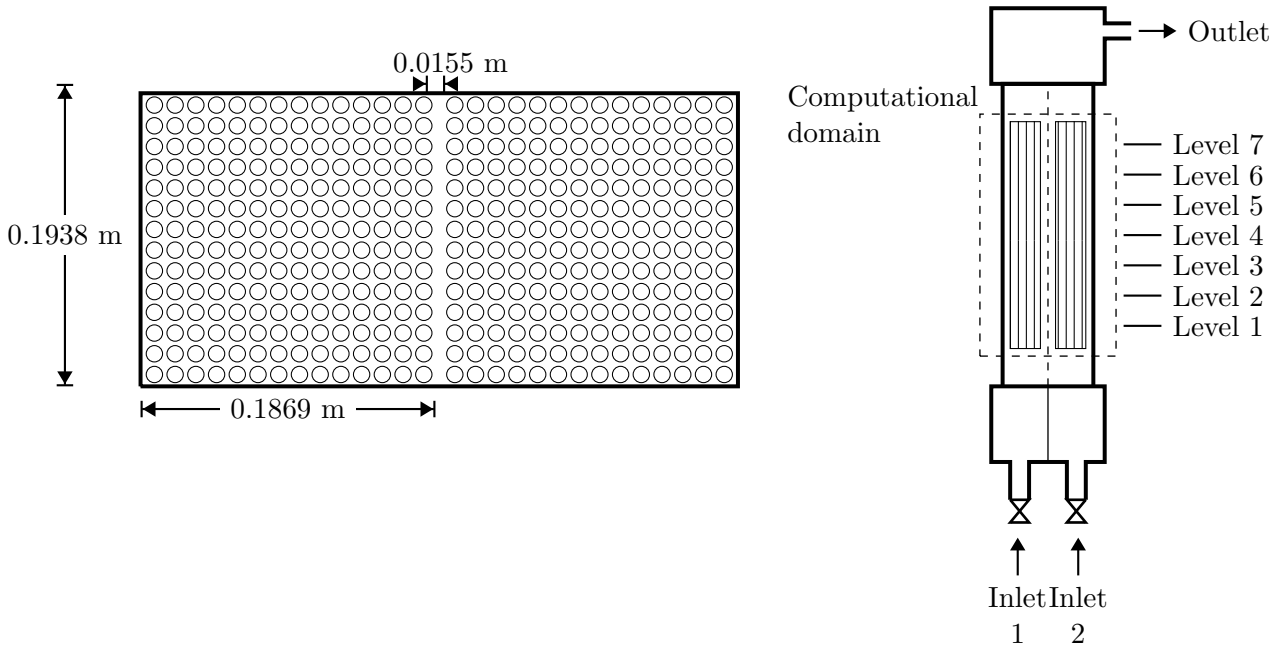


Figure 4.12: Layout of the WH 14 x 14 assemblies

Table 4.7: Parameters for WH 14 × 14 assembly test

Parameter	**Value**
Fuel assembly parameters	
Number of fuel rods	196
Fuel rod diameter	0.0108 m
Pitch	0.0138 m
Cross-sectional area	0.1938 m × 0.3893 m
Fuel rod length	0.9652 m
Operation Pressure	0.1 MPa
Operation Temperature	299.8 K
Subchannel T/H models	
Friction loss	$f = 0.316Re^{-0.25}$
Turbulent mixing	$\beta = 0.02$

10.8%. This discrepancy between the results is noticeable in the given Figure. One reason for this is the volume-averaging in the porous-media approximation. This assumes each subchannel has a uniform velocity, whereas in reality there is a significant velocity gradient between the edge of the subchannel on the surface of the fuel rods and the centre of the subchannel. This is more accurately represented in the coarse-grid CFD results where Figure 4.13 shows how the velocity undulates between each subchannel. The velocity field in fuel assemblies is not precisely captured in porous-media CFD or subchannel codes even though the results are still reasonable.

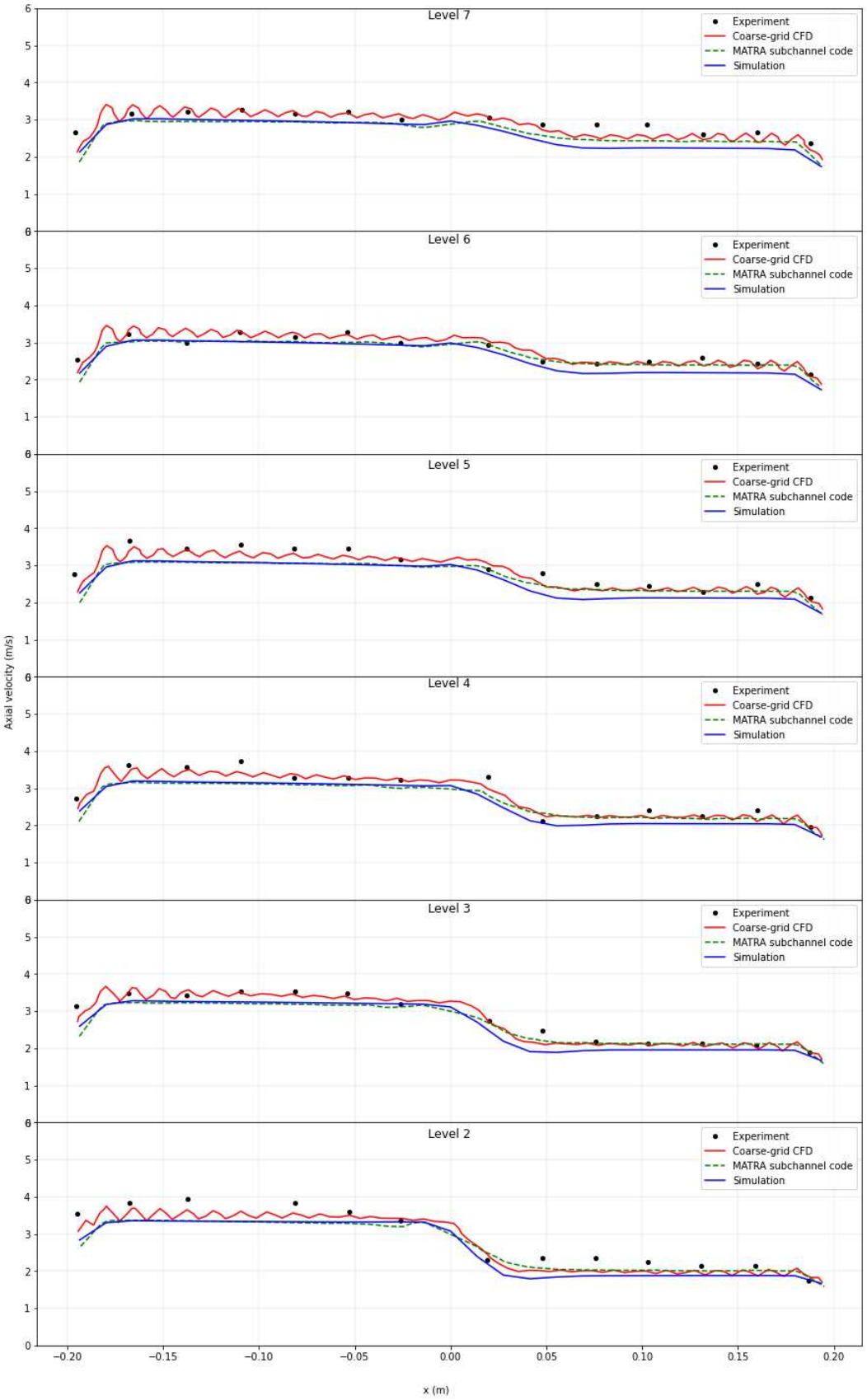


Figure 4.13: A comparison between the present study, the experimental results and CUPID's porous media code

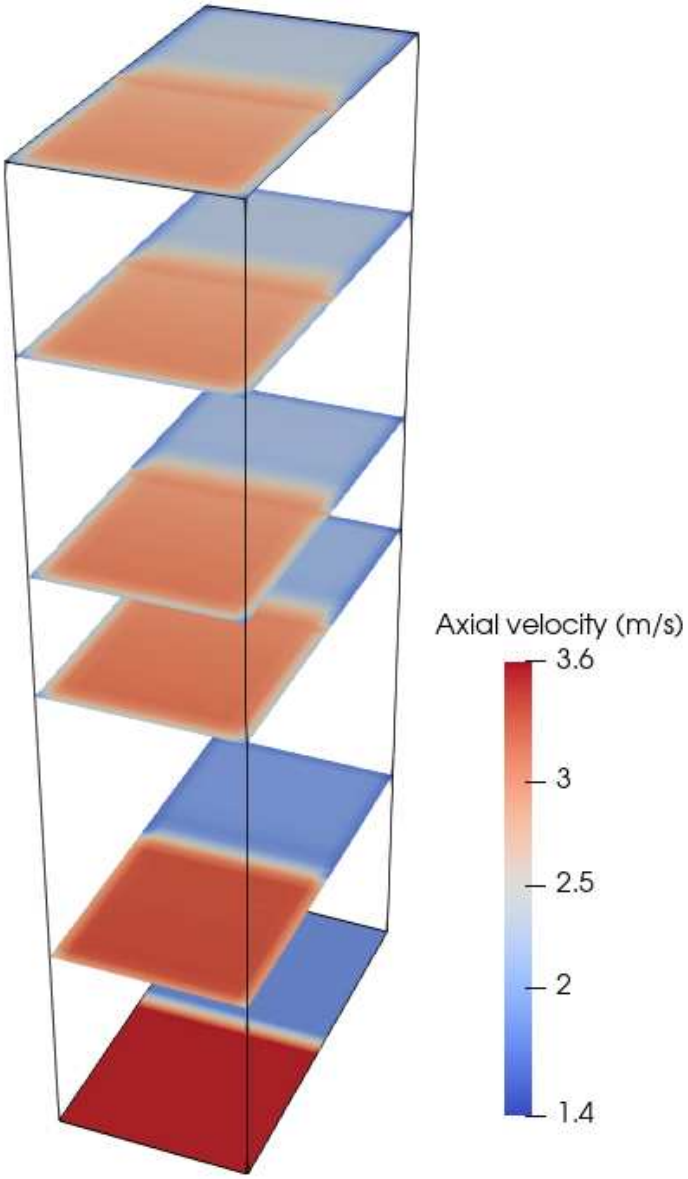


Figure 4.14: The velocity at each level for the WH 14 x 14 assembly test

4.5.3 Pacific Northwest Laboratories 2 × 6 buoyancy test

The Pacific Northwest Laboratories (PNL) 2×6 test was carried out to investigate the effect of buoyancy on flow distribution. The assembly had dimensions of 0.0324 m by 0.908 m with a heated length of 1.21 m. To test the effect of buoyancy, the heating rate was varied between the 12 rods in each test case. This caused variations in the density of the fluid between the subchannels. In the test case used here, the heating rate was 1.134 kW for 6 rods and 0.567 kW for the other 6 rods. This operated at 0.1 MPa and 285.2 K with an inlet velocity of 0.1 m/s. A structured grid was used of 24 cells in the axial direction, 7 cells in the x-direction and 3 in the y-direction (504 cells). To account for how the density varies as a function of temperature, the Boussinesq approximation was added as a source term.

$$S_M = \frac{f}{D_h}\frac{\rho}{2}\,\boldsymbol{u}|\boldsymbol{u}| + \frac{K''}{\Delta z}\frac{\rho}{2}\boldsymbol{u_0}|\boldsymbol{u_0}| - \rho g \alpha_V (T - T_0), \tag{4.46}$$

where α_V is the thermal expansion coefficient, g is the gravitational acceleration and T_0 is the reference temperature. The other thermal hydraulic correlations are shown in Table 4.8.

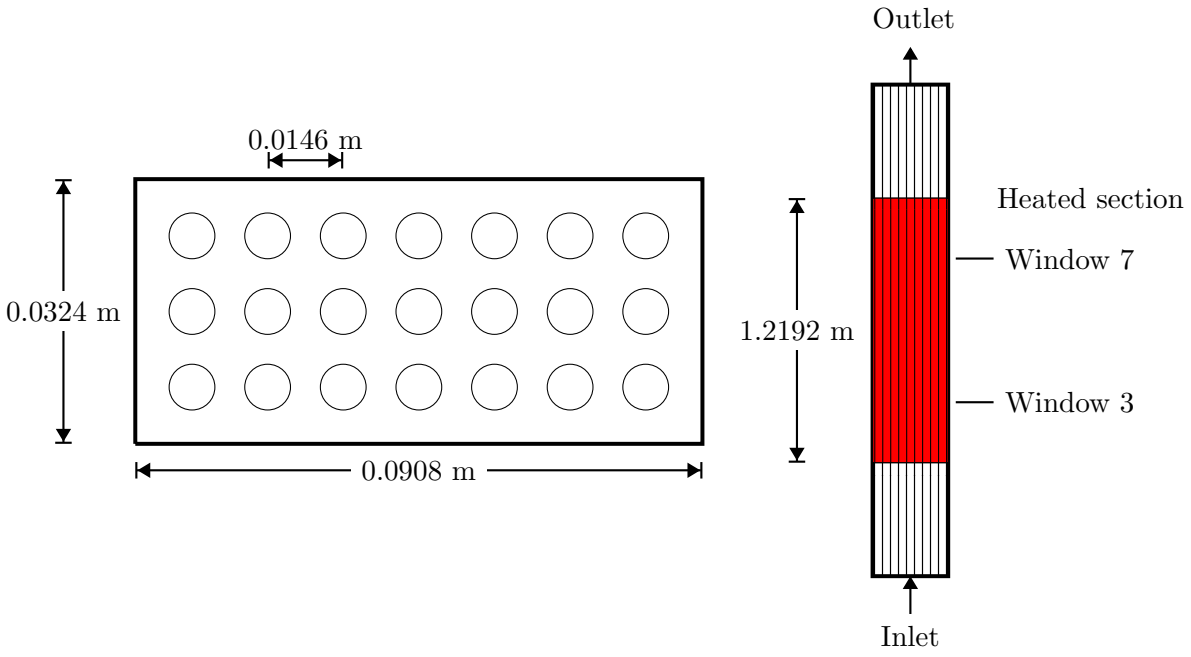


Figure 4.15: Layout of the PNL 2 x 6 assembly

Table 4.8: Parameters of the PNL 2 x 6 assembly

Parameter	**Value**
Fuel rod parameters	
Fuel rod diameter	$0.012\,\mathrm{m}$
Fuel rod length	$1.2192\,\mathrm{m}$
Density	$10,970\,\mathrm{kg/m^3}$
Thermal conductivity	$6.0869\,\mathrm{W/(m\,K)}$
Specific heat capacity	$262.43\,\mathrm{J/(kg\,K)}$
Fuel assembly parameters	
Number of fuel rods	12
Pitch	$0.0146\,\mathrm{m}$
Cross-sectional area	$0.0324\,\mathrm{m} \times 0.0908\,\mathrm{m}$
Thermal power of hot fuel rod	$1.134\,\mathrm{kW}$
Operational Pressure	$0.1\,\mathrm{MPa}$
Inlet Temperature	$285.15\,\mathrm{K}$
Subchannel T/H models	
Friction loss	$f = 64Re^{-1}$
Turbulent mixing	$\beta = 0.005$
Thermal expansion coefficient	$\alpha_V = 210 \times 10^{-6}\,\mathrm{K^{-1}}$

Figure 4.16 shows the temperature and velocity in sub-channels 8 to 14 at two different positions along the fuel assembly. The mean error for the temperature between the porous media code and the experimental data was $< 1\%$. This is mainly because this benchmark was performed over a very narrow temperature range. The results in Figure 4.16 shows the discrepancy is still very noticeable. Similarly to the previous test case, the gradient between the centre and edge of each subchannel is not captured because each subchannel is volume-averaged. This is shown in Figure 4.16 by comparing the subchannel models to the resolved CFD. The resolved CFD is more in line with the experimental results and models the temperature undulations between the hot surface of the fuel rods and the colder centre of each subchannel. Also, constant values of thermo-physical properties were used in the porous media simulation. The thermo-physical properties vary with temperature, which is why updating these values at each time-step would be more accurate.

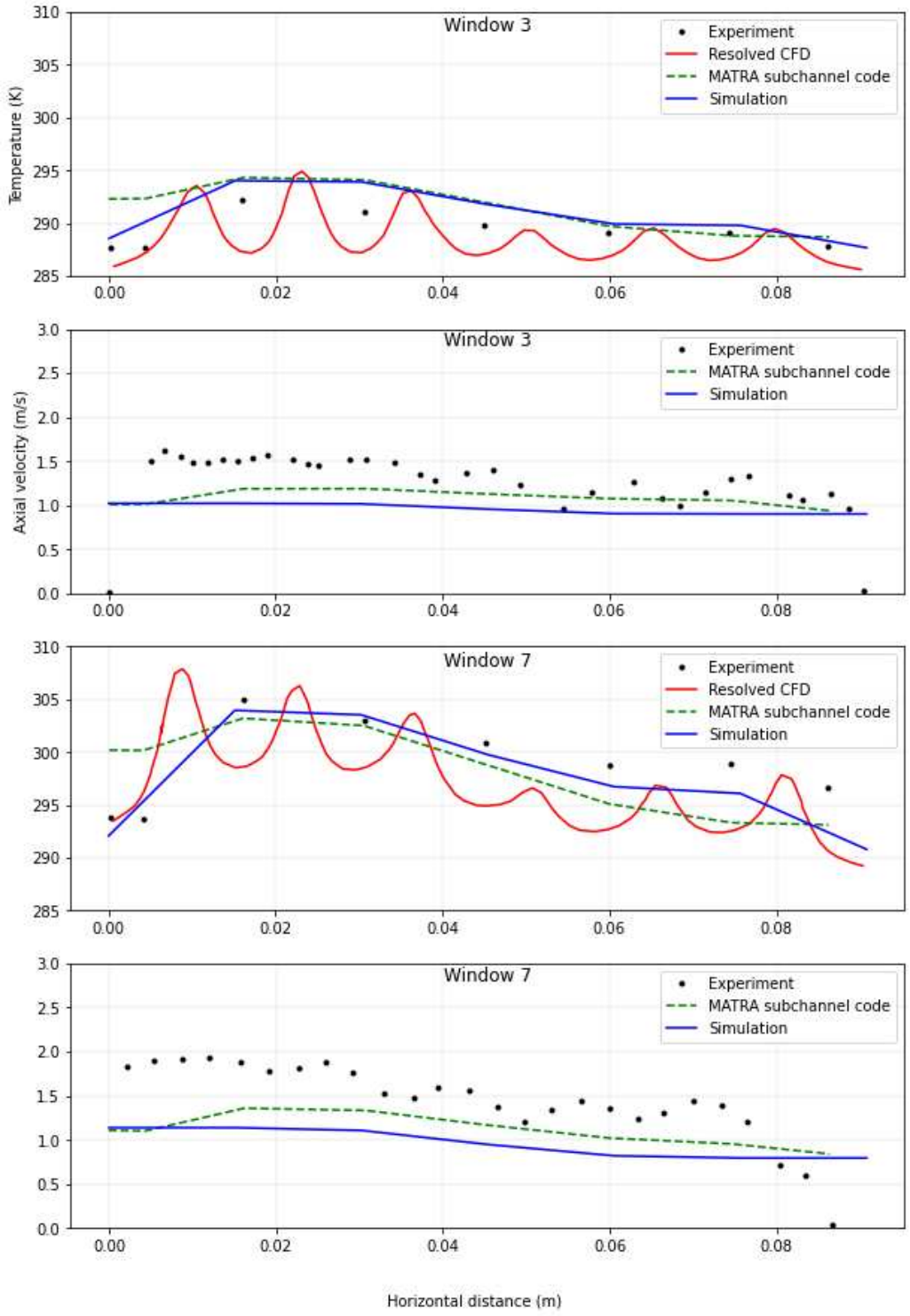


Figure 4.16: A comparison between the experimental results, resolved CFD results from Liu *et al.* [19], MATRA's subchannel results and the present study

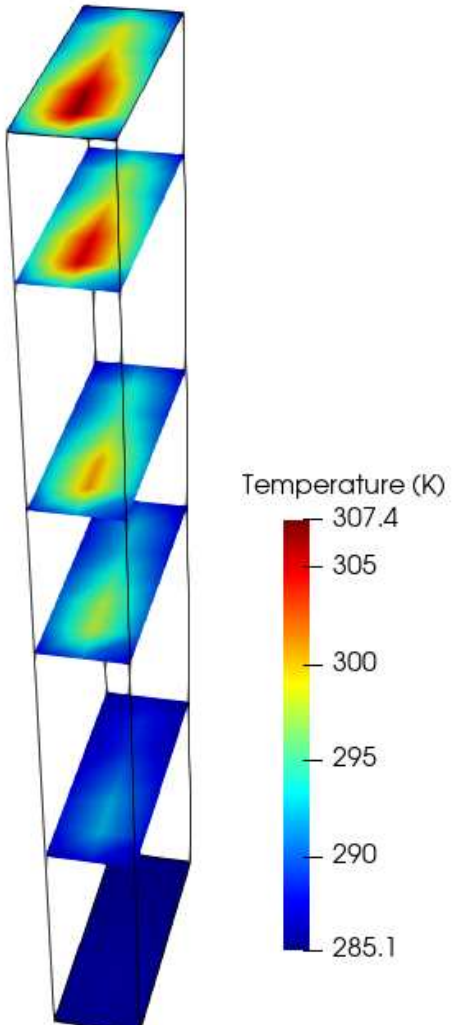


Figure 4.17: The temperature at different axial positions for the PNL 2 x 6 assembly test

4.6 Conclusions

In this chapter, the porous-media CFD code has been validated against nuclear thermal hydraulics benchmarks. This uses the numerical methods presented in the previous chapter for the source term discretization. The first test cases verified the discretized heat equation using analytical solutions. This was performed using steady state and transient test cases for 1D heat transfer. The next test cases validated the subchannel models using experimental data. This showed that subchannel-scale analysis can be performed accurately using this code. In future, more models could be added to represent more complicated behaviour. For example, the tur-

bulence in mixing vanes could be simulated using additional mixing vane turbulence models. Also, additional pressure drop models could be added for different fuel assembly geometries. In the next chapter, the numerical solver shown here will be used to simulate a pressurized water reactor core.

Chapter 5

Nuclear reactor core simulation on CPU and GPU architectures

In this chapter, the fluid flow and heat transfer within a pressurized water reactor core has been simulated using the porous-media CFD code. This uses the numerical methods and subchannel models from the previous chapter for the numerical solver. An important feature of this code is the ability to run simulations on different computing architectures. To test this, numerical simulations have been performed on CPU and GPU architectures using different parallel programming frameworks.

5.1 Introduction

In the early development of subchannel codes, the geometry of the reactor core was represented by using a very coarse approximation to perform numerical simulations. Each fuel assembly was represented as a subchannel in the computational domain. Since then, there have been continuous advances in computing hardware leading to more powerful computing systems. Parallel computing frameworks allow simulations to run on multiple central processing units (CPUs) simultaneously. This has motivated research into novel numerical solvers which can run efficiently in parallel. Computational modelling for nuclear thermal hydraulics has benefited

from these advancements. It has allowed pin-level thermal hydraulics (T/H) analysis of a reactor core to become a feasible option. This is where each individual fuel rod is modelled and every flow channel between the fuel rods is represented by a subchannel in the computational domain. Using the porous media code developed in this thesis, a whole core pin level T/H analysis will be conducted for a pressurized water reactor.

Several studies have performed for pin level T/H simulations using parallel programming frameworks on high performance computing clusters. COBRA-TF, a legacy Fortran code, was parallelized by assigning one Message Passing Interface (MPI) process per fuel assembly [91] and solving the pressure equation using the Parallel Extensible Toolkit for Scientific Computation (PETSc) library. This improved the performance allowing large problems to be solved on multiple cores. Similarly, CUPID [22] and ESCOT [40], two subchannel scale codes developed in Korea, were also parallelized using MPI for full core simulations. START is another subchannel code developed in Korea [41]. This was parallelized using OpenMP and showed good parallel performance for a PWR core simulation. SACOS, a subchannel code developed by XJTU-NuTheL [42] used a hybrid OpenMP and MPI parallel scheme for more flexibility.

The studies listed above all focused on scaling for CPUs. There is less evidence that thermal hydraulics codes are being run on graphic processing units (GPU). There are a few noteable examples, including a preliminary study from ESCOT [92] and a more recent study by Luo *et al.* [93]. Apart from these exceptions, many current subchannel codes are updated iterations of legacy codes that were developed before GPUs became widespread. It is a time-consuming process to redevelop them for new architectures. In contrast, many modern CFD codes have supported GPUs for several years [94], [95]. It is easier to develop a GPU-based solver if utilizing GPUs is mapped out early in the development process. But, implementing CPU parallelism and GPU-acceleration can be a difficult task. This requires understanding parallel programming frameworks, computer hardware and memory management. For domain specialists working in scientific simulation, they need to have a broad understanding of both computer science and their respective domain. The time spent doing software development will be split between the physical modelling and the parallel algorithms.

Automatic code generation could aid in this area by separating the development of physical models from the parallel algorithms. This can substantially reduce the time taken to write a GPU-based numerical solver. In domain specific languages which use automatic code generation, the differential equations are written in a high level syntax. This has has been demonstrated in OpenSBLI [44] and PyFR [53], which are both high-order CFD solvers that run efficiently on CPU and GPU architectures. Devito [45] is another code-generation framework that runs on CPUs and GPUs. This is the framework used for running the whole core pin level T/H simulation.

In Section 5.2 a brief explanation is given on how Devito supports parallel programming. In Section 5.3 the specifications for the PWR test case are given followed by the numerical results from the CFD solver. In Section 5.4, the scaling results are shown for the simulations on a HPC cluster.

5.2 Parallel computing using Devito

In Chapter 2, an overview was given on how the Devito DSL is used to solve partial differential equations. Devito has mainly been used for seismic imaging problems by numerically solving the wave equation [96, 97]. Some of these seismic imaging techniques, such as full-waveform inversion require a huge computational cost to generate the numerical solution. This is why a great deal of work has gone into optimization methods for the generated C code.

These methods include parallel programming optimizations that can be tailored to the specific hardware the users code is being run on. Examples include SIMD vectorization, OpenMP for CPU parallelism, OpenACC for GPU parallelism and MPI for multi-node parallelism. Enables these methods can be done without making any modifications to the user's code. This separation of layers between the symbolic equations and the parallel computing can make the user's code more maintainable and future proof for new architectures. In comparison, hand-coding the parallel implementation could be a difficult task.

Previous studies have investigated the performance of Devito on CPU and GPU architectures.

These have focused on explicit solvers applied to geophysics benchmarks. In this work, the objective is to do a performance evaluation for a thermal hydraulics test case. The CFD code used here solves for pressure implicitly at each time step as shown in Chapter 4 Section 4.3.

5.3 PWR core simulation

The test cases in the following section have been taken from the study by Liu *et al.* [19], which uses the parameters for a pressurized water reactor core operating at full power. The first test case is the simulation of a single fuel assembly under steady-state conditions. This contains 264 fuel rods and 25 guide tubes. A detailed list of the parameters is given in Table 5.1. The diagrams of the thermal power factor are shown in Figure 5.1, where the left image shows the axial power factor, while the right image shows the horizontal power factor of each fuel rod. The numerical simulations were performed on a cartesian grid of $18 \times 18 \times 50$ cells. This represents the 324 subchannels in the fuel assembly divided into 50 axial nodes. Each subchannel was coupled to a heat equation for the fluid-solid heat transfer. The heat equation was solved over 18 radial nodes, representing the pin-scale temperature distribution.

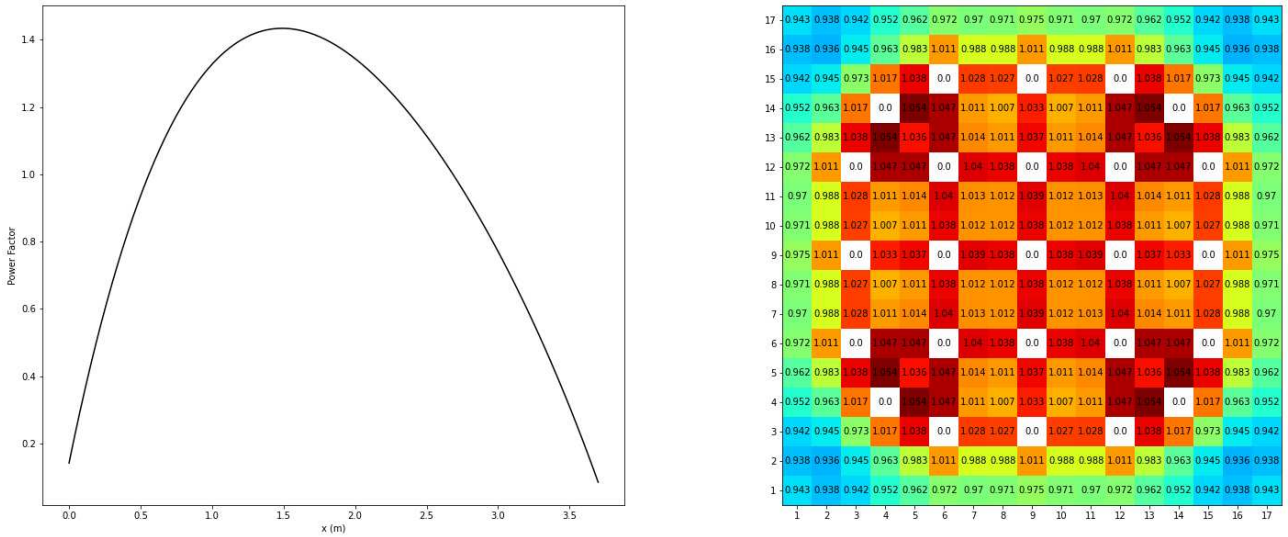


Figure 5.1: Axial and horizontal power factor within a single fuel assembly

Figures 5.3 and 5.5 show that the pellet temperature along its centreline and the coolant temperature are slightly overpredicted compared to the results from COBRA [19]. There is

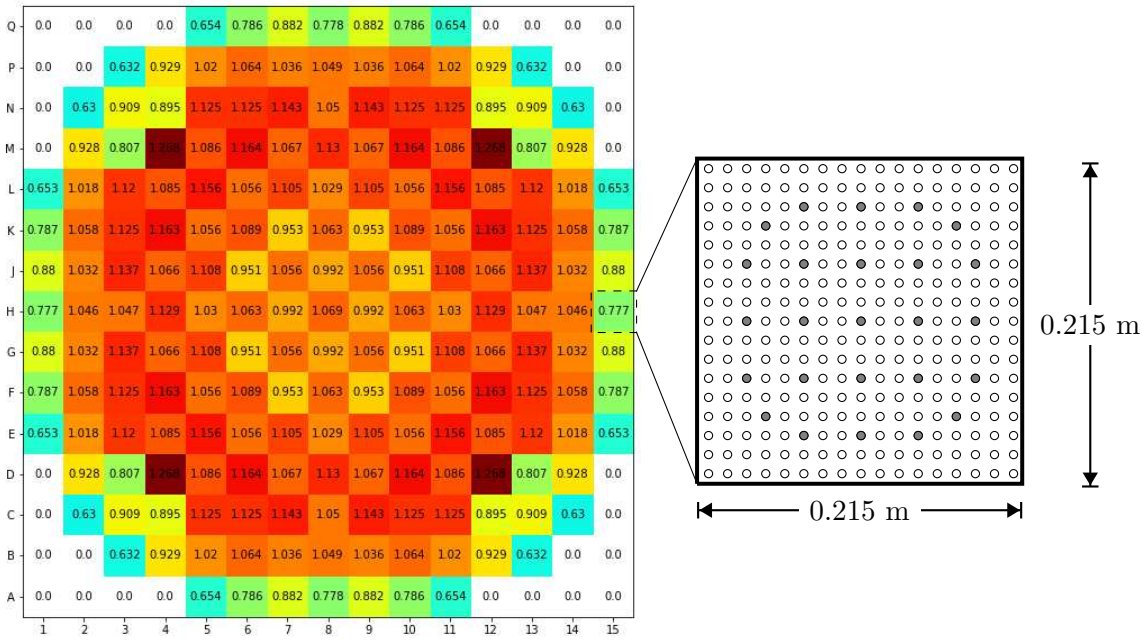


Figure 5.2: The Power factor of the PWR core. Each 0.215×0.215 assembly in the core has the power distributions shown above in Figure 5.1

Table 5.1: Parameters for fuel assembly

Parameter	**Value**
Fuel rod parameters	
Fuel rod diameter	0.0095 m
Fuel pellet diameter	0.00819 m
Fuel rod length	3.658 m
Density	$10,970\,\mathrm{kg/m^3}$
Thermal conductivity	$6.0869\,\mathrm{W/(m\,K)}$
Specific heat capacity	$262.43\,\mathrm{J/(kg\,K)}$
Fuel assembly parameters	
Number of fuel rods	264
Number of guide tubes	25
Pitch	0.0126 m
Cross-sectional area	$0.215\,\mathrm{m} \times 0.215\,\mathrm{m}$
Thermal power	17.7 MW
Operational Pressure	15.5 MPa
Inlet Temperature	566 K
Subchannel T/H models	
Friction loss	$f = 0.316Re^{-0.25}$
Turbulent mixing	$\beta = 0.05$
Gap conductance	$h_{gap} = 30000$

also a discrepancy in the cladding surface temperature as shown in Figure 5.4. One reason for this is the estimates for the thermo-physcial properties. In the present study, constant values were used based on the coolant temperature at the central axial position of the fuel assembly.

In COBRA's solution procedure, the thermo-physical properties are updated at each time step. This can have a large influence on the calculated results because many of the properties are functions of temperature. For example, the density of the fluid varies between the inlet of the fuel assembly and the outlet as the temperature increases.

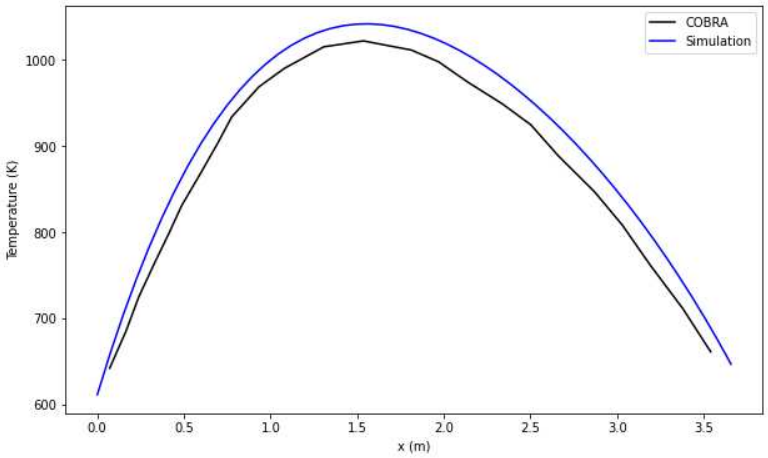


Figure 5.3: Pellet temperature of the fuel assembly

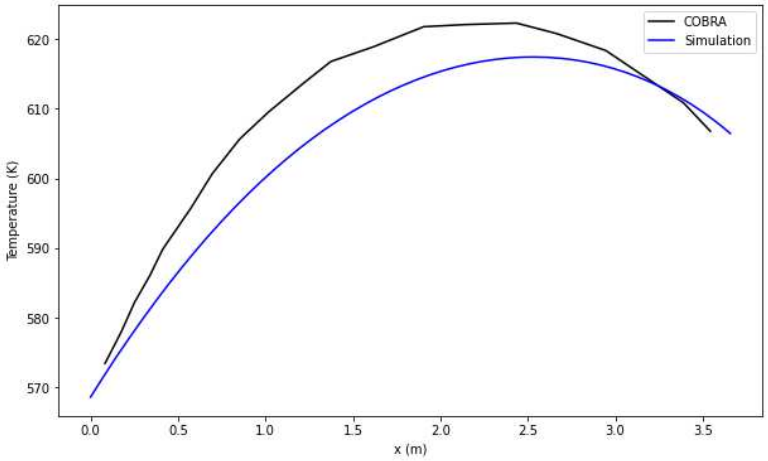


Figure 5.4: Cladding temperature of the fuel assembly

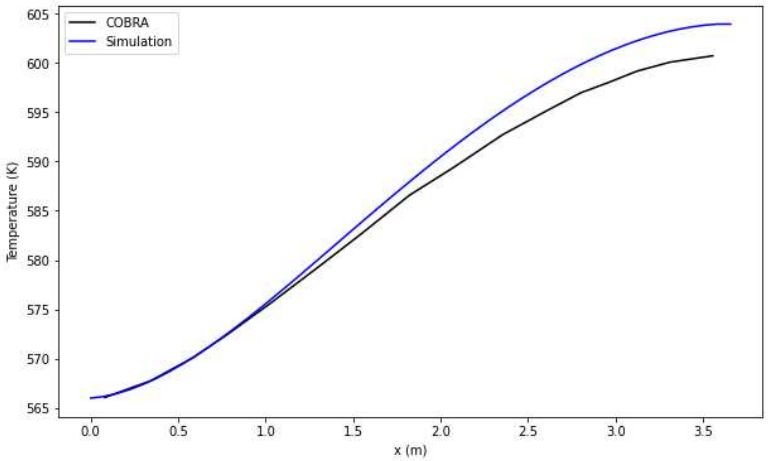


Figure 5.5: Coolant temperature of the fuel assembly

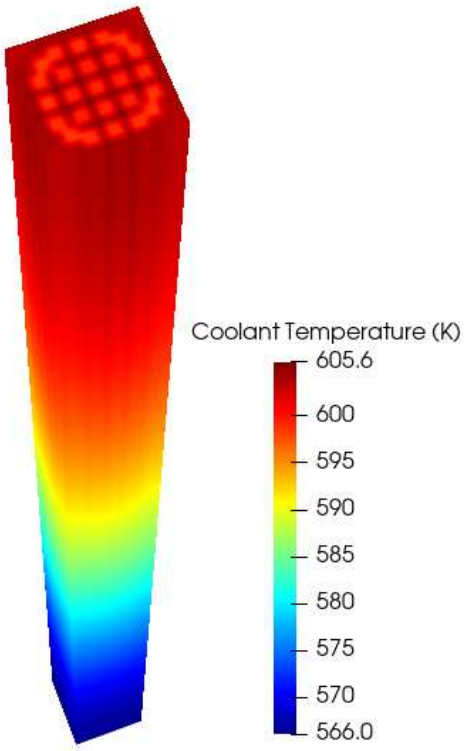


Figure 5.6: 3D plot of the coolant temperature in the fuel assembly

The PWR core was simulated at steady state conditions. It contains 193 full assemblies, using the same geometric parameters as and power distribution as the assembly detailed in Table 5.1. A detailed list of all the geometric parameters and operating conditions are given in Table 5.2. The PWR arrangement and thermal power factor are shown in Figure 5.2. In the computational domain for the fluid solver, a structured Cartesian grid of $270 \times 270 \times 50$ cells was used for subchannel-scale analysis. Each subchannel was coupled to a heat equation to model the heat

transfer. The results are shown in Figures 5.8, 5.9 and 5.10. These are in good qualitative agreement with the numerical results from Liu *et al.* [19]. The maximum fluid temperature of 620 K and the maximum cladding surface temperature of 630 K are in the operational range of a typical PWR core.

Table 5.2: Parameters for reactor core

Parameter	**Value**
Fuel rod parameters	
Fuel rod diameter	$0.0095\,\mathrm{m}$
Fuel pellet diameter	$0.00819\,\mathrm{m}$
Fuel rod length	$3.658\,\mathrm{m}$
Density	$10,970\,\mathrm{kg/m^3}$
Thermal conductivity	$6.0869\,\mathrm{W/(m\,K)}$
Specific heat capacity	$262.43\,\mathrm{J/(kg\,K)}$
Fuel assembly parameters	
Number of fuel rods	264
Number of guide tubes	25
Pitch	$0.0126\,\mathrm{m}$
Cross-sectional area	$0.215\,\mathrm{m} \times 0.215\,\mathrm{m}$
Reactor core parameters	
Number of fuel assemblies	193
Thermal power	$3400\,\mathrm{MW}$
Operation Pressure	$15.5\,\mathrm{MPa}$
Operation Temperature	$566\,\mathrm{K}$
Subchannel T/H models	
Friction loss	$f = 0.316Re^{-0.25}$
Turbulent mixing	$\beta = 0.05$
Gap conductance	$h_{gap} = 20000$

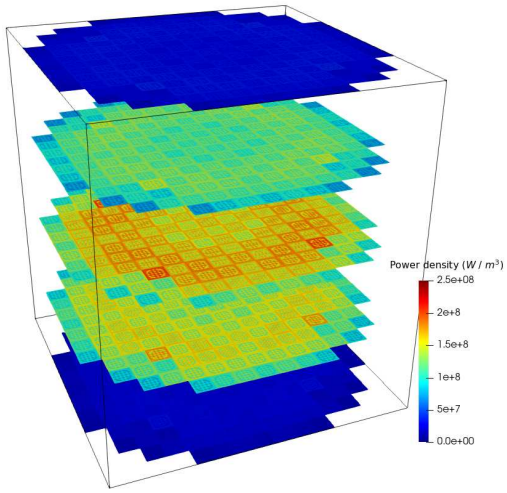


Figure 5.7: The power density in the PWR core

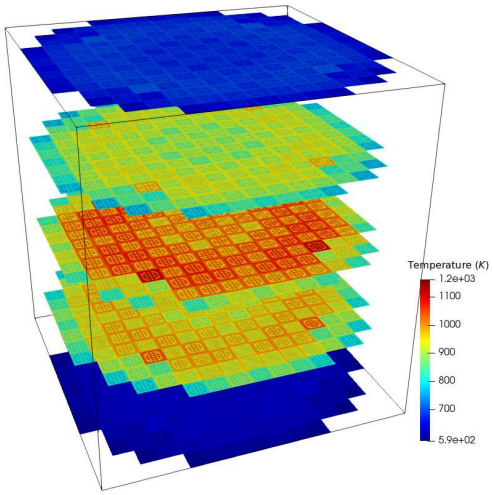


Figure 5.8: The pellet centerline temperature in the PWR core

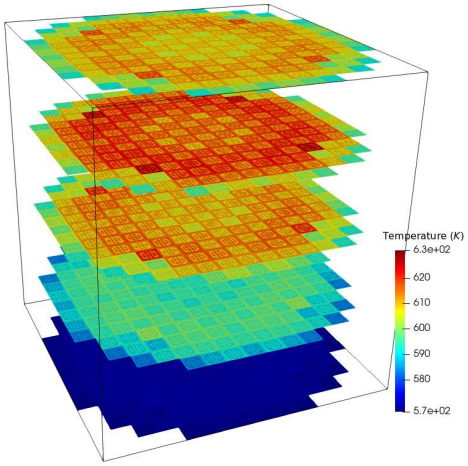


Figure 5.9: The cladding temperature in the PWR core

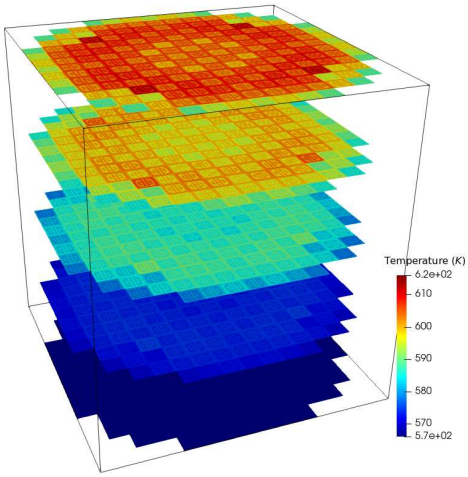


Figure 5.10: The coolant temperature in the PWR core

5.4 HPC cluster parallel

The performance of the solver has been evaluated using different parallel programming frameworks. OpenMP/MPI has been used for CPU parallelism on compute nodes. OpenACC has been used for GPU acceleration on GPU nodes. The hardware for this performance evaluation comes from the Young HPC facility which is hosted by the Materials and Molecular Modelling

Hub. The specifications of the CPU and GPU nodes used for these simulations are shown in Table 5.3. In each simulation, the PWR core test case was run for 1.0 seconds with a fixed number of iterations.

Table 5.3: Young computer nodes used for the performance evaluation

Node type	**CPU**	**Cores per node**	**Memory**	**GPUs per node**
CPU	Intel Xeon CPU Max 9462	64	500 GB	N/A
GPU	2x AMD EPYC 7543 32-Core Processor	64	1 TB	8 A100 40 GB GPUs

5.4.1 CPU nodes - OpenMP/MPI

CPU parallelism was done using an OpenMP/MPI hybrid on a single node. This used 2 OpenMP threads per MPI task. The scaling results are shown below.

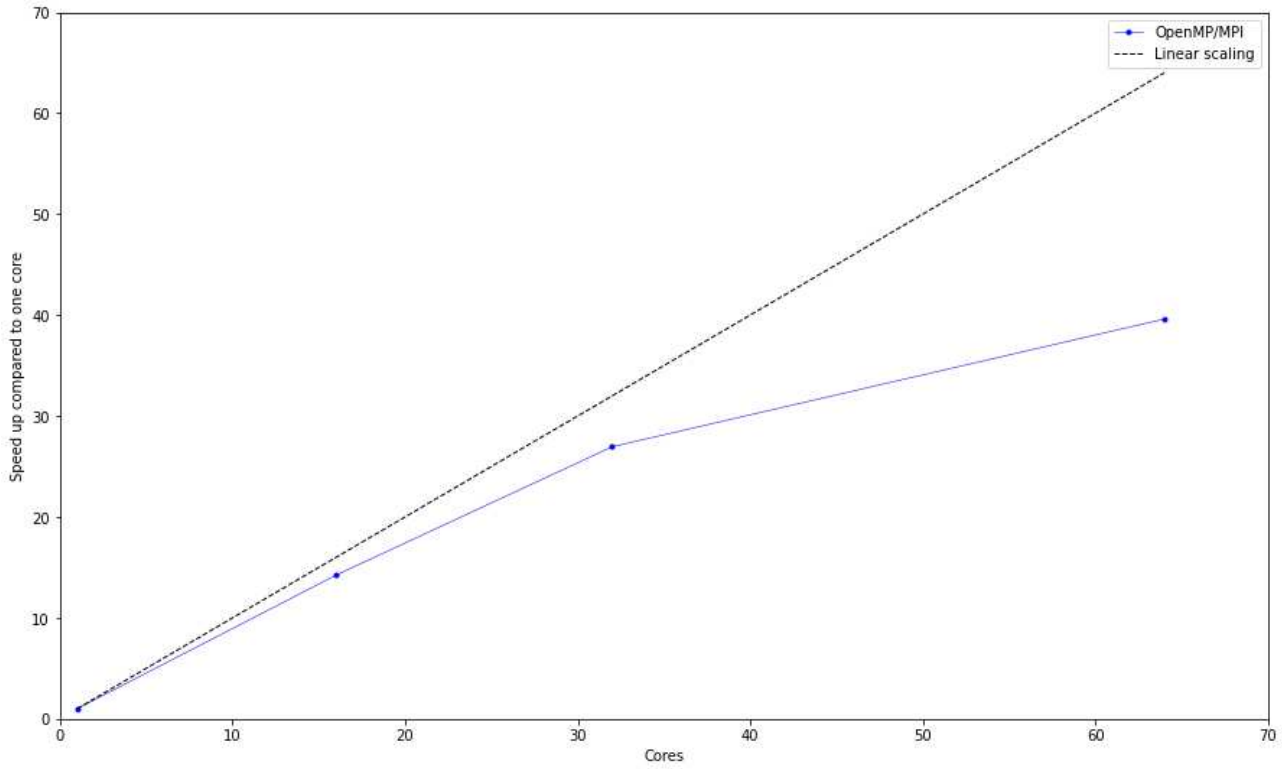


Figure 5.11: The speed up for strong scaling

The serial simulation was completed in 8213 seconds while the OpenMP/MPI hybrid on 64 cores was completed in 206 seconds, which is a speedup of 40. This preliminary test here

demonstrates the implicit solvers performance for a reactor core test case. In the future, more thorough scaling tests can be performed across multiple nodes for a larger problem size.

5.4.2 GPU nodes - OpenACC

GPU acceleration was done using OpenACC. This was performed on a single Nvidia A100 40GB GPU. The simulation was run in 38 seconds, which is a substantial speed up compared to the performance on the CPU node.

5.5 Conclusions

In this chapter a PWR fuel assembly and PWR core have been simulated using pin-scale thermal hydraulics analysis. This uses subchannel models to correlate turbulence, pressure drop and the heat transfer between the fluid and fuel rods. The validation results show good agreement with COBRA and CorTAF, although there is a slight discrepancy in the cladding temperature. The PWR core test case was performed on HPC clusters to evaluate the scaling performance. The scaling tests on CPU nodes demonstrated the efficient parallel execution using OpenMP and MPI. The simulation on a GPU node demonstrates the substantial GPU-acceleration.

Chapter 6

Fluid flow in a reactor pressure vessel

In this chapter, a three-dimensional simulation has been carried out to show the coolant mixing behaviour in a nuclear reactor pressure vessel. This replicates an experimental benchmark that investigated the transport and mixing of a scalar within the Rossendorf Coolant Mixing Model (ROCOM) test facility. The scalar quantity in this experiment is a salt tracer, which represents the boron concentration or the temperature field in a pressurized water reactor (PWR). In the numerical simulation, the reactor core, support plate and perforation drum were represented by a porous medium. The other solid boundaries of the reactor pressure vessel were simulated using a forced term-immersed boundary method. The aim is to demonstrate how to represent complex solid geometries in porous-media codes using a forcing term immersed boundary method.

6.1 Introduction

Boron is used to control reactivity in a nuclear reactor core. The B^{10} isotope of boron has a large neutron cross section, which slows down the rate of fission reactions by absorbing neutrons. In a PWR, this is dissolved in the coolant as boric acid, with the concentration adjusted depending on the operating conditions. During start up, the boric acid concentration is decreased to ramp up the reactivity to criticality. Then, throughout the nuclear fuel cycle, the boron concentration is decreased to increase reactivity, which compensates for the build

up of fission products. Finally, during shutdown boron concentration is increased to lower the reactivity [98].

Understanding boron dilution plays a key role in PWR nuclear safety. The aftermath of a small break loss of coolant accident (SB-LOCA) can cause an inflow of de-borated water into the reactor core. This results in re-criticality and power instabilities. To avert this, it is necessary to understand the dilution of boron during these transient scenarios. This is why large scale facilities have been designed to study these processes [99][100].

Numerical modelling is also necessary to study boron dilution. one-dimensional system codes such as CATHARE [101], RELAP[102] and TRAC-PF1 [103] can be used to simulate solute transport. The advantage of system codes is that they have been extensively validated with experimental data. They are used in licensing applications, operational safety and academic research. This is why system codes remain the industry standard for nuclear safety. However, these are commonly implemented with fully implicit time integration and first order upwind scheme. This results in a high amount of numerical diffusion which reduces the accuracy of the results. Implementing higher order schemes can considerably improve their results for simulating solute transport [103]. Even with this, the governing equations in system codes will not fully capture the physics of three-dimensional flow behaviour. To compensate for this, system codes use broader safety margins to account for the uncertainty of the numerical results.

Improvements in computing power and numerical algorithms has meant that it is now feasible to use computational fluid dynamics codes for large scale problems. These solve the Navier-Stokes equations for single or multi-phase flow. Solute transport and other 3-D mixing phenomena can now be conducted using CFD codes instead of system codes. This has resulted in more CFD validation studies for thermal hydraulics benchmarks. The FLOWMIX-R project [104] is a good example of this. This project was an extensive study to investigate the flow distribution within the primary circuit by conducting tests at several nuclear facilities. Additionally, the experimental data from these tests were used as benchmarks for CFD validation. Several CFD codes were used in these validation studies including ANSYS Fluent[105], ANSYS CFX [106], Trio-U [107] and CUPID [24].

In this work, the porous media-CFD code has been used to simulate a steady-state mixing experiment from the FLOWMIX-R project. This experiment was carried out at the ROCOM test facility. Validation is necessary to assess the capabilities of this code for flow mixing and flow distribution. Then, the CFD methods can be improved for subsequent validation exercises. Section 6.2 describes the ROCOM test facility and the flow mixing experiments. Section 6.3 explains the numerical methods within the CFD solver. Section 6.4 shows the specifications for the numerical simulation, then Section 6.5 shows the numerical results.

6.2 ROCOM Facility

The ROCOM facility is a 1:5 linear and 1:125 volumetric scale model of the KONVOI type PWR reactor [108]. It was built to investigate coolant mixing under single phase conditions. This facility consists of a reactor pressure vessel with four cold legs attached to the inlet nozzles and four hot legs attached to the outlet nozzles 6.5. Each of the cold legs is connected to a pump which controls the flow rate independently. The hot legs are connected to steam generator simulators, which are cylindrical tanks. In the fourth cylindrical tank there is an opening in the tank to equalize the pressure with the atmosphere.

Within the RPV are several structural components:

- A cylindrical barrel to separate the downcomer and the core simulator. The fluid flows from the downcomer into the lower plenum.
- A perforation plate that contains 410 small holes of 15 mm diameter. This balances the fluid flowing through the lower plenum.
- A lower support plate attached underneath the core simulator. This distributes the fluid between 410 channels, each with a diameter of 30 mm.
- A core simulator where the fluid flows through 410 aluminium tubes of 30 mm diameter. The solid resistance replicates the resistance of the fuel rods in an actual nuclear core.

A tracer solution is used to model the expected diffusion of boron or temperature in the test facility. In this case, sodium chloride is injected into the coolant in one of the cold legs. This has

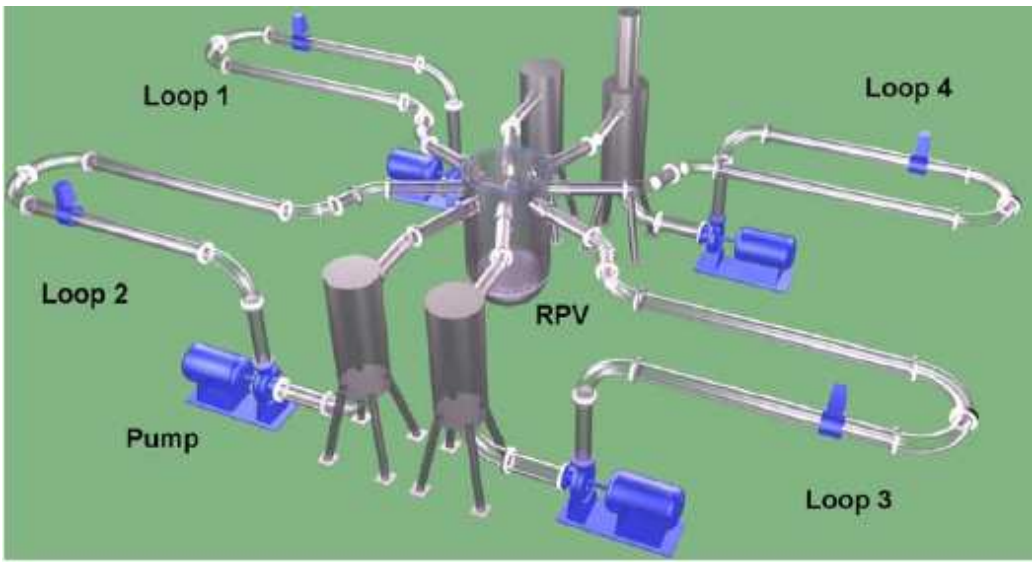


Figure 6.1: Sketch of the ROCOM facility [107] showing the reactor pressure vessel, and the cold and hot legs

little influence on the coolant density such that it can be treated as a passive scalar. The water containing the salt tracer represents water with a low concentration of boron. The other streams with clear water represent water with normal concentrations of boron. The salt concentration was measured using several wire mesh sensors within the facility that measured the conductivity. The salt conductivity can then be normalized using the clear water conductivity at the inlet and the salted water to define a mixing scalar quantity. Using the mixing scalar quantity, the concentration or temperature can be calculated. This requires having reference values of these parameters for the clear and salted water. The equation for this is given below

$$\theta_{x,y,z,t} = \frac{\sigma_{x,y,z,t} - \sigma_0}{\sigma_1 - \sigma_0} = \frac{C_{x,y,z,t} - C_0}{C_1 - C_0} = \frac{T_{x,y,z,t} - T_0}{T_1 - T_0} \tag{6.1}$$

where θ, σ , C and T indicate the mixing scalar quantity, conductivity, concentration of boron and the temperature. The subscripts 1 and 0 indicate the salted and clear water add respectively. Experiments using this setup were conducted in the ROCOM facility as part of the FLOWMIX-R project. These experiments were categorized into three different groups as they investigated different aspects of coolant mixing:

- Steady state mixing experiments

- Transient slug mixing experiments
- Transient buoyancy slug mixing experiments

The steady state mixing experiment ROCOM-stat01 has been used as the test case in this study. In this experiment, the inlet flow rate in all four of the legs was $185\,\mathrm{m}^3/\mathrm{h}$. After some time, the salt tracer was injected at the inlet of one of the legs for 7.0 seconds. This is shown in Figure 6.2.

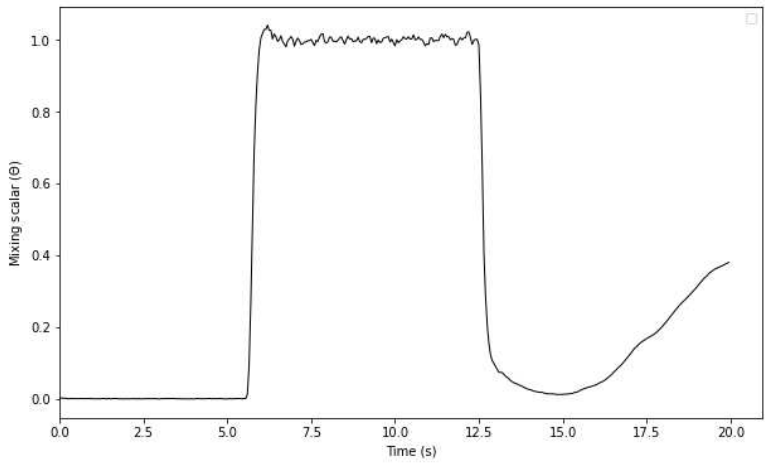


Figure 6.2: Mixing scalar quantity of salt and water in cold leg

6.3 Governing equations and numerical methods

6.3.1 Physics models

The computational fluid dynamics solver for porous-media developed in the previous chapters has been used for these numerical simulations. This is a pressure-based solver for simulating incompressible fluid flow. Finite difference schemes have been used for the discretization, with 2nd order upwind schemes for the advection terms. Then for the time integration, convection terms are treated explicitly while viscous terms are treated implicitly. Turbulence is represented using the mixing length model. It is more common to use two-equation RANS turbulence models such as the standard K-ϵ, RNG K-ϵ, and SST models [109], [110]. However, the aim of this study is to show qualitative results, for which a simple turbulence model is sufficient.

The salt tracer is represented using an advection-diffusion equation, where the mass diffusivity value has been taken from another numerical study of the ROCOM experiments [104]. With that said, the turbulent coolant mixing should be much more significant than the molecular diffusion, given how small the mass diffusivity is. The exact value of the mass diffusivity should have a minimal influence on the numerical results.

6.3.2 Geometry simplification

The original mesh of the ROCOM test facility is shown in Figure 6.3. This was simplified within the computational domain to speed up the simulation time.

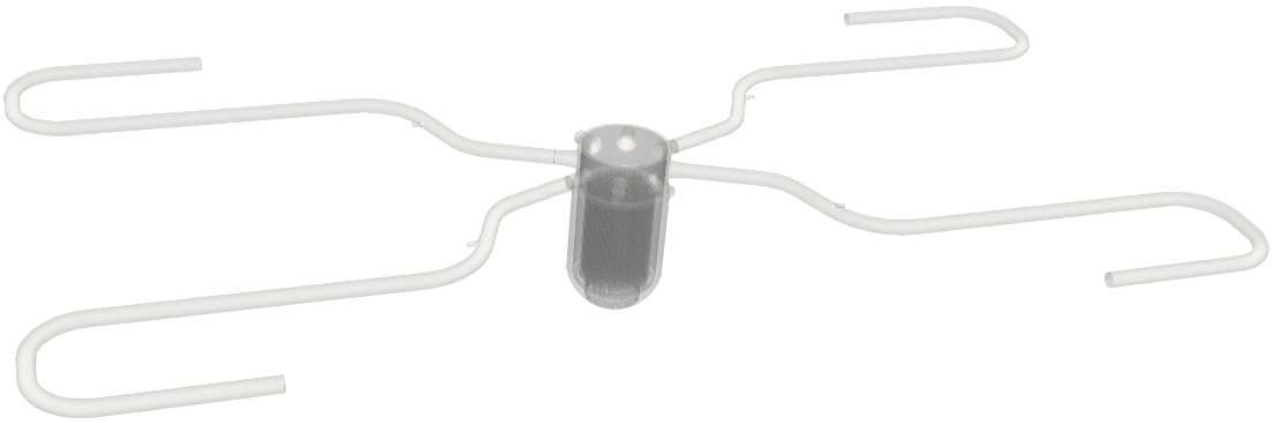


Figure 6.3: Original CAD geometry of the ROCOM test facility

First, the clog legs that were attached to the inlet nozzles were removed from the geometry. Next, the reactor core simulator, lower support plate and perforation drum were represented as a porous medium rather than the original solid structures. These solid structures can have significant effects on the behaviour of the fluid. To replicate these effects, loss coefficients were modelled in these sections to add drag resistance.

$$\frac{\partial P}{\partial x} = K_{loss}\frac{\rho}{2}|u|u \tag{6.2}$$

The reactor core simulator has a porosity of ≈ 0.229. Its loss coefficient can be calculated using an appropriate correlation for frictional drag in turbulent flows. In this case, the Blasius

correlation has been used:

$$K_{loss} = \frac{f}{D_h} = 0.316 Re^{-0.25} \,. \tag{6.3}$$

The support plate also has a porosity of ≈ 0.229, while its loss coefficient in the axial direction is $85\,\mathrm{m}^{-1}$. The perforation drum has a porosity of ≈ 0.209, while its loss coefficient in the axial direction is $13.1\,\mathrm{m}^{-1}$. These loss coefficients have been taken from the numerical study of Boumaza *et al.* [110] for the ROCOM-stat02 test case. The simplified reactor pressure vessel is shown in Figure 6.4.



Figure 6.4: Simplified CAD geometry of the ROCOM test facility

6.3.3 Grid generation and immersed boundary methods

Structured Cartesian grids are the simplest and easiest computational grids to generate. The equal node spacing makes them suitable for finite difference schemes. However, unstructured grids are favoured for representing complex geometries because a body-conforming mesh can be used. This allows the grid resolution to be refined in the specific sections where the boundaries are important. A common alternative to this is immersed boundary methods [111, 112]. These

are commonly used in finite difference schemes for mapping curvilinear solid boundaries onto structured grids.

There are many implementations of immersed boundary methods. Caunt *et al.* [113]'s study shows an immersed boundary method that has been implemented into Devito. In this method, the solid regions are represented as sign distance functions (SDF). Then, the finite-difference stencils are modified in the vicinity of the sign distance functions that are in the solid regions. This method is adaptable to different partial differential equations and discretization schemes. It has been used for geophysics problems where the numerical solver uses the 2nd order acoustic wave equation.

In this study, SDFs are also used to represent the solid regions. However, instead of using stencil modification, a direct forcing term immersed boundary has been used [114–116]. In direct forcing term immersed boundaries, a source term in the momentum equation represents the solid regions. If there is no solid boundary the source term is absent. If there is a solid boundary the source term exerts a force on the fluid. In the method presented here, the source term is determined by the SDF as shown in Equation (6.4). If the SDF is greater than zero, the source term is equal to zero. Otherwise the drag term is enforced which reduces the velocity to 0.

$$S_{solid\,region} = (1 - max(0, sign(SDF))) \times K_{factor} \tag{6.4}$$

$S_{solid\,region}$ is the source term in the momentum equation, max is the maximum function, $sign$ is the sign function and K_{factor} is a factor that ensures the source term is sufficiently large. Compared to the stencil modification method, the present method is a lower-order approximation, which leads to stair case boundaries. However, it is simple to implement and adaptable to many boundary conditions. It is also consistent with the source term discretization shown in Chapter 3. The immersed boundary source term can be discretized using the same formulation as other general source terms.

The SDF of the RPV was generated from an STL file of the modified geometry shown in

Figure 6.4. This was done using the python package Mesh2SDF [117]. The SDF used for the simulations is shown below.

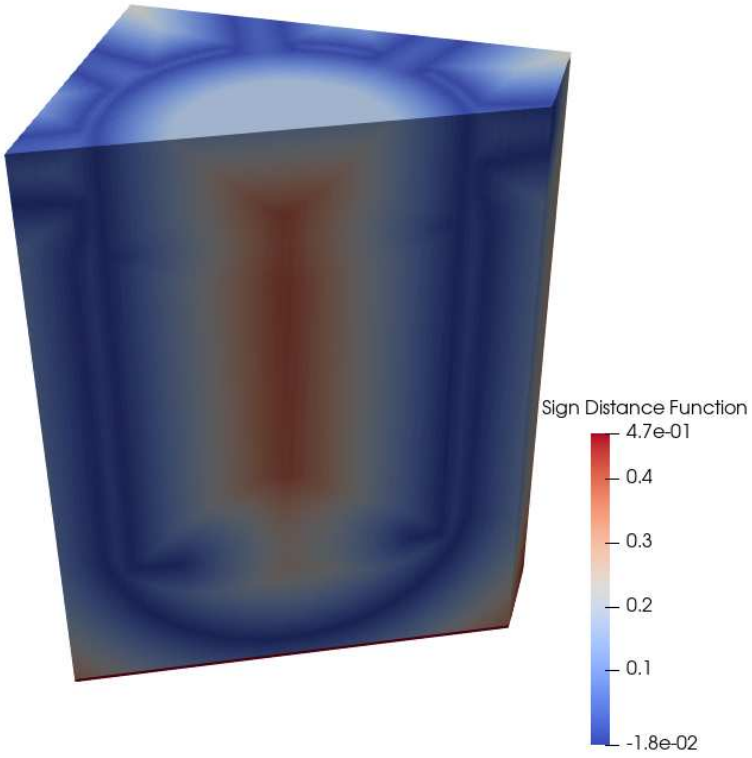


Figure 6.5: SDF of the reactor pressure vessel

6.3.4 Boundary conditions

A constant pressure of 1 atm and temperature of 273 K are specified to obtain the fluid properties. These fluid properties remain constant in the solution procedure to simplify the numerical methods. In the four inlets, the mass flow rates of 185 m^3/h, correspond to inlet velocities of 2.91 m/s.

6.4 Simulation Setup

Mahaffy *et al.* [118]'s study explains the best practices for applying CFD to nuclear engineering problems. A comprehensive numerical study requires understanding the limitations of the

physics models and the numerical methods. In the present work, the notable modelling errors include the porous media approximation for the internal components, the turbulence model and the physical properties of the fluid. Then there are the numerical errors in the grid refinement, discretization schemes, convergence criteria and the time step size.

A more detailed study can be conducted in the future to properly validate the CFD solver. For now, a single simulation has been performed to test the immersed boundary implementation. The fluid properties and physics models have been summarised in Table 6.1. Table 6.2 shows the numerical methods for this simulation.

Table 6.1: Parameters for ROCOM experiment

Parameter	**Value**
Operating conditions	
Inlet velocity	$2.91\,\mathrm{m/s}$
Operation Pressure	$1\,\mathrm{atm}$
Operation Temperature	$298\,\mathrm{K}$
Fluid properties	
Density	$997\,\mathrm{kg/m^3}$
Dynamic viscosity	$8.899 \times 10^{-4}\,\mathrm{kg/(m\,s)}$
Tracer Mass Diffusivity	$2.88 \times 10^{-5}\,\mathrm{m^2/s}$
Physics models	
Porosity and Loss coefficient:	
• Core	$\epsilon = 0.229$ and $K_{loss} = 0.316Re^{-0.25}$
• Lower support plate	$\epsilon = 0.229$ and $K_{loss} = 85$
• Perforation drum	$\epsilon = 0.209$ and $K_{loss} = 13.1$
Turbulence	Mixing length model

Table 6.2: Numerical methods

Parameter	**Value**
Solver	Transient
Spatial Discretization	2nd order upwind
Time integration	IMEX
Courant number	0.1
Simulation time	$20\,\mathrm{s}$
Grid size	1946880

6.5 Numerical Results

Figure 6.6 shows a snapshot of the fluid velocity in the reactor pressure vessel. The fluid has flowed through the downcomer, then into the plenum where there is a significant amount of turbulence. Qualitatively, this shows the expected mixing behaviour for the STAT01 experiment. But, as the simulation progressed, numerical instabilities prevented a stable solution being reached. So, it was not possible to produce quantitative comparisons with the experimental benchmarks.

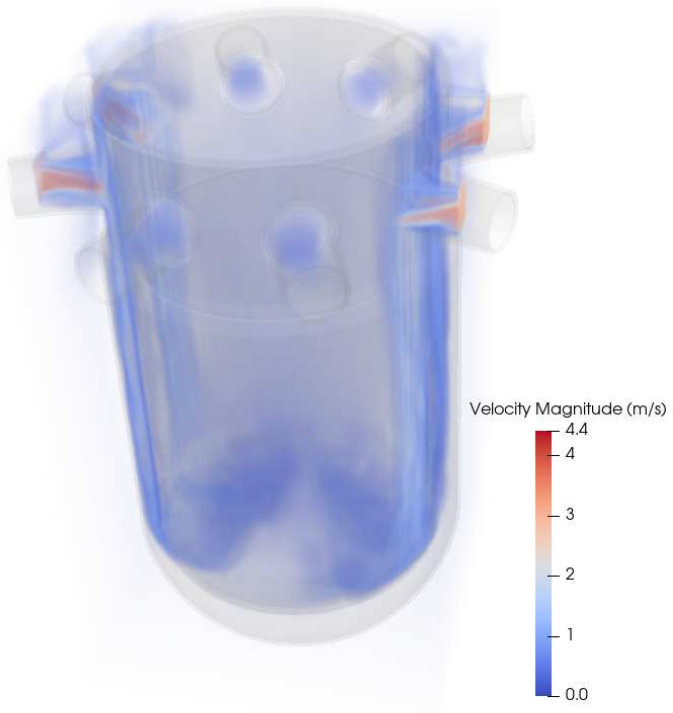


Figure 6.6: Velocity distribution in the simulations results

The turbulent mixing of the fluid dictates the mixing of the transport scalar. The main objective of this experiment was to understand how the turbulent mixing affects the transport scalar. A tracer was injected for 7.0 seconds as shown in Figure 6.2. In the numerical simulation, the tracer was also injected for 7.0 seconds as a boundary condition on one of the inlet nozzles. Figure 6.7 overlays the tracer concentration at the inlet in the experiment with the numerical model. In the experiment, the tracer concentration undulates during the 7.0 second injection

period, then it begins to rise afterwards. In the numerical model, a constant tracer concentration is used for the whole injection period.

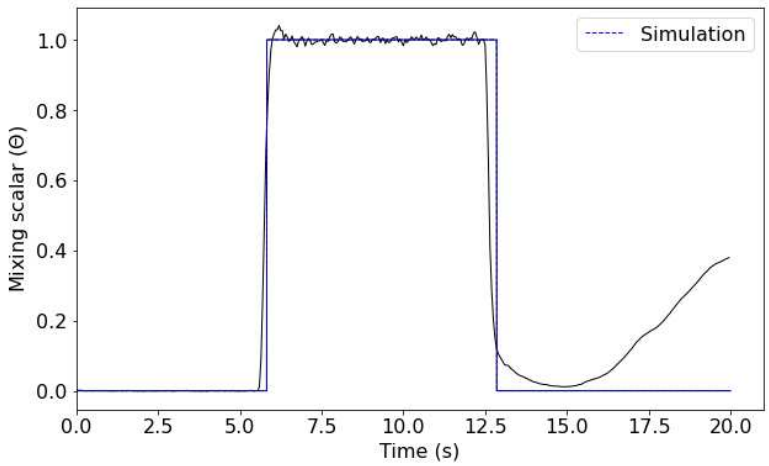


Figure 6.7: The tracer concentration at the inlet in the experiment and the numerical model

Figure 6.8 shows a snapshot of the solute tracer distribution. The tracer is carried by the fluid through the downcomer into the plenum. But, the numerical instabilties in the velocity affected the advection-diffusion equation. So, the validation against the experimental tracer concentration could not be performed.

These results have demonstrated the forcing term immersed boundary method using Sign Distance Functions. This allows an STL file for a solid object to be represented as source terms in the momentum equation. But, more development of the physics model and numerical methods are needed to complete the validation. A full grid convergence study can be performed along with modifications to the solution procedure and turbulence models.

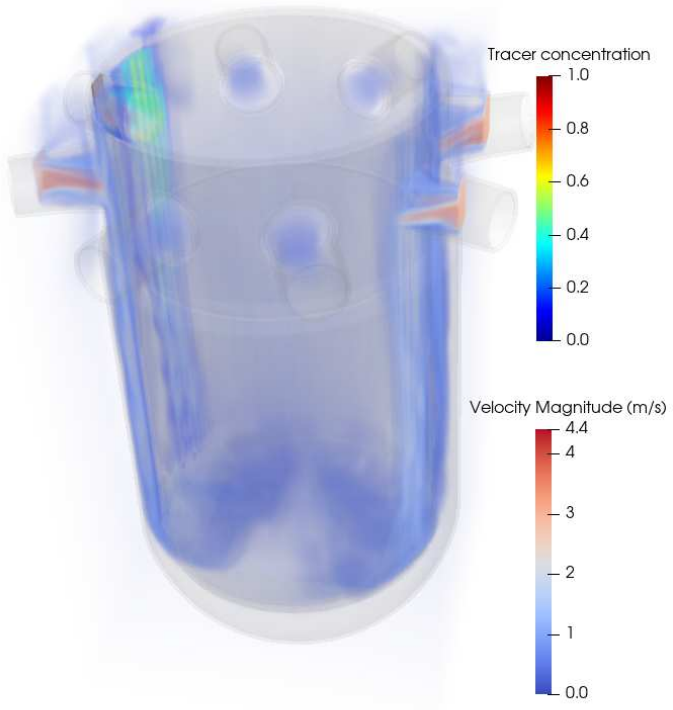


Figure 6.8: Tracer distribution in the simulation results

6.6 Conclusions

A forcing term immersed boundary has been developed using Sign Distance Functions. This is able to recreate fluid-solid interactions using source terms in the momentum equation. In this test case, the ROCOM facility was represented in the numerical model. This was to replicate the STAT01 steady state mixing experiment, which tests the turbulent mixing of a transport scalar. The results showed the flow behaviour in the downcomer and the turbulent mixing in the plenum. But, numerical instabilities formed in the solution, which prevented a steady solution being reached.

Subsequent work would be to do a more detailed study of this test case using different methods. For example, two-equation turbulence models were commonly used for the production level results from Rohde [104]. These models could be implemented into this code. Additionally, the scaling results from Chapter 5 show how simulations can be performed on multi-CPU/GPU clusters. This makes it feasible to conduct finer resolution 3D simulations in a reasonable time.

Chapter 7

Conclusions

This chapter explains the conclusions drawn from this research project. The four main outcomes from this PhD thesis are summarised in the first section. Then, an overview is given on possible applications of this CFD solver for future projects. Finally, an explanation is given on areas that could be improved. This includes improvements for the current CFD code and also ways it could be extended.

7.1 Summary of Thesis Achievements

The first outcome from this thesis was the development of an accurate formulation for the volume-averaged Navier-Stokes equations. This represents the macroscopic fluid flow through a porous medium. The motivation for developing this formulation is to simulate sharp discontinuities in porosity and permeability with no numerical oscillations. This is applicable to Darcy flows where the inertial forces are very small.

To verify this numerical model, the numerical results were compared to analytical solutions for porous media flows. The first numerical experiment compared different derivations of the volume-averaged momentum equation. This showed that the convective term can have a bigger influence on the numerical errors than the Rhie-Chow interpolation error. The semi-heuristic

momentum equation from Kaviany [73] produced the least oscillations. The next test cases verified the explicit and implicit formulations for discretizating the drag terms. Both formulations showed no oscillations across extreme discontinuities in porosity and permeability.

The second outcome was the implementation and validation of subchannel models for nuclear thermal hydraulics analysis. These are single-phase models that represent the fluid flow in nuclear fuel assemblies. The empirical correlations for heat transfer, pressure drop and turbulence were implemented into the porous-media CFD code. This was validated against subchannel codes and experimental benchmarks, where the results were in good agreement with each other.

The third outcome was showing the parallel performance of this solver on modern hardware architectures. This solver is compatible with OpenMP, MPI and OpenACC for CPU parallelism and GPU acceleration. This was demonstrated with the performance tests where the parallel simulations caused a substantial speed up compared to the serial simulations.

The fourth outcome was to simulate the fluid flow in a reactor pressure vessel. A common limitation of finite difference schemes is that they can be unsuitable for curvilinear geometries. This limits the applications to simple geometries. In this work, a forcing term immersed boundary method was developed. This was used to represent the geometry of the ROCOM facility. Numerical simulations of this facility showed the fluid flow in the downcomer and the turbulence in the lower plenum. But, more validation is needed to fully validate the turbulent mixing in the ROCOM steady state experiment.

7.2 Applications

The motivation for developing this numerical solver is to use it for nuclear thermal hydraulic analysis. This solver can used subchannel scale simulations of a fuel assembly, a reactor core or steam generator. It can also be used for resolved CFD simulations of the turbulent mixing in the plenum or the directed flow in mixing vanes. In this work, the focus has been on pressurized water reactors. However, porous-media CFD codes have also been used for boiling water reactors, CANDU reactors, and sodium-cooled fast reactors. The developments from other

porous-media codes such as GenFOAM, CorTAF, CUPID, TWOPORFLOW and PORFLO demonstrates the flexibility of this modelling approach for a variety of nuclear reactor designs.

Additionally, this code could potentially be used for system-scale analysis. The Devito framework allows parallel computations to be performed efficiently on HPC clusters. This makes it feasible to perform system-scale simulations. It avoids the need for complex coupling strategies between system codes and porous media CFD codes.

Porous media CFD solvers also have applications outside nuclear engineering. Porous media modelling is used to simulate fuel cells, heat exchangers, catalytic converters and other unit operations. The relevant models for these applications can be integrated into the solver developed in this thesis.

7.3 Future Work

There are many different ways this code could be improved. Firstly, an important part of the CFD solver is the iterative solver for solving the pressure equation. Currently, the pressure equation is solver using a Jacobi solver. This is easy to implement and easy to scale in parallel. But, the convergence is slow and it is normally used as a preconditioner for more efficient iterative methods. As shown in Chapter 2, it is possible to implement iterative loops within Devito. A suitable project could be to develop efficient iterative methods within Devito. Additionally, there is ongoing work to integrate the PETSc library within Devito. PETSc is a toolkit that contains numerous iterative solvers.

The physics modelling could also be extended in multiple areas. Two-phase flows occur in many nuclear reactors. Implementing a two-phase flow model and suitable multiphase empirical correlations will substantially increase the scope of nuclear applications. There are also many options for implementing turbulence models. A mixing length model has been used for the work in this thesis. But, two-equation RANS models have more generality and are more suitable for 3D thermal hydraulics simulations.

Appendix A

Supplementary code from Chapter 2

A.1 Python code for Burgers' equation

```
import devito as dv

width = 1
nx = 41
ny = 41

grid = dv.Grid(shape=(nx, ny), extent=(width, width))

# Constants
nu = .1

# Variables are automatically initalized at 0.
u = dv.TimeFunction(name='u', grid=grid, space_order=2)
v = dv.TimeFunction(name='v', grid=grid, space_order=2)

# Burgers equation in 2 dimensions
eq_u = dv.Eq(u.dt + u*u.dx + v*u.dy,  nu*(u.laplace),  subdomain=grid.interior)
eq_v = dv.Eq(v.dt + u*v.dx + v*v.dy,  nu*(v.laplace),  subdomain=grid.interior)

# Let SymPy rearrange our stencils to form the update expressions
stencil_u = dv.solve(eq_u , u.forward)
stencil_v = dv.solve(eq_v , v.forward)

update_u = dv.Eq(u.forward, stencil_u)
update_v = dv.Eq(v.forward, stencil_v)

t = grid.stepping_dim
x, y = grid.dimensions
# Create Dirichlet BC expressions using the low-level API
bc_u  = [dv.Eq(u[t+1, 0, y], 0)]
bc_u += [dv.Eq(u[t+1, nx-1, y], 0)]
bc_u += [dv.Eq(u[t+1, x, 0], 0)]
```

```
bc_u += [dv.Eq(u[t+1, x, ny-1], 1)]  # u=1 for y=2
bc_v  = [dv.Eq(v[t+1, 0, y], 0)]
bc_v += [dv.Eq(v[t+1, nx-1, y], 0)]
bc_v += [dv.Eq(v[t+1, x, ny-1], 0)]
bc_v += [dv.Eq(v[t+1, x, 0], 0)]


# Time stepping dimensions
nt = 1000
delta_t = .001

op = dv.Operator([update_u, update_v] + bc_u + bc_v )
op(time_m=0, time_M=nt, dt=delta_t)
```

A.2 C code for Burgers' equation

```
#define _POSIX_C_SOURCE 200809L
#define START(S) struct timeval start_ ## S , end_ ## S ; gettimeofday(&start_ ## S ,
↪  NULL);
#define STOP(S,T) gettimeofday(&end_ ## S, NULL); T->S += (double)(end_ ## S .tv_sec-start_
↪  ## S.tv_sec)+(double)(end_ ## S .tv_usec-start_ ## S .tv_usec)/1000000;

#include "stdlib.h"
#include "math.h"
#include "sys/time.h"
#include "xmmintrin.h"
#include "pmmintrin.h"

struct dataobj
{
  void *restrict data;
  unsigned long * size;
  unsigned long * npsize;
  unsigned long * dsize;
  int * hsize;
  int * hofs;
  int * oofs;
  void * dmap;
} ;

struct profiler
{
  double section0;
  double section1;
  double section2;
} ;


int Kernel(struct dataobj *restrict u_vec, struct dataobj *restrict v_vec, const float dt,
↪  const float h_x, const float h_y, const int time_M, const int time_m, const int x_M,
↪  const int x_m, const int y_M, const int y_m, struct profiler * timers)
{
  float (*restrict u)[u_vec->size[1]][u_vec->size[2]] __attribute__ ((aligned (64))) =
  ↪  (float (*)[u_vec->size[1]][u_vec->size[2]]) u_vec->data;
```

```
float (*restrict v)[v_vec->size[1]][v_vec->size[2]] __attribute__ ((aligned (64))) =
↪  (float (*)[v_vec->size[1]][v_vec->size[2]]) v_vec->data;

/* Flush denormal numbers to zero in hardware */
_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);

float r0 = 1.0F/(h_x*h_x);
float r1 = 1.0F/(h_y*h_y);
float r2 = 1.0F/dt;
float r3 = 1.0F/h_x;
float r4 = 1.0F/h_y;

for (int time = time_m, t0 = (time)%(2), t1 = (time + 1)%(2); time <= time_M; time += 1,
↪  t0 = (time)%(2), t1 = (time + 1)%(2))
{
  START(section0)
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(u,v:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      u[t1][x + 2][y + 2] = dt*(r2*u[t0][x + 2][y + 2] + (-2.0e-1F)*(r0*u[t0][x + 2][y +
      ↪  2] + r1*u[t0][x + 2][y + 2]) - (r3*(-u[t0][x + 2][y + 2]) + r3*u[t0][x + 3][y +
      ↪  2])*u[t0][x + 2][y + 2] - (r4*(-u[t0][x + 2][y + 2]) + r4*u[t0][x + 2][y +
      ↪  3])*v[t0][x + 2][y + 2] + 1.0e-1F*(r0*u[t0][x + 1][y + 2] + r0*u[t0][x + 3][y +
      ↪  2] + r1*u[t0][x + 2][y + 1] + r1*u[t0][x + 2][y + 3]));
      v[t1][x + 2][y + 2] = dt*(r2*v[t0][x + 2][y + 2] + (-2.0e-1F)*(r0*v[t0][x + 2][y +
      ↪  2] + r1*v[t0][x + 2][y + 2]) - (r3*(-v[t0][x + 2][y + 2]) + r3*v[t0][x + 3][y +
      ↪  2])*u[t0][x + 2][y + 2] - (r4*(-v[t0][x + 2][y + 2]) + r4*v[t0][x + 2][y +
      ↪  3])*v[t0][x + 2][y + 2] + 1.0e-1F*(r0*v[t0][x + 1][y + 2] + r0*v[t0][x + 3][y +
      ↪  2] + r1*v[t0][x + 2][y + 1] + r1*v[t0][x + 2][y + 3]));
    }
  }
  STOP(section0,timers)

  START(section1)
  for (int y = y_m; y <= y_M; y += 1)
  {
    u[t1][2][y + 2] = 0;
    u[t1][42][y + 2] = 0;
    v[t1][2][y + 2] = 0;
    v[t1][42][y + 2] = 0;
  }
  STOP(section1,timers)

  START(section2)
  for (int x = x_m; x <= x_M; x += 1)
  {
    u[t1][x + 2][2] = 0;
    u[t1][x + 2][42] = 1;
    v[t1][x + 2][42] = 0;
    v[t1][x + 2][2] = 0;
  }
  STOP(section2,timers)
}

return 0;
```

```
}
```

A.3 Python code for the incompressible Navier-Stokes equations

```
import devito as dv

width = 1
nx = 41
ny = 41

grid = dv.Grid(shape=(nx, ny), extent=(width, width))

#Constants
nu = .1

#Variables are automatically initalized at 0.
u = dv.TimeFunction(name='u', grid=grid, space_order=2)
v = dv.TimeFunction(name='v', grid=grid, space_order=2)

# Time stepping dimensions
time = grid.time_dim

# Iterative stepping dimensions (within each time step)
i = dv.TimeDimension(name='i', parent=time)

# Iterative stepping dimensions (within each time step)
i_1 = dv.SteppingDimension(name='i_1', parent=i)

#Variable p, defined for the iterative dimension, is initalized at 0.
p = dv.TimeFunction(name='p',  grid=grid, time_dim=i_1, space_order=2)

# First order derivatives will be handled with p.dxc
eq_u = dv.Eq((u.dt + u*u.dx + v*u.dy), -p.dxc + nu*(u.laplace),  subdomain=grid.interior)
eq_v = dv.Eq((v.dt + u*v.dx + v*v.dy), -p.dyc + nu*(v.laplace),  subdomain=grid.interior)

delta_t = .001
eq_p = dv.Eq(p.laplace,
↪  1./delta_t*(u.dxc+v.dyc)-(u.dxc*u.dxc)-2*(u.dyc*v.dxc)-(v.dyc*v.dyc),
↪  subdomain=grid.interior)

# NOTE: Pressure has no time dependence so we solve for the other pressure buffer.
stencil_u = dv.solve(eq_u , u.forward)
stencil_v = dv.solve(eq_v , v.forward)
stencil_p = dv.solve(eq_p, p)

update_u = dv.Eq(u.forward, stencil_u)
update_v = dv.Eq(v.forward, stencil_v)

idims_i = (time, i,) + tuple(grid.dimensions)
update_p = dv.Eq(p.forward, stencil_p, implicit_dims=idims_i,)
```

```
t = grid.stepping_dim
x, y = grid.dimensions
# Boundary Conds. u=v=0 for all sides
bc_u  = [dv.Eq(u[t+1, 0, y], 0)]
bc_u += [dv.Eq(u[t+1, nx-1, y], 0)]
bc_u += [dv.Eq(u[t+1, x, 0], 0)]
bc_u += [dv.Eq(u[t+1, x, ny-1], 1)]  # except u=1 for y=2
bc_v  = [dv.Eq(v[t+1, 0, y], 0)]
bc_v += [dv.Eq(v[t+1, nx-1, y], 0)]
bc_v += [dv.Eq(v[t+1, x, ny-1], 0)]
bc_v += [dv.Eq(v[t+1, x, 0], 0)]

bc_p  = [dv.Eq(p[i_1+1, 0, y],p[i_1, 1,y], implicit_dims=idims_i,)]  # dpn/dx = 0 for x=0.
bc_p += [dv.Eq(p[i_1+1,nx-1, y],p[i_1,nx-2, y], implicit_dims=idims_i,)]  # dpn/dx = 0 for
↪  x=2.
bc_p += [dv.Eq(p[i_1+1, x, 0],p[i_1,x ,1], implicit_dims=idims_i,)]  # dpn/dy = 0 at y=0
bc_p += [dv.Eq(p[i_1+1, x, ny-1],p[i_1, x, ny-2], implicit_dims=idims_i,)]  # pn=0 for y=2
bc_p += [dv.Eq(p[i_1+1, 0, 0], 0, implicit_dims=idims_i,)]

nt = 1000
nit = 50
op = dv.Operator([update_p] + bc_p + [update_u, update_v] + bc_u + bc_v )
op(time_m=0, time_M=nt-1, i_m=0, i_M = nit, dt=delta_t)
```

A.4 C code for the incompressible Navier-Stokes equations

```
#define _POSIX_C_SOURCE 200809L
#define START(S) struct timeval start_ ## S , end_ ## S ; gettimeofday(&start_ ## S ,
↪  NULL);
#define STOP(S,T) gettimeofday(&end_ ## S, NULL); T->S += (double)(end_ ## S .tv_sec-start_
↪  ## S.tv_sec)+(double)(end_ ## S .tv_usec-start_ ## S .tv_usec)/1000000;

#include "stdlib.h"
#include "math.h"
#include "sys/time.h"
#include "xmmintrin.h"
#include "pmmintrin.h"

struct dataobj
{
  void *restrict data;
  unsigned long * size;
  unsigned long * npsize;
  unsigned long * dsize;
  int * hsize;
  int * hofs;
  int * oofs;
  void * dmap;
} ;
```

```
struct profiler
{
  double section0;
  double section1;
  double section2;
  double section3;
  double section4;
} ;


int Kernel(struct dataobj *restrict p_vec, struct dataobj *restrict u_vec, struct dataobj
↪  *restrict v_vec, const float dt, const float h_x, const float h_y, const int i_M, const
↪  int i_m, const int time_M, const int time_m, const int x_M, const int x_m, const int
↪  y_M, const int y_m, struct profiler * timers)
{
  float (*restrict p)[p_vec->size[1]][p_vec->size[2]] __attribute__ ((aligned (64))) =
  ↪  (float (*)[p_vec->size[1]][p_vec->size[2]]) p_vec->data;
  float (*restrict u)[u_vec->size[1]][u_vec->size[2]] __attribute__ ((aligned (64))) =
  ↪  (float (*)[u_vec->size[1]][u_vec->size[2]]) u_vec->data;
  float (*restrict v)[v_vec->size[1]][v_vec->size[2]] __attribute__ ((aligned (64))) =
  ↪  (float (*)[v_vec->size[1]][v_vec->size[2]]) v_vec->data;

  /* Flush denormal numbers to zero in hardware */
  _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
  _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);

  float r6 = h_x*h_x;
  float r7 = h_y*h_y;
  float r0 = 1.0F/(r6 + r7);
  float r1 = 1.0F/h_x;
  float r2 = 1.0F/r6;
  float r3 = 1.0F/r7;
  float r4 = 1.0F/dt;
  float r5 = 1.0F/h_y;

  for (int time = time_m, t0 = (time)%(2), t1 = (time + 1)%(2); time <= time_M; time += 1,
  ↪  t0 = (time)%(2), t1 = (time + 1)%(2))
  {
    for (int i = i_m, i_10 = (i)%(2), i_11 = (i + 1)%(2); i <= i_M; i += 1, i_10 = (i)%(2),
    ↪  i_11 = (i + 1)%(2))
    {
      START(section0)
      for (int x = x_m; x <= x_M; x += 1)
      {
        #pragma omp simd aligned(p,u,v:32)
        for (int y = y_m; y <= y_M; y += 1)
        {
          float r8 = h_y*h_y;
          float r9 = h_x*h_x;
```

```
      p[i_11][x + 2][y + 2] = -2.5e+2F*r0*((-5.0e-4F*r8)*(u[t0][x + 1][y + 2]*u[t0][x +
      ↪  1][y + 2]) + r8*(-1.0F*h_x*u[t0][x + 1][y + 2] + h_x*u[t0][x + 3][y + 2] +
      ↪  (-5.0e-4F)*(u[t0][x + 3][y + 2]*u[t0][x + 3][y + 2]) + (-2.0e-3F)*(p[i_10][x +
      ↪  1][y + 2] + p[i_10][x + 3][y + 2]) + 1.0e-3F*u[t0][x + 1][y + 2]*u[t0][x +
      ↪  3][y + 2]) + (-5.0e-4F*r9)*(v[t0][x + 2][y + 1]*v[t0][x + 2][y + 1]) +
      ↪  r9*(-1.0F*h_y*v[t0][x + 2][y + 1] + h_y*v[t0][x + 2][y + 3] +
      ↪  (-5.0e-4F)*(v[t0][x + 2][y + 3]*v[t0][x + 2][y + 3]) + (-2.0e-3F)*(p[i_10][x +
      ↪  2][y + 1] + p[i_10][x + 2][y + 3]) + 1.0e-3F*v[t0][x + 2][y + 1]*v[t0][x +
      ↪  2][y + 3]) + h_x*(1.0e-3F*(-h_y*u[t0][x + 2][y + 1]*v[t0][x + 1][y + 2] +
      ↪  h_y*u[t0][x + 2][y + 1]*v[t0][x + 3][y + 2] + h_y*u[t0][x + 2][y + 3]*v[t0][x
      ↪  + 1][y + 2] - h_y*u[t0][x + 2][y + 3]*v[t0][x + 3][y + 2])));
    }
  }
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(p:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      p[i_11][2][y + 2] = p[i_10][3][y + 2];
    }
  }
  STOP(section0,timers)

  START(section1)
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(p:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      p[i_11][42][y + 2] = p[i_10][41][y + 2];
    }
  }
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(p:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      p[i_11][x + 2][2] = p[i_10][x + 2][3];
    }
  }
  STOP(section1,timers)

  START(section2)
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(p:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      p[i_11][x + 2][42] = p[i_10][x + 2][41];
    }
  }
  for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(p,u,v:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      p[i_11][2][2] = 0;
```

```
          u[t1][x + 2][y + 2] = dt*(-5.0e-1F*r1*(-p[i_10][x + 1][y + 2] + p[i_10][x + 3][y +
          ↪  2]) + r4*u[t0][x + 2][y + 2] - (r1*(-u[t0][x + 2][y + 2]) + r1*u[t0][x + 3][y
          ↪  + 2])*u[t0][x + 2][y + 2] + (-2.0e-1F)*(r2*u[t0][x + 2][y + 2] + r3*u[t0][x +
          ↪  2][y + 2]) - (r5*(-u[t0][x + 2][y + 2]) + r5*u[t0][x + 2][y + 3])*v[t0][x +
          ↪  2][y + 2] + 1.0e-1F*(r2*u[t0][x + 1][y + 2] + r2*u[t0][x + 3][y + 2] +
          ↪  r3*u[t0][x + 2][y + 1] + r3*u[t0][x + 2][y + 3]));
          v[t1][x + 2][y + 2] = dt*(r4*v[t0][x + 2][y + 2] - 5.0e-1F*r5*(-p[i_10][x + 2][y +
          ↪  1] + p[i_10][x + 2][y + 3]) - (r1*(-v[t0][x + 2][y + 2]) + r1*v[t0][x + 3][y +
          ↪  2])*u[t0][x + 2][y + 2] + (-2.0e-1F)*(r2*v[t0][x + 2][y + 2] + r3*v[t0][x +
          ↪  2][y + 2]) - (r5*(-v[t0][x + 2][y + 2]) + r5*v[t0][x + 2][y + 3])*v[t0][x +
          ↪  2][y + 2] + 1.0e-1F*(r2*v[t0][x + 1][y + 2] + r2*v[t0][x + 3][y + 2] +
          ↪  r3*v[t0][x + 2][y + 1] + r3*v[t0][x + 2][y + 3]));
        }
      }
      STOP(section2,timers)
    }
    START(section3)
    for (int y = y_m; y <= y_M; y += 1)
    {
      u[t1][2][y + 2] = 0;
      u[t1][42][y + 2] = 0;
      v[t1][2][y + 2] = 0;
      v[t1][42][y + 2] = 0;
    }
    STOP(section3,timers)

    START(section4)
    for (int x = x_m; x <= x_M; x += 1)
    {
      u[t1][x + 2][2] = 0;
      u[t1][x + 2][42] = 1;
      v[t1][x + 2][42] = 0;
      v[t1][x + 2][2] = 0;
    }
    STOP(section4,timers)
  }

  return 0;
}
```

Bibliography

[1] D. Bestion, "From the direct numerical simulation to system codes—perspective for the multi-scale analysis of lwr thermalhydraulics," *Nuclear Engineering and Technology*, vol. 42, no. 6, pp. 608–619, 2010.

[2] R. R. Schultz, R. A. Riemke, C. B. Davis, and G. Nurnberg, "Comparison: RELAP5-3D© systems analysis code and fluent CFD code momentum equation formulations," 2003.

[3] E. Brega and E. Salina, "COBRA-EN, Thermal-Hydraulic Transient Analysis of Reactor Cores," 2001.

[4] M. N. Avramova and R. K. Salko, "CTF theory manual," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States). Consortium for ..., Tech. Rep., 2016.

[5] V. Kucukboyaci, Y. Sung, and R. Salko, "COBRA-TF parallelization and application to PWR reactor core subchannel DNB analysis," in *Joint International Conference on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method*, 2015, pp. 1–18.

[6] Y. Yoo and D. Hwang, "Development of a subchannel analysis code MATRA (Ver. α)," Korea Atomic Energy Research Institute, Tech. Rep., 1998.

[7] V. Sanchez, U. Imke, A. Ivanov, and R. Gomez, "SUBCHANFLOW: a thermal hydraulic sub-channel program to analyse fuel rod bundles and reactor cores," 2010.

[8] G. Chen *et al.*, "Research on the Subchannel Analysis Method via CFD Analysis for PWR," *Nuclear Science and Engineering*, vol. 188, no. 3, pp. 270–281, 2017.

[9] J. Mahaffy *et al.*, "Best practice guidelines for the use of CFD in nuclear reactor safety applications," Organisation for Economic Co-Operation and Development, Tech. Rep., 2007.

[10] K. Zhang, "The multiscale thermal-hydraulic simulation for nuclear reactors: A classification of the coupling approaches and a review of the coupled codes," *International Journal of Energy Research*, vol. 44, no. 5, pp. 3295–3315, 2020.

[11] F. D'Auria, J. L. G. Moreno, G. M. Galassi, D. Grgic, and A. Spadoni, "Three mile island unit 1 main steam line break three-dimensional neutronics/thermal-hydraulics analysis: Application of different coupled codes," *Nuclear technology*, vol. 142, no. 2, pp. 180–204, 2003.

[12] J. J. Escalante, V. Di Marcello, V. S. Espinoza, and Y. Perin, "Application of the athlet/cobra-tf thermal-hydraulics coupled code to the analysis of bwr atws," *Nuclear Engineering and Design*, vol. 321, pp. 318–327, 2017.

[13] K. Zhang and V. H. Sanchez-Espinoza, "Coupling of trace and subchanflow based on the exterior communication interface," *Progress in Nuclear Energy*, vol. 119, p. 103 040, 2020.

[14] D. Aumiller, E. Tomlinson, and R. Bauer, "A coupled relap5-3d/cfd methodology with a proof-of-principle calculation," *Nuclear Engineering and Design*, vol. 205, no. 1-2, pp. 83–90, 2001.

[15] Y. Yan, "Development of a coupled CFD–system-code capability (with a modified porous media model) and its applications to simulate current and next generation reactors," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012.

[16] X. Zhang, K. Zhang, V. Sanchez-Espinoza, and H. Chen, "Multi-scale coupling of cfd code and sub-channel code based on a generic coupling architecture," *Annals of Nuclear Energy*, vol. 141, p. 107 353, 2020.

[17] Y. Yu, E. Merzari, A. Obabko, and J. Thomas, "A porous medium model for predicting the duct wall temperature of sodium fast reactor fuel assembly," *Nuclear Engineering and Design*, vol. 295, pp. 48–58, 2015.

[18] E. Zarifi, G. Jahanfarnia, and F. Veysi, "Thermal–hydraulic modeling of nanofluids as the coolant in VVER-1000 reactor core by the porous media approach," *Annals of Nuclear Energy*, vol. 51, pp. 203–212, 2013.

[19] K. Liu, M. Wang, W. Tian, J. Zhang, S. Qiu, and G. Su, "CorTAF: A nuclear reactor core three-dimensional thermal-hydraulic characteristics analysis code based on OpenFOAM," *Nuclear Engineering and Design*, vol. 405, p. 112 209, 2023.

[20] R. Chen, M. Tian, S. Chen, W. Tian, G. Su, and S. Qiu, "Three dimensional thermal hydraulic characteristic analysis of reactor core based on porous media method," *Annals of Nuclear Energy*, vol. 104, pp. 178–190, 2017.

[21] Z. V. Stosic and V. D. Stevanovic, "Advanced three-dimensional two-fluid porous media method for transient two-phase flow thermal-hydraulics in complex geometries," *Numerical Heat Transfer: Part B: Fundamentals*, vol. 41, no. 3-4, pp. 263–289, 2002.

[22] S. J. Yoon, S. B. Kim, G. C. Park, H. Y. Yoon, and H. K. Cho, "Application of CUPID for subchannel-scale thermal–hydraulic analysis of pressurized water reactor core under single-phase conditions," *Nuclear Engineering and Technology*, vol. 50, no. 1, pp. 54–67, 2018.

[23] H. Kim *et al.*, "Development of CUPID-SG for the analysis of two-phase flows in PWR steam generators," *Progress in Nuclear Energy*, vol. 77, pp. 132–140, 2014.

[24] Y. J. Cho and H. Y. Yoon, "Numerical analysis of the ROCOM boron dilution benchmark experiment using the CUPID code," *Nuclear Engineering and Design*, vol. 341, pp. 167–175, 2019.

[25] J. Miettinen and I. Karppinen, "A porous media calculation applications for the isolation condenser heat transfer and circulation," 2003.

[26] M. Ilvonen and V. Hovi, "8.2 The porous medium model PORFLO for 3D two-phase flow and its application to BWR fuel bundle simulations," *The Finnish Research Programme on Nuclear Power Plant Safety 2007–2010 Interim Report*, p. 131, 2007.

[27] E. Takasuo, V. Hovi, and M. Ilvonen, "Applications and development of the PORFLO 3D code in nuclear power plant thermal hydraulics," in *International Conference on Nuclear Engineering*, American Society of Mechanical Engineers, vol. 44984, 2012, pp. 393–402.

[28] M. Ilvonen, V. Hovi, and V. Taivassalo, "3D Core Thermal Hydraulics With the PORFLO Code: Turbulence Modeling and Porous Medium With Porosity Steps," in *International Conference on Nuclear Engineering*, American Society of Mechanical Engineers, vol. 45943, 2014, V004T10A032.

[29] V. J. Chavez, U. Imke, and V. Sanchez-Espinoza, "TWOPORFLOW: A two-phase flow porous media code, main features and validation with BWR-relevant bundle experiments," *Nuclear Engineering and Design*, vol. 338, pp. 181–188, 2018.

[30] W. Feng, U. Imke, K. Zhang, V. Sanchez-Espinoza, and H. Chen, "Validation of the TWOPORFLOW code for the core analysis of liquid metal-cooled reactor with selected experiments," *Nuclear Engineering and Design*, vol. 390, p. 111 708, 2022.

[31] H. Jasak, A. Jemcov, Z. Tukovic, *et al.*, "Openfoam: A c++ library for complex physics simulations," in *International workshop on coupled methods in numerical dynamics*, Dubrovnik, Croatia), vol. 1000, 2007, pp. 1–20.

[32] C. Fiorina, I. Clifford, M. Aufiero, and K. Mikityuk, "GeN-Foam: a novel OpenFOAM® based multi-physics solver for 2D/3D transient analysis of nuclear reactors," *Nuclear Engineering and Design*, vol. 294, pp. 24–37, 2015.

[33] S. Radman, C. Fiorina, and A. Pautz, "Development of a novel two-phase flow solver for nuclear reactor analysis: Validation against sodium boiling experiments," *Nuclear Engineering and Design*, vol. 384, p. 111 422, 2021.

[34] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandie, "Moose: A parallel computational framework for coupled systems of nonlinear equations," *Nuclear Engineering and Design*, vol. 239, no. 10, pp. 1768–1778, 2009.

[35] A. Novak *et al.*, "Pronghorn: A multidimensional coarse-mesh application for advanced reactor thermal hydraulics," *Nuclear Technology*, vol. 207, no. 7, pp. 1015–1046, 2021.

[36] M. Viellieber and A. G. Class, "Anisotropic porosity formulation of the coarse-grid-cfd (cgcfd)," in *International Conference on Nuclear Engineering*, American Society of Mechanical Engineers, vol. 44984, 2012, pp. 473–483.

[37] B. Liu, S. He, C. Moulinec, and J. Uribe, "Sub-channel cfd for nuclear fuel bundles," *Nuclear Engineering and Design*, vol. 355, p. 110 318, 2019.

[38] B. Liu, S. He, C. Moulinec, and J. Uribe, "Coupled porous media approaches in sub-channel CFD," *Nuclear Engineering and Design*, vol. 377, p. 111 159, 2021.

[39] I. Park, J. Lee, S. Lee, H. Yoon, and J. Jeong, "An implicit code coupling of 1-D system code and 3-D in-house CFD code for multi-scaled simulations of nuclear reactor transients," *Annals of Nuclear Energy*, vol. 59, pp. 80–91, 2013.

[40] J. Lee, A. Facchini, and H. G. Joo, "Development of a drift-flux model based core thermal-hydraulics code for efficient high-fidelity multiphysics calculation," *Nuclear Engineering and Technology*, vol. 51, no. 6, pp. 1487–1503, 2019.

[41] K. S. Chaudri, J. Kim, and Y. Kim, "Development and validation of a fast sub-channel code for lwr multi-physics analyses," *Nuclear Engineering and Technology*, vol. 51, no. 5, pp. 1218–1230, 2019.

[42] M. Gui, W. Tian, D. Wu, R. Chen, M. Wang, and G. Su, "Parallelization and application of sacos for whole core thermal-hydraulic analysis," *Nuclear Engineering and Technology*, vol. 53, no. 12, pp. 3902–3909, 2021.

[43] F. Rathgeber *et al.*, "Firedrake: Automating the finite element method by composing abstractions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 43, no. 3, pp. 1–27, 2016.

[44] C. T. Jacobs, S. P. Jammy, and N. D. Sandham, "OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures," *Journal of Computational Science*, vol. 18, pp. 12–23, 2017.

[45] F. Luporini *et al.*, "Architecture and performance of Devito, a system for automated stencil computation," *arXiv preprint arXiv:1807.03032*, 2018.

[46] M. Alnæs *et al.*, "The fenics project version 1.5," *Archive of numerical software*, vol. 3, no. 100, 2015.

[47] J. Hoffman, J. Jansson, and N. Jansson, "Fenics-hpc: Automated predictive high-performance finite element computing with applications in aerodynamics," in *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part I 11*, Springer, 2016, pp. 356–365.

[48] Q. Zhu and J. Yan, "A moving-domain cfd solver in fenics with applications to tidal turbine simulations in turbulent flows," *Computers & Mathematics with Applications*, vol. 81, pp. 532–546, 2021.

[49] M. N. Vlasov and I. G. Merinov, "Application of an integral turbulence model to close the model of an anisotropic porous body as applied to rod structures," *Fluids*, vol. 7, no. 2, p. 77, 2022.

[50] T. M. Bendall, "On coupling resolved and unresolved physical processes in finite element discretisations of geophysical fluids," Ph.D. dissertation, Imperial College London, 2019.

[51] W. I. Scott, S. C. Kramer, P. R. Holland, K. W. Nicholls, M. J. Siegert, and M. D. Piggott, "Towards a fully unstructured ocean model for ice shelf cavity environments: Model development and verification using the firedrake finite element framework," *Ocean Modelling*, vol. 182, p. 102 178, 2023.

[52] D. Shapero, J. Badgeley, A. Hoffmann, and I. Joughin, "Icepack: A new glacier flow modeling package in python, version 1.0," *Geoscientific Model Development Discussions*, vol. 2021, pp. 1–34, 2021.

[53] F. D. Witherden, A. M. Farrington, and P. E. Vincent, "Pyfr: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach," *Computer Physics Communications*, vol. 185, no. 11, pp. 3028–3040, 2014.

[54] J. Park, F. Witherden, and P. Vincent, "High-order implicit large-eddy simulations of flow over a naca0021 aerofoil," *AIAA journal*, vol. 55, no. 7, pp. 2186–2197, 2017.

[55] N. A. Loppi, "High-order incompressible computational fluid dynamics on modern hardware architectures," Ph.D. dissertation, Imperial College London, 2019.

[56] D. J. Lusher, S. P. Jammy, and N. D. Sandham, "Shock-wave/boundary-layer interactions in the automatic source-code generation framework opensbli," *Computers & Fluids*, vol. 173, pp. 17–21, 2018.

[57] F. H. Harlow and J. E. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *The physics of fluids*, vol. 8, no. 12, pp. 2182–2189, 1965.

[58] C. M. Rhie and W.-L. Chow, "Numerical study of the turbulent flow past an airfoil with trailing edge separation," *AIAA journal*, vol. 21, no. 11, pp. 1525–1532, 1983.

[59] S. K. Choi, "Note on the use of momentum interpolation method for unsteady flows," *Numerical Heat Transfer: Part A: Applications*, vol. 36, no. 5, pp. 545–550, 1999.

[60] Y. Kawaguchi, W.-Q. Tao, and H. Ozoe, "Checkerboard pressure predictions due to the underrelaxation factor and time step size for a nonstaggered grid with momentum interpolation method," *Numerical Heat Transfer: Part B: Fundamentals*, vol. 41, no. 1, pp. 85–94, 2002.

[61] S.-K. Choi, S.-O. Kim, C.-H. Lee, and H.-K. Choi, "Use of the momentum interpolation method for flows with a large body force," *Numerical Heat Transfer: Part B: Fundamentals*, vol. 43, no. 3, pp. 267–287, 2003.

[62] S. Zhang, X. Zhao, and S. Bayyuk, "Generalized formulations for the rhie–chow interpolation," *Journal of Computational Physics*, vol. 258, pp. 880–914, 2014.

[63] P. Bartholomew, F. Denner, M. H. Abdol-Azis, A. Marquis, and B. G. van Wachem, "Unified formulation of the momentum-weighted interpolation for collocated variable arrangements," *Journal of Computational Physics*, vol. 375, pp. 177–208, 2018.

[64] M. Nordlund, M. Stanic, A. K. Kuczaj, E. M. Frederix, and B. J. Geurts, "Improved piso algorithms for modeling density varying flow in conjugate fluid–porous domains," *Journal of computational physics*, vol. 306, pp. 199–215, 2016.

[65] M. Stanic, M. Nordlund, E. Frederix, A. Kuczaj, and B. Geurts, "Evaluation of oscillation-free fluid-porous interface treatments for segregated finite volume flow solvers," *Computers & Fluids*, vol. 131, pp. 169–179, 2016.

[66] A. Kozelkov, S. Lashkin, V. Efremov, K. Volkov, Y. A. Tsibereva, and N. Tarasova, "An implicit algorithm of solving navier–stokes equations to simulate flows in anisotropic porous media," *Computers & Fluids*, vol. 160, pp. 164–174, 2018.

[67] C.-T. Hsu and P. Cheng, "Thermal dispersion in a porous medium," *International Journal of Heat and Mass Transfer*, vol. 33, no. 8, pp. 1587–1597, 1990.

[68] J. C. Slattery, "Flow of viscoelastic fluids through porous media," *AIChE Journal*, vol. 13, no. 6, pp. 1066–1071, 1967.

[69] S. Whitaker, "Diffusion and dispersion in porous media," *AIChE Journal*, vol. 13, no. 3, pp. 420–427, 1967.

[70] L. Wang, L.-P. Wang, Z. Guo, and J. Mi, "Volume-averaged macroscopic equation for fluid flow in moving porous media," *International Journal of Heat and Mass Transfer*, vol. 82, pp. 357–368, 2015.

[71] K. Vafai, *Handbook of porous media*. Crc Press, 2015.

[72] A. Di Meglio, E. Di Giulio, R. Dragonetti, and N. Massarotti, "A novel model for macroscopic simulation of oscillating heat and fluid flow in porous media," *International Journal of Thermal Sciences*, vol. 181, p. 107 758, 2022.

[73] M. Kaviany, *Principles of heat transfer in porous media*. Springer Science & Business Media, 2012.

[74] W. Boscheri and L. Pareschi, "High order pressure-based semi-implicit imex schemes for the 3d navier-stokes equations at all mach numbers," *Journal of Computational Physics*, vol. 434, p. 110 206, 2021.

[75] C.-N. Xiao, F. Denner, and B. G. van Wachem, "Fully-coupled pressure-based finite-volume framework for the simulation of fluid flows at all speeds in complex geometries," *Journal of Computational Physics*, vol. 346, pp. 91–130, 2017.

[76] V. Costa, L. Oliveira, B. Baliga, and A. Sousa, "Simulation of coupled flows in adjacent porous and open domains using acontrol-volume finite-element method," *Numerical Heat Transfer, Part A: Applications*, vol. 45, no. 7, pp. 675–697, 2004.

[77] R. K. Ray and A. Kumar, "Numerical study of shear rate effect on unsteady flow separation from the surface of the square cylinder using structural bifurcation analysis," *Physics of Fluids*, vol. 29, no. 8, 2017.

[78] R. Davis and E. Moore, "A numerical study of vortex shedding from rectangles," *Journal of Fluid Mechanics*, vol. 116, pp. 475–506, 1982.

[79] T.-C. Jue, "Numerical analysis of vortex shedding behind a porous square cylinder," *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 14, no. 5, pp. 649–663, 2004.

[80] S. Dong, W. Liu, Y. Liu, and J. Shan, "Application of the improved spacer grid model in subchannel analysis code," *Nuclear Technology*, vol. 205, no. 1-2, pp. 352–363, 2019.

[81] S. S. Arshi, S. Mirvakili, and F. Faghihi, "Modified cobra-en code to investigate thermal-hydraulic analysis of the iranian vver-1000 core," *Progress in Nuclear Energy*, vol. 52, no. 6, pp. 589–595, 2010.

[82] D. Thatoi, A. Acharya, N. Anand, P. Acharya, and V. Agrawal, "Numerical solution of radial heat conduction in an infinitely long cylinder," in *Proceedings of the International Conference on Mechanical Engineering, Dhaka, Bangladesh (ICME2011)*, 2011, pp. 18–20.

[83] Y. I. AlZaben and E. A. Villarino, "Development of fuel rod thermal-hydraulic model for the thermal-hydraulic feedback in condor code," *Mecanica Computacional*, vol. 33, no. 47, pp. 2969–2982, 2014.

[84] F. P. Incropera, D. P. DeWitt, T. L. Bergman, A. S. Lavine, *et al.*, *Fundamentals of heat and mass transfer*. Wiley New York, 1996, vol. 6.

[85] J. Odii, E. Agyekum, B. Afornu, and M. Ansah, "Validation of results of analytical calculation of steady-state heat transfer in nuclear fuel element using ansys apdl," *Int Res J Eng Technol (IRJET)*, vol. 5, no. 1, pp. 1050–1055, 2018.

[86] A. Ross and R. Stoute, "Heat transfer coefficient between uo 2 and zircaloy-2," Atomic Energy of Canada Limited, Tech. Rep., 1962.

[87] R. Othman, "Steady state and transient analysis of heat conduction in nuclear fuel elements," *masters degree project, KTH-stockholm sweden*, 2004.

[88] T. Yodo, N. Takeda, N. Iida, and M. Kawachi, "Development of a sub-channel analysis code "midac" for core thermal-hydraulic design and safety analysis in pwr plants," in *International Conference on Nuclear Engineering*, American Society of Mechanical Engineers, vol. 57847, 2017, V006T08A107.

[89] J. Creer, D. Rowe, J. Bates, and A. Sutey, "Effects of sleeve blockages on axial velocity and intensity of turbulence in an unheated 7 x 7 rod bundle.[pwr]," Battelle Pacific Northwest Labs., Richland, WA (United States), Tech. Rep., 1976.

[90] E. Weiss, R. Markley, and A. Battacharyya, "Open duct cooling-concept for the radial blanket region of a fast breeder reactor," *Nuclear Engineering and Design*, vol. 16, no. 4, pp. 375–386, 1971.

[91] R. K. Salko, R. C. Schmidt, and M. N. Avramova, "Optimization and parallelization of the thermal–hydraulic subchannel code ctf for high-fidelity multi-physics applications," *Annals of Nuclear Energy*, vol. 84, pp. 122–130, 2015.

[92] K. M. Kim, N. Choi, J. Lee, and H. G. Joo, "Preliminary performance assessment of the gpu acceleration module in a pinwise core thermal hydraulics code escot," *Proceedings of Reactor Physics Asia, Osaka, Japan*, 2019.

[93] X. Luo *et al.*, "Parallelization and optimization for a pin-by-pin whole core thermal–hydraulic analysis code," *Annals of Nuclear Energy*, vol. 196, p. 110 199, 2024.

[94] D. Jacobsen, J. Thibault, and I. Senocak, "An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters," in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010, p. 522.

[95] M. T. Ho *et al.*, "A multi-level parallel solver for rarefied gas flows in porous media," *Computer Physics Communications*, vol. 234, pp. 14–25, 2019.

[96] M. Louboutin *et al.*, "Devito (v3. 1.0): An embedded domain-specific language for finite differences and geophysical exploration," *Geoscientific Model Development*, vol. 12, no. 3, pp. 1165–1187, 2019.

[97] N. Kukreja, M. Louboutin, F. Vieira, F. Luporini, M. Lange, and G. Gorman, "Devito: Automated fast finite difference computation," in *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*, IEEE, 2016, pp. 11–19.

[98] A. W. Cronenberg, "Summary of boron dilution issues and regulatory actions, memorandum awc-114.2000," U.S. Nuclear Regulatory Commission (NRC), Advisory Comittee on Reactor Safeguards, D.C. 20555·0001, Tech. Rep., 2000.

[99] K. Umminger, W. Kastner, J. Liebert, and T. Mull, "Thermal hydraulics of pwrs with respect to boron dilution phenomena. experimental results from the test facilities pkl and uptf," *Nuclear engineering and design*, vol. 204, no. 1-3, pp. 191–203, 2001.

[100] P. Gango, "Numerical boron mixing studies for loviisa nuclear power plant," *Nuclear engineering and design*, vol. 177, no. 1-3, pp. 239–254, 1997.

[101] F. Barre and M. Bernard, "The cathare code strategy and assessment," *Nuclear engineering and design*, vol. 124, no. 3, pp. 257–284, 1990.

[102] R. A. Berry *et al.*, "Relap-7 theory manual," Idaho National Lab.(INL), Idaho Falls, ID (United States), Tech. Rep., 2018.

[103] R. Macian-Juan and J. H. Mahaffy, "Numerical diffusion and the tracking of solute fields in system codes: Part i. one-dimensional flows," *Nuclear engineering and design*, vol. 179, no. 3, pp. 297–319, 1998.

[104] U. Rohde, "Flomix-r fluid mixing and flow distribution in the primary circuit,"

[105] U. Rohde *et al.*, "Fluid mixing and flow distribution in a primary circuit of a nuclear pressurized water reactor—validation of cfd codes," *Nuclear Engineering and Design*, vol. 237, no. 15-17, pp. 1639–1655, 2007.

[106] F. Moretti, D. Melideo, F. D'auria, T. Höhne, and S. Kliem, "Cfx simulations of rocom slug mixing experiments," *Journal of Power and Energy Systems*, vol. 2, no. 2, pp. 720–733, 2008.

[107] T. Höhne, S. Kliem, and U. Bieder, "Modeling of a buoyancy-driven flow experiment at the rocom test facility using the cfd codes cfx-5 and trio_u," *Nuclear Engineering and Design*, vol. 236, no. 12, pp. 1309–1325, 2006.

[108] S. Kliem, T. Sühnel, U. Rohde, T. Höhne, H.-M. Prasser, and F.-P. Weiss, "Experiments at the mixing test facility rocom for benchmarking of cfd codes," *Nuclear engineering and design*, vol. 238, no. 3, pp. 566–576, 2008.

[109] T. Toppila, "Description of the experimental and numerical studies of flow distribution in the reactor primary circuit," *Final report on WP3, EU-projekt FLOMIX-R, Fortum Nuclear Services, Helsinki*, 2004.

[110] M. Boumaza, F. Moretti, and R. Dizene, "Numerical simulation of flow and mixing in rocom facility using uniform and non-uniform inlet flow velocity profiles," *Nuclear Engineering and Design*, vol. 280, pp. 362–371, 2014.

[111] R. Mittal and G. Iaccarino, "Immersed boundary methods," *Annu. Rev. Fluid Mech.*, vol. 37, no. 1, pp. 239–261, 2005.

[112] R. Verzicco, "Immersed boundary methods: Historical perspective and future outlook," *Annual Review of Fluid Mechanics*, vol. 55, no. 1, pp. 129–155, 2023.

[113] E. Caunt, R. Nelson, T. MacArthur, F. Luporini, and G. Gorman, "A novel immersed boundary approach for irregular topography with acoustic wave equations," *Geophysics*, vol. 89, no. 4, T207–T226, 2024.

[114] M. Uhlmann, "An immersed boundary method with direct forcing for the simulation of particulate flows," *Journal of computational physics*, vol. 209, no. 2, pp. 448–476, 2005.

[115] N. Zhang and Z. C. Zheng, "An improved direct-forcing immersed-boundary method for finite difference applications," *Journal of Computational Physics*, vol. 221, no. 1, pp. 250–268, 2007.

[116] K. Taira and T. Colonius, "The immersed boundary method: A projection approach," *Journal of Computational Physics*, vol. 225, no. 2, pp. 2118–2137, 2007.

[117] P.-S. Wang, Y. Liu, and X. Tong, *Mesh2sdf*, 2022.

[118] J. Mahaffy *et al.*, "Best practice guidelines for the use of cfd in nuclear reactor safety applications-revision," Organisation for Economic Co-Operation and Development, Tech. Rep., 2015.