Development and Integration of a GPU-Accelerated Lattice Boltzmann Simulation Tool for Thermal Hydraulics with Neutronics for Multiphysics Simulations in Molten Salt Fast Reactor Cores

Master Thesis

Tom Entes

Development and Integration of a GPU-Accelerated Lattice Boltzmann Simulation Tool for Thermal Hydraulics with Neutronics for Multiphysics Simulations in Molten Salt Fast Reactor Cores

Master Thesis

by

Tom Entes

to obtain the degree of Master of Science at the Delft University of Technology,
to be defended publicly on Thursday February 13, 2025 at 15:00.

Student number:	4856015	
Project duration:	January 18, 2024 - February 13, 2025 (part time)	
Thesis committee:	Dr. Ir. M. Rohde,	TU Delft, Supervisor
	Dr. Ir. D. Lathouwers,	TU Delft, Supervisor
	Prof. Dr. Ir. B.J.H. van de Wiel,	TU Delft
	Prof. Dr. Ir. C. Vuik,	TU Delft

TU Delft

Highlights

The most important contributions from this research project are listed below.

- *Development of a novel GPU-accelerated Filter-Matrix Lattice Boltzmann Method (FM-LBM) algorithm for thermal hydraulics and precursor transport simulations. This algorithm demonstrates computational performance comparable to other GPU-optimized LBM algorithms reported in the literature. Additionally, it achieves a performance gain of approximately a factor of 1000 compared to serial implementations of the LBM algorithm for similar applications.*
- *Integration of the developed FM-LBM algorithm with the discontinuous Galerkin finite element method-based Phantom-S_N algorithm for neutron transport calculations. This integration resulted in a multiphysics simulation tool suitable for molten salt fast reactor core simulations. The tool was successfully validated under both steady-state and transient conditions against well-established benchmark codes from the literature.*
- *A forward-looking and comprehensive discussion on potential extensions and improvements to the developed multiphysics simulation tool, aimed at achieving even more realistic simulations of molten salt fast reactor cores.*

Abstract

This research focuses on developing a novel Thermal-Hydraulics-Neutronics (NTH) simulation tool to capture the complex multiphysics inside an Molten Salt Fast Reactor (MSFR) core. This is achieved by developing a GPU-accelerated Filter Matrix Lattice Boltzmann Method (FM-LBM) algorithm to simulate thermal hydraulics and precursor transport and integrating this tool with the existing Discontinuous-Galerkin Finite Element Method (DG-FEM) based Phantom-SN algorithm for neutronics simulation.

The FM-LBM algorithm simulates the evolution of thermal fluids through alternating propagation and collision steps at each grid point within a lattice grid. In our implementation, we adopt the double distribution function approach, wherein distinct distribution functions are defined for the velocity, temperature, and precursor fields, which interact via convective and source terms during the collision step. Due to its inherently symmetric computational steps performed at each lattice node across the domain, the FM-LBM algorithm is particularly well-suited for parallel computation. As a result, this study implements the algorithm using the Julia-CUDA framework, allowing us to leverage the computational power of NVIDIA GPUs. The Julia-CUDA framework was specifically selected for its high-level syntax, superior computational performance compared to other scientific languages, and its flexibility in GPU programming. This flexibility enables the optimization of GPU performance through low-level memory management, effectively utilizing the hierarchical memory architecture of NVIDIA GPUs. Consequently, a highly optimized GPU-accelerated FM-LBM algorithm is developed, tailored for thermal fluid simulation and precursor transport.

In the second stage of this research, the FM-LBM algorithm is integrated with the existing Phantom-S_N algorithm for neutronics simulations, written in Fortran-90. This integration is facilitated by enabling interaction between the Julia-based FM-LBM executable and the Fortran-90 code through system calls via the command line interface. The difference in spatial discretization between the two algorithms is addressed by applying interpolation techniques to the FM-LBM lattice grid and subsequently transforming the interpolated variables into coefficients of basis functions within the DG-FEM framework using Galerkin projection. The execution order and information exchange between the two algorithms are managed through separate implementations for steady-state and transient simulations. In steady-state simulations, the power method is employed to solve the k_{eff} eigenvalue problem. For transient simulations, the algorithm proceeds as an alternation between the FM-LBM algorithm and Phantom-S_N, with each simulating a single time step in sequence.

The developed NTH tool is benchmarked against literature data in three stages. First, the FM-LBM code is independently validated by simulating a thermal fluid flow in a side-heated cavity. The maximum velocities along the horizontal and vertical centerlines are compared with reference results from the literature, showing average discrepancies below 1%, indicating satisfactory performance of the FM-LBM algorithm. In the second and third stages, the full multiphysics simulation tool is benchmarked using the Tiberga benchmark case under steady-state and transient conditions, respectively. The steady-state study adopts a step-by-step approach to progressively couple the physical fields. At each step, the relevant physical variables are evaluated along the centerlines of the domain. For all but one step, the steady-state results showed average discrepancies below 1% when compared to benchmark data. For the transient analysis, the benchmark evaluates the system's power output response to a perturbation in the heat sink term within the frequency domain. The study examines normalized gains and phase shifts in the power output across a range of perturbation frequencies. Our simulation results align closely with the benchmark data, demonstrating that the fully coupled NTH tool provides accurate simulations for both steady-state and transient scenarios.

Additionally, the performance of the FM-LBM algorithm was evaluated by comparing the number of Million Lattice Updates Per Second (MLUPS) it achieves with results from literature focused on optimizing GPU implementations for LBM. Our solution achieved 390 MLUPS in the side-heated cavity benchmark case using double-precision floating-point numbers. This performance is considered satisfactory, as studies in the literature report performance close to 1200 MLUPS. These studies, however, consider simplified conditions, where they simulate only the velocity field, whereas our approach simulates both velocity and temperature fields. Moreover, these studies employ simplified collision operators, while our implementation utilizes the filter matrix operator, which is considered as the most computationally intensive step in our algorithm.

Finally, a forward looking discussion is introduced in which extension of the simulation tool are proposed to advance towards realistic MSFR core simulations, such as the integration of turbulence modeling by including Large Eddy Simulation (LES) into the LBM framework, and by implementing the actual geometry of an MSFR by simulating a segment of the cylindrical design of the MSFR reactor core.

Contents

Highlights ii

Abstract iii

Nomenclature x

1 Introduction 1
- 1.1 The Molten Salt Fast Reactor 1
- 1.2 Existing Literature 3
  - 1.2.1 LBM for Thermal Fluid Simulation 3
  - 1.2.2 GPU Accelerated LBM Simulation Techniques 4
  - 1.2.3 Multiphysics Simulation Tools for MSFR Calculations 4
- 1.3 Research Goals 5
- 1.4 Thesis Outline 5

2 Theory 6
- 2.1 Fluid Dynamics 6
  - 2.1.1 Conservation Laws 6
  - 2.1.2 Thermal Hydraulics 7
  - 2.1.3 Dimensionless Numbers 7
  - 2.1.4 Kinetic Theory 8
- 2.2 Nuclear Reactor Physics 9
  - 2.2.1 Neutron Transport 10
  - 2.2.2 Nuclear Reactor Kinetics 10
- 2.3 Multiphysics of an MSFR Reactor Core 11
- 2.4 Parallel Programming 11
  - 2.4.1 CUDA Programming Language 12
  - 2.4.2 GPU Hardware Architecture 12
  - 2.4.3 GPU Memory Hierarchy 13
  - 2.4.4 CUDA Software Abstractions 13
  - 2.4.5 Warp Scheduling Process 14

3 Numerical Method 16
- 3.1 Thermal Fluid Simulation 16
  - 3.1.1 Lattice Boltzmann Method 16
  - 3.1.2 Filter-Matrix Lattice Boltzmann Method 18
  - 3.1.3 FM-LBM for Thermal Flows 19
  - 3.1.4 FM-LBM for Precursor Transport 20
  - 3.1.5 Boundary Conditions 21
  - 3.1.6 Unit Conversion 23
- 3.2 GPU Accelerated Thermal Fluid Simulation 24
  - 3.2.1 Julia-CUDA 24
  - 3.2.2 LBM Kernel Functions 24
  - 3.2.3 Race Conditions 25
  - 3.2.4 Performance Gains 26
- 3.3 Neutronics Simulation 28
  - 3.3.1 S_N-Equations 28
  - 3.3.2 Phantom-S_N 29
- 3.4 Multiphysics Simulation Tool 29
  - 3.4.1 Coupling Terms 30
  - 3.4.2 Steady-State Simulation 30
  - 3.4.3 Transient Simulation 32
  - 3.4.4 Galerkin Projection 33
  - 3.4.5 B-Spline Interpolation 34

3.4.6 Software Coupling . . . 35

4 Validation of Thermal Flow Simulation 37
4.1 Benchmark Setup . . . 37
4.2 Benchmark Results . . . 38
4.3 Performance Analysis . . . 41

5 Validation of Steady-State Multiphysics Coupled Simulation 44
5.1 Benchmark Setup . . . 44
5.2 Error Quantification . . . 46
5.3 Step 0.1: Velocity field . . . 46
5.4 Step 0.2: Neutron flux . . . 48
5.5 Step 0.3: Temperature field . . . 48
5.6 Step 1.1: Circulating fuel . . . 51
5.7 Step 1.2: Power coupling . . . 54
5.8 Step 1.3: Buoyancy . . . 56
5.9 Step 1.4: Full coupling . . . 58
5.10 Convergence Speed . . . 59

6 Validation of Transient Multiphysics Coupled Simulation 61
6.1 Benchmark Setup . . . 61
6.2 Benchmark Results . . . 62
6.3 Performance Analysis . . . 63

7 Advancing Towards Actual Molten Salt Fast Reactor Core Simulations 66
7.1 3D Modeling . . . 66
7.1.1 Pure Buoyancy-Driven Flow . . . 67
7.1.2 Buoyancy-Driven Flow With Shear Momentum Source . . . 68
7.2 Turbulence Modeling Using LBM . . . 70
7.2.1 Energy Cascade . . . 70
7.2.2 Large Eddy Simulation . . . 71
7.2.3 LES-LBM . . . 72
7.3 MSFR Core Geometry Implementation . . . 72
7.3.1 Periodic Boundary Conditions . . . 73
7.3.2 No-Slip Boundary Conditions . . . 74

8 Conclusion and Recommendations 78
8.1 FM-LBM Model Development . . . 78
8.2 Coupling Mechanism . . . 79
8.3 Comparison to Benchmark . . . 79
8.4 Computational Performance . . . 80
8.5 GPU-Accelerated FM-LBM in MSFR Core Simulation Tools . . . 81
8.6 Recommendations for Future Research . . . 81

References 83

A Neutronics Data From The Tiberga Benchmark Case 88

B Simulated Observables for Step 1.4 of The Tiberga Benchmark 90

C Snapshots of Transient Simulation Results 92

D 1D Cross-Section Plots of 3D Simulation Results 94
D.1 Pure Buoyancy-Driven Flow . . . 94
D.2 Buoyancy-Driven Flow With Shear Momentum Source . . . 95

List of Figures

1.1 Overview of the MSFR reactor core and its components 2

2.1 Schematic overview of the relevant physical quantities in an MSFR reactor core and the processes through which they are interrelated 11
2.2 Hardware architecture of an NVIDIA GPU 13
2.3 Software abstractions used in CUDA programming 14
2.4 Illustration of the scheduling process of a warp scheduler 14

3.1 Examples of velocity schemes used in LBM simulations 17
3.2 Visual representation of the LBM algorithm 18
3.3 Difference between link-wise and wet-node boundary conditions for LBM simulation 21
3.4 Visualization of periodic boundary conditions in LBM 22
3.5 Visualization of Dirichlet boundary conditions in LBM 22
3.6 Visualization of Neumann boundary conditions in LBM 23
3.7 Overview of the GPU accelerated FM-LBM algorithm 25
3.8 Illustration of the storage of particle distribution functions in 1D arrays 26
3.9 Different configurations of nested for loops when performing matrix multiplication. 27
3.10 Extended schematic overview of key physical quantities in MSFR reactor core simulations . . . 30
3.11 Abstract overview of the coupling between the FM-LBM and Phantom-S_N codes 30
3.12 Overview of the differential spatial discretization of the FM-LBM and Phantom-S_N algorithms . 34
3.13 1D zero-order, first-order, and second-order base functions for B-spline functions 35
3.14 Illustration of interpolation in FM-LBM scheme using B-splines for both 1D and 2D 35

4.1 Problem domain of a side-heated 2D square cavity 37
4.2 Grid dependence of side-heated square cavity simulation results 39
4.3 Simulation results of the side-heated 2D square cavity 40
4.4 Simulation speed of the thermal FM-LBM algorithm expressed in MLUPS 41
4.5 Execution time of various kernel functions for simulating both velocity and temperature fields . 42

5.1 Problem domain of the Tiberga benchmark study 45
5.2 Average discrepancies of the horizontal and vertical components of the simulated velocity field along the horizontal and vertical centerlines as a function of grid size 47
5.3 Simulation results of step 0.1 form the Tiberga benchmark, horizontal velocity 47
5.4 Simulation results of step 0.1 form the Tiberga benchmark, vertical velocity 48
5.5 Simulated fission rate density along the horizontal centerline from Step 0.2 of the Tiberga benchmark case 49
5.6 Average discrepancy with benchmark results as a function of Prandtl number and grid size . . 50
5.7 Discrepancy per evaluation point with benchmark results as a function of Prandtl number . . . 50
5.8 Simulated temperature field along the horizontal and vertical centerlines for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000, compared to the benchmark results 51
5.9 Average discrepancy with benchmark results as a function of Schmidt number and grid size . . 52
5.10 Discrepancy per evaluation point with benchmark results as a function of Schmidt number . . . 53
5.11 Simulated delayed neutron source along the horizontal and vertical centerlines for a simulation grid of 501 $\times$ 501 with a Schmidt number of 1500, compared to the benchmark results 53
5.12 Simulated temperature field along the horizontal and vertical centerlines in step 1.2 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results 54
5.13 Differential fission rate density with the simulation result of step 0.2 along the horizontal and vertical centerlines in step 1.2 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results 55
5.14 Simulated horizontal velocity component along the horizontal and vertical centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results 56

5.15 Simulated vertical velocity component along the horizontal and vertical centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results . . . 57
5.16 Simulated temperature field along the horizontal and vertical centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results . . . 57
5.17 Simulated delayed neutron source along the horizontal and vertical centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results . . . 57

6.1 Snapshots of the normalized deviations of the heat transfer coefficient and the resulting power output in the transient simulations for two different frequencies . . . 63
6.2 System response to a perturbation in the frequency domain on the heat transfer coefficients illustrated with the use of Bode plots . . . 63
6.3 Simulation speed of the FM-LBM algorithm including precursor transport expressed in MLUPS 64

7.1 Temperature field and delayed neutron source from the three dimensional steady-state simulation of a pure buoyancy-driven molten salt . . . 68
7.2 Velocity field from the three dimensional steady-state simulation of a pure buoyancy-driven molten salt . . . 68
7.3 Temperature field and delayed neutron source from the three dimensional steady-state simulation of a buoyancy-driven molten salt with shear momentum source . . . 69
7.4 Velocity field from the three dimensional steady-state simulation of a buoyancy-driven molten salt with shear momentum source . . . 70
7.5 Illustration of the energy cascade in turbulent flows, showing the energy distribution of turbulent eddies as a function of the wavenumber . . . 71
7.6 Problem geometry of an MSFR core segment for numerical simulation . . . 73
7.7 Visualization of the challenge in implementing periodic boundary conditions along the sides of a segment of the MSFR core problem domain . . . 74
7.8 Illustration of the staircase approximation on a circular domain . . . 75
7.9 Visual representation of the IBB boundary scheme . . . 76
7.10 Visual representation of the PSBB boundary method . . . 77

B.1 Simulated horizontal velocity component along the horizontal and vertical centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500 . . . 90
B.2 Simulated vertical velocity component along the horizontal and vertical centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500 . . . 90
B.3 Simulated temperature field along the horizontal and vertical centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500 . . . 91
B.4 Simulated delayed neutron source along the horizontal and vertical centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501 $\times$ 501 with a Prandtl number of 1000 and Schmidt number 1500 . . . 91

C.1 Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.025$ . . . 92
C.2 Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.05$ . . . 92
C.3 Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.1$ . . . 92
C.4 Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.4$ . . . 92
C.5 Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.8$ . . . 93

D.1 u_x along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow . . . 94
D.2 u_y along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow . . . 94
D.3 u_z along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow . . . 95

D.4 Temperature field along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow 95
D.5 Delayed neutron source along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow 95
D.6 u_x along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source 95
D.7 u_y along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source 96
D.8 u_z along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source 96
D.9 Temperature field along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source 96
D.10 Delayed neutron source along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source 96

List of Tables

2.1 Overview of the dimensionless numbers used in this research . . . 8
2.2 Overview of the memory hierarchy in an NVIDIA GPU . . . 13

4.1 Overview of the FM-LBM parameters used for the simulation of a side-heated 2D square cavity 38
4.2 Quantitative comparison between simulation results and benchmark results of the side-heated square cavity 501×501 . . . 39

5.1 Atomic composition of the LiF - BeF_2 - UF_4 fuel salt . . . 44
5.2 Upper bounds of the six energy groups used in the Tiberga benchmark study . . . 45
5.3 Fractions and decay constants of the eight precursor families used in the Tiberga benchmark study . . . 45
5.4 Simulation parameter used for step 0.1 of the Tiberga benchmark case . . . 46
5.5 Simulation parameter used for step 0.2 of the Tiberga benchmark case . . . 48
5.6 Fission rate density at key coordinates along the AA' centerline calculated using the simulated neutron flux from step 0.2 of the Tiberga benchmark case . . . 49
5.7 Simulation parameter used for step 0.3 of the Tiberga benchmark case . . . 49
5.8 Simulation parameter used for step 1.1 of the Tiberga benchmark case . . . 51
5.9 Effective multiplication factor, reactivity, and reactivity change relative to step 0.2, as calculated in this study for step 1.1 of the Tiberga benchmark case, alongside the benchmark values for comparison . . . 54
5.10 Effective multiplication factor, reactivity, and reactivity change relative to step 1.1, as calculated in this study for step 1.2 of the Tiberga benchmark case, alongside the benchmark values for comparison . . . 55
5.11 Simulation parameter used for step 1.3 of the Tiberga benchmark case . . . 56
5.12 Effective multiplication factor, reactivity, and reactivity change relative to step 0.2, as calculated in this study for step 1.3 of the Tiberga benchmark case, alongside the benchmark values for comparison . . . 58
5.13 Reactivity (in per cent mille) and reactivity change relative to step 0.2 for different values of U_{lid} and P_{ref} as calculated in step 1.4 of the Tiberga benchmark case . . . 58

6.1 Overview of the number of FM-LBM timesteps per Phantom-S_N timestep for various perturbation frequencies . . . 62

7.1 Simulation parameter used for the simulation of a pure buoyancy-driven molten salt in a cubic box in steady state conditions . . . 67
7.2 Simulation parameter used for the simulation of a buoyancy-driven molten salt with shear momentum source in a cubic box in steady state conditions . . . 69

A.1 Average total number of neutrons emitted per fission event, prompt and delayed fission spectra, average energy emitted per fission event, and inverse neutron group velocities . . . 88
A.2 Total and fission cross-sections per energy group . . . 88
A.3 P_0 scattering cross sections . . . 89
A.4 P_1 scattering cross sections . . . 89
A.5 P_2 scattering cross sections . . . 89
A.6 P_3 scattering cross sections . . . 89

Nomenclature

Abbreviations

Abbreviation	Definition
ABB	Anti Bounce Back
BGK	Bhatnagar Gross Krook
COP	Conference of the Parties
CUDA	Compute Unified Device Architecture
DDF	Double Distribution Function
DG-FEM	Discontinuous Galerkin Finite Elements Method
ECS	Equilibrium Climate Sensitivity
FM-LBM	Filter-Matrix Lattice Boltzmann Method
FNSE	Filter Navier-Stokes Equation
GPU	Graphics Processing Unit
GPGPU	General-Purpose computing on Graphics Processing Units
HBB	Halfway Bounce Back
HPC	High Performance Computing
IBB	Interpolated Bounce-Back
IPCC	Intergovernmental Panel on Climate Change
LBE	Lattice Boltzmann Equation
LBM	Lattice Boltzmann Method
LES	Large Eddy Simulation
LUPS	Lattice Updates Per Second
MG-NTE	Multi-Group Neutron Transport Equation
MLUPS	Million Lattice Updates Per Second
MRT	Multi Relaxation Times
MSFR	Molten Salt Fast Reactor
MSR	Molten Salt Reactor
NSE	Navier-Stokes Equations
NTE	Neutron Transport Equation
NTH	Neutronics Thermal Hydraulics
PSBB	Partially Saturated Bounce-Back
SBB	Simple Bounce-Back
SGS	Sub-Grid Scale
SM	Streaming Multiprocessor
TRT	Two Relaxation Times
UNFCCC	UN Framework Convention on Climate Change
WALE	Wall-Adapting Local Eddy

Symbols

Symbol	Definition	Unit
B_p	Basis function of B-spline with order p	[-]
$\mathbf{c}_i$	Discrete lattice velocity in direction i	[m s^{-1}]
c_s	Speed of sound	[m s^{-1}]
C	Precursor concentration	[m^{-3}]
C_p	Specific heat capacity	[J kg^{-1} K^{-1}]
D	Molecular diffusivity	[m^2 s^{-1}]
Da	Damköhler number	[-]
E	Energy	[J]
E_{ik}	Filter matrix	[-]

Symbol	Definition	Unit
$\mathbf{f}$	Specific body force	[N kg^{-1}]
f	Particle distribution function for velocity	[-]
f	Perturbation frequency	[s^{-1}]
$\mathbf{g}$	Gravitational constant of the earth	[m s^{-2}]
g	Particle distribution function for temperature	[-]
Gr	Grashof number	[-]
h^j	Particle distribution function for precursor family j	[-]
h_i	Basis functions of FEM mesh element	[-]
k	Multiplication factor	[-]
k_{eff}	Effective multiplication factor	[-]
Ma	Mach number	[-]
N_{grid}	Number of grid points in lattice	[-]
Nu	Nusselt number	[-]
p	Pressure	[N m^{-2}]
P	Power	[W]
Pr	Prandtl number	[-]
q	Temperature source term	[W m^{-3}]
$\mathbf{r}$	Space coordinate	[m]
Ra	Rayleigh number	[-]
Re	Reynolds number	[-]
S_f	Fission source	[m s^{-1}]
Sc	Schmidt number	[-]
t	Time	[s]
T	Temperature	[K]
$\mathbf{u}$	Velocity	[m s^{-1}]
v	Neutron velocity	[m s^{-1}]
V_e	Volume of FEM mesh element	[m^3]
$\mathbf{x}$	Space coordinate on lattice grid	[m]
Y	Spherical harmonic	[-]
α	Thermal diffusivity	[m^2 s^{-1}]
$\alpha^{\pm}$	Solution vector for velocity	[-]
β	Delayed neutron fraction	[-]
β_{th}	Thermal expansion coefficient	[K^{-1}]
$\beta^{\pm}$	Solution vector for temperature	[-]
γ	Heat transfer coefficient	[W K^{-1} m^{-3}]
$\gamma^{(l)}$	Solution vector of power iteration	[-]
$\gamma^{j\,\pm}$	Solution vector for precursor family j	[-]
Δ	Cut-off length	[m]
ε	Discrepancy	[-]
λ	Decay constant	[s^{-1}]
ν	Average number of neutrons emitted per fission	[-]
ν	Kinematic viscosity	[m^2 s^{-1}]
ν_t	Eddy viscosity	[m^2 s^{-1}]
ν_e	Effective viscosity	[m^2 s^{-1}]
$\boldsymbol{\xi}$	Particle velocity	[m s^{-1}]
ρ	Density	[kg m^{-3}]
ρ	Reactivity	[-]
$\boldsymbol{\sigma}$	Cauchy stress tensor	[N m^{-2}]
σ	Microscopic nuclear cross section	[m^2]
Σ	Macroscopic nuclear cross section	[m^{-1}]
τ	Relaxation time	[s]
ϕ	Scalar neutron flux	[m^{-2} s^{-1}]
φ	Angular neutron flux	[m^{-2} s^{-1}]
χ	Energy spectrum of emitted neutrons	[-]
ω	Numerical weight	[-]
Ω	Collision operator	[-]
$\hat{\boldsymbol{\Omega}}$	Angular direction	[-]

1 Introduction

“Much more likely than not, global warming is upon us. We should expect weather patterns to continue to change and the seas to continue to rise, in an ever worsening pattern, in our lifetimes and on into our grandchildren’s. The question has graduated from the scientific community: climate change is a major social, economic, and political issue. Nearly everyone in the world will need to adjust. It will be hardest for the poorer groups and nations among us, but nobody is exempt”

– Spencer R. Weart

In his book *The Discovery of Global Warming* [113], Spencer R. Weart provides a comprehensive overview of the scientific history surrounding global warming and its connection to the rise in atmospheric greenhouse gases due to the industrialization of modern society. The book describes how pioneering climate scientists and key scientific studies led to a consensus within the scientific community on the causal relationship between human activities and climate change. In response to these findings, numerous geopolitical efforts have been undertaken to reduce global greenhouse gas emissions, primarily facilitated through the UN Framework Convention on Climate Change (UNFCCC), an international treaty signed by 197 countries in 1992. The UNFCCC’s supreme governing body, the Conference of the Parties (COP), holds an annual forum where governments debate the international efforts made to mitigate climate change. These meetings have led to significant international agreements on emission-reduction commitments, including the Kyoto Protocol in 2005 [81] and the Paris Agreement in 2015 [2]. The latter established the global goal of achieving net-zero emissions by 2050 and aimed to limit global warming to 1.5°C above pre-industrial levels, a threshold generally regarded as the upper limit beyond which we risk experiencing severe and irreversible effects on the Earth’s ecosystems [26].

With the latest report from the Intergovernmental Panel on Climate Change (IPCC) estimating the Equilibrium Climate Sensitivity (ECS)[1] at 3°C [59], the demand for renewable, carbon-free energy sources such as wind, solar, and hydroelectric power has never been greater. While these renewable sources are effective and widely deployed, their energy output remains dependent on environmental and weather conditions. Consequently, nuclear energy is emerging as an increasingly attractive option for sustainable power generation, due to its ability to produce continuous, reliable energy regardless of external factors. In response to this need, the Generation IV International Forum (GIF) was established in 2001 as a collaborative research initiative aimed at developing advanced nuclear reactors with improved sustainability, safety, and reliability [114]. Six reactor designs have been selected for further research under the Generation IV program [51], one of which is the Molten Salt Reactor, the focus of this research.

1.1. The Molten Salt Fast Reactor

The Molten Salt Reactor (MSR) refers to a family of reactor designs that are uniquely characterized by the use of a fluid molten salt mixture, which functions as both the fuel and the coolant of the system [4]. Liquid-fueled reactor designs based on molten salt offer enhanced safety, sustainability, and waste management compared to conventional solid-fuel reactors, making them promising candidates to fulfill the goals set by the GIF. One such design is the Molten Salt Fast Reactor (MSFR), in which the fuel salt contains fertile isotopes.

[1]ECS refers to the projected long-term increase in global surface temperature resulting from a doubling of atmospheric carbon dioxide concentrations, once the climate system has reached equilibrium.

Combined with the fast neutron spectrum in which the reactor operates, this configuration enables a breeding mechanism. The MSFR reactor design is still in the conceptual phase, with most design decisions guided by numerical modeling results. The reference MSFR model used in these simulations is a 3 GW_{th} reactor that employs a molten binary fluoride salt, composed of 77.5 mol% lithium fluoride (LiF), 20 mol% fertile thorium fluoride (ThF_4), and 2.5 mol% fissile uranium fluoride (UF_4) [4, 38, 83]. The reactor system comprises three circuits: the fuel circuit, the intermediate circuit, and the power conversion circuit. The fuel circuit, shown in Figure 1.1, is regarded as the reactor core. It has a toroidal shape with inner and outer radii of 1.05 m and 1.41 m, respectively, and a height of 1.6 m at the center, increasing to 2.25 m at the boundary [4, 83]. The fuel salt inside the reactor core occupies a total volume of 18 m^3 and operates at a maximum temperature of 750°C. It flows through 16 segments of heat exchangers and pumps symmetrically arranged around the core, with a total circulation time of approximately 3 to 4 seconds [4, 83]. These heat exchangers and pumps facilitate the transfer of thermal energy from the fuel circuit to the intermediate circuit.

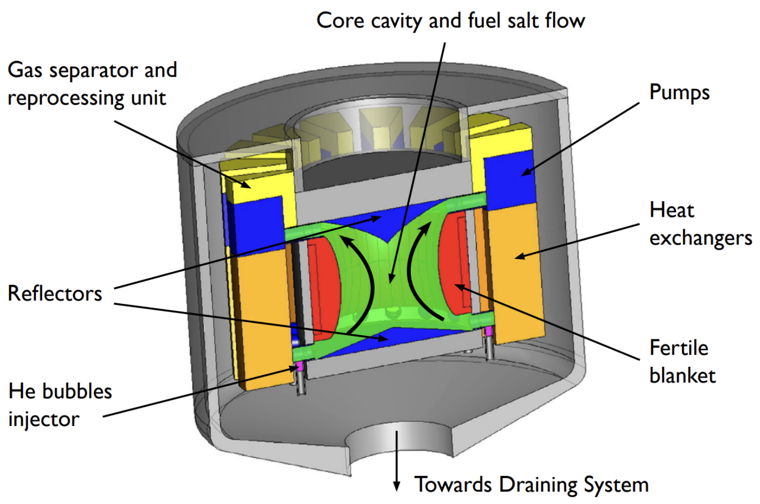

Figure 1.1: Overview of the MSFR reactor core and its components. The fuel circuit contains molten salt that serves as both the fuel and coolant. The fuel salt circulates through 16 segments of symmetrically placed heat exchangers and pumps around the core. Gaseous and metallic insoluble fission products are removed from the reactor core by injecting helium bubbles, which are recaptured at the gas separator. This separator also collects small amounts of fuel salt for continuous reprocessing. The core includes reflectors on the top and bottom walls to enhance neutron population, and a fertile blanket on the annular wall to increase the reactor's breeding ratio. A passive safety system, consisting of a draining mechanism, activates if the core overheats, diverting the fuel salt into sub-critical storage tanks positioned underground, where it is safely neutralized.

Figure 1.1 illustrates additional key features of the MSFR reactor core. For instance, the core is surrounded by neutron reflectors at both the top and bottom to prevent neutron leakage from the reactor core, as well as by an annular fertile blanket made of LiF - ThF_4 on the sides, which increases the reactor's breeding ratio [4, 83]. Moreover, helium bubbles are injected into the core via a bubble injector to capture gaseous and metallic insoluble fission products, thereby purifying the fuel salt. These bubbles are then recaptured at the gas separator, which also extracts a small volume of fuel salt daily for continuous reprocessing. This continuous reprocessing allows for real-time adjustments to the fuel composition (i.e., the ratio of fertile to fissile materials) and the removal of neutron-absorbing fission products, such as xenon, without requiring a reactor shutdown [46, 58, 84]. Additionally, the reactor incorporates a completely passive safety system by connecting the fuel circuit to a salt-draining system. The valve of this draining system is sealed with a freeze plug, which is actively cooled under normal operating conditions. In the event of an accident where excessive heat cannot be managed by the cooling system, the freeze plug melts, triggering the draining system. This allows gravitational forces to transfer the fuel salt into geometrically sub-critical tanks positioned beneath the reactor core, thereby neutralizing the fuel salt [58, 95, 109]. Finally, the outer core structure is fully enclosed by thick neutron reflectors, which reflect 99% of the neutrons back into the core. These reflectors themselves are then surrounded by a 20 cm thick layer of boron carbide (B_4C) to absorb any remaining neutrons that escape [4, 83].

In addition to the continuous reprocessing of fuel salt without requiring a reactor shutdown and the passive safety system facilitated by the salt-draining mechanism, the fluid-fuel reactor design of the MSFR offers several other advantages related to safety, sustainability, and reliability. The most notable benefit is the strong negative temperature feedback effect [46, 58, 84]. Due to the large negative feedback coefficients associated with both density feedback (arising from the thermal expansion of the fuel salt) and Doppler feedback, a total reactivity feedback of -5 pcm/K is achieved [4]. This inherent negative feedback loop causes the system to stabilize itself when heat production increases within the core. Compared to solid-fuel reactors, this negative feedback effects act much more rapidly. This is due to the density feedback effect, which acts almost instantaneous, given that there is no heat transfer delay, as the heat is generated directly within the fuel salt mixture. Consequently, reactivity in the MSFR reactor core can be entirely controlled through temperature regulation and by dissolving helium bubbles in the fuel salt, eliminating the need for control rods [4, 58, 92]. Additional safety and sustainability advantages include:

- The MSFR operates under low pressures due to the high boiling point and low vapor pressure of the fuel salt. This reduces the pressure on reactor materials, lowering the risk of pipe ruptures and other mechanical failures [58, 92].
- The continuous circulation of fuel salt ensures uniform irradiation and burnup, leading to more efficient fuel utilization compared to solid-fuel reactor designs [84, 92].
- The thorium breeding fuel cycle leads to substantially lower production of long-lived radioactive actinides, such as plutonium. In solid-fuel reactor designs, the fuel must be replaced long before achieving complete burnup. However, in the MSFR, continuous reprocessing of the fuel salt allows these elements to remain in the salt until nearly all are burned by the fast neutron spectrum. Consequently, the long-term radiotoxicity of the nuclear waste is greatly reduced in the MSFR design, with waste remaining hazardous for only a few centuries [29, 46]. Moreover, the MSFR reactor design also has the potential to burn away long-lived actinide-containing waste from other reactors, reducing its radioactivity to similar levels of only a few hundred years [29, 58].
- Thorium is approximately four times more abundant than uranium and is already produced as a by-product of rare-earth mining, making fuel acquisition easier and more sustainable [29, 58].

Most of these safety and sustainability advantages of the MSFR reactor design stem directly from the use of a liquideous fuel in the reactor core. However, this fluid-fuel design also introduces significant challenges in the design process, particularly when simulating the full multiphysics behavior of the reactor core. Since various physical fields are now strongly interrelated, conventional codes used for describing the neutronics behavior in solid-fuel reactors are inadequate for MSFR cores. These codes fail to account for the additional processes involving thermal hydraulics and precursor flow. As a result, current literature on numerical methods for MSFR core calculations, such as [92], aims to develop new simulation tools focused on combining thermal hydraulics and neutronics simulations into a single model. This research aims to contribute to this literature by introducing a new multiphysics simulation tool for the MSFR reactor cores, where the thermal flow simulation part is facilitated through GPU-accelerated Lattice Boltzmann techniques.

1.2. Existing Literature

This thesis aims to develop a so-called neutronics-thermal-hydraulics (NTH) simulation tool to model the complete multiphysics behavior of an MSFR reactor core. Broadly, this tool consists of three components: a GPU-accelerated Filter-Matrix Lattice Boltzmann Method (FM-LBM) code for thermal hydraulics simulations, a Discontinuous Galerkin Finite Element Method (DG-FEM) based code for neutronics simulations, and the coupling mechanism between these codes. The GPU-accelerated FM-LBM code for fluid simulation, along with its integration with the neutronics code, has been fully developed in this research. In contrast, an in-house neutron transport code, Phantom-S_N, is used for DG-FEM-based neutronic calculations. Consequently, this literature review will only focus on studies related to Lattice Boltzmann techniques for thermal fluid simulation and the development of NTH simulation tools for MSFR reactor core simulations, as research specifically on the topic of DG-FEM-based neutronics simulations is not directly relevant to this work.

1.2.1. LBM for Thermal Fluid Simulation

The Lattice Boltzmann Method (LBM) is a numerical technique that has gained widespread use and attention since the 1980s due to its straightforward structure and suitability for parallel implementation [55]. The book *The Lattice Boltzmann Method* by Krüger et al. [55] provides a thorough summary of LBM fundamentals, aiming to organize and standardize knowledge collected from scientific literature over the past several decades. This book is used as the main reference for the implementation of LBM in this study.

One of the key advancements in LBM was introduced by He et al. [47], who proposed the double distribution function (DDF) approach to model temperature and velocity fields simultaneously. This method solves both the momentum and energy equations within the LBM framework, allowing the simulation of thermal fluids. Since then, many studies have adapted the DDF approach to model other convective flows. For example, Chatterjee et al. [19] introduced a distribution function for enthalpy instead of temperature to simulate the energy equation.

Furthermore, Chatterjee et al. also use the DDF approach to perform magnetohydrodynamics simulations by adding a distribution function for the magnetic field along with the velocity and temperature fields. More recently, Wang et al. [110] used the DDF method to simulate convective precursor flow by adding a distribution function for precursor density. These studies provide insights into applying the DDF approach in this work, enabling the modeling of velocity, temperature, and precursor fields within the LBM framework.

In contrast to conventional LBM implementations, which often use single or multiple relaxation time collision operators, this research employs the Filter-Matrix Lattice Boltzmann Method (FM-LBM), first introduced by Somers [86]. The FM-LBM is an extension of the conventional LBM that filters out non-physical terms introduced during the discretization of the governing equations, thereby enhancing the algorithm's stability. Zhuo et al. [121, 122] demonstrated how the FM-LBM can be applied to velocity and temperature fields in both 2D and 3D thermal fluid simulations. To our knowledge, the FM-LBM approach has not yet been applied to the precursor field, however, this can be achieved through a straightforward extension, as the filter matrix operator for precursor density can be implemented analogously to that of the temperature field.

1.2.2. GPU Accelerated LBM Simulation Techniques

One key advantage of LBM algorithms over other numerical methods for fluid simulation is its scalability potential through parallel computing. The LBM algorithm consists of repeated alternations between simple calculations performed locally at each grid point in the spatial discretization, making it highly suitable for GPU computation. In 2003, Li et al. [62] demonstrated how general-purpose computing on GPUs could be used to efficiently perform LBM calculations using the OpenGL graphics API. Since then, many studies have investigated optimizations for implementing LBM on GPU hardware to achieve high performance. Notably, Habich et al. [43] and Tölke [97] illustrated how shared memory — a GPU hardware feature that provides rapid access to short-lived memory — can be leveraged to accelerate the LBM algorithm by an order of magnitude. Additionally, Delbosc et al. [24] and Tran et al. [98] showed how optimizing data layout can enhance memory coalescence, further improving the performance of LBM on GPUs. More recent studies, such as Xu et al. [116], have extended these optimizations to multi-GPU applications, enabling LBM simulations to achieve exceptionally high computational performance.

1.2.3. Multiphysics Simulation Tools for MSFR Calculations

Numerous studies have developed NTH simulation tools for MSFR core calculations. Wang et al. [110], for example, created a complete NTH simulation tool solely using LBM techniques, introducing additional distribution functions for precursor density and neutron flux. While this study provides valuable insights into using LBM for modeling precursor flow, its neutronics simulation relies on the neutron diffusion approximation. The motivation behind Wang et al.'s approach is to avoid external data exchange and interpolation that would be necessary if separate models were used for thermal-hydraulics and neutronics simulations. However, the study acknowledges the limitations in accuracy introduced by the diffusion approximation and highlights the need for neutron transport calculations to capture more complex neutronic behaviors. Other studies have similarly developed NTH simulation tools using the diffusion approximation, however, these tools often rely on multiphysics software packages such as OpenFOAM [36, 48] and COMSOL [13] for their simulations.

In contrast, this research adopts a more advanced approach by coupling the FM-LBM code for thermal-hydraulics simulations with a neutronics code specifically designed to solve the Neutron Transport Equation (NTE). This method removes the need for the neutron diffusion approximation, enabling the NTH tool to achieve greater accuracy, particularly in small-scale conditions. Such hybrid approaches, which use distinct models for thermal-hydraulics and neutronics, are also present in the literature. Many of these approaches rely on multiphysics software for thermal-fluid simulations and couple these models with advanced neutron transport algorithms, such as MCNP [20, 104] or OpenMC [33, 70].

Central to this research is the Tiberga benchmark case study, which is used to validate our NTH simulation tool. This benchmark study is a collaborative project involving researchers from four universities, each developing its own NTH simulation tool for MSFR core simulations. The most relevant benchmark results for our work are those produced by TU Delft's code, developed by Tiberga [93]. This code features a thermal-fluid simulation tool based on a DG-FEM solver to solve the Reynolds-averaged Navier–Stokes equations for incompressible fluids and, importantly, integrates this tool with the same neutronics solver, Phantom-S_N, that we use in our research. Given the identical approach for neutronics simulation, we expect our results to closely align with those generated by Tiberga.

Other NTH simulation tools in the Tiberga benchmark study include the CNRS code developed at LPSC/CNRS-Grenoble [9], the PSI code from the Paul Scherrer Institute [36], and the PoliMi code developed by Cervi et al. [15]. These tools all utilize OpenFOAM for thermal-fluid simulations and perform neutron diffusion simulations with Monte Carlo methods. Consequently, the TU Delft's code is the only tool in the Tiberga benchmark study that accounts for neutron transport effects.

1.3. Research Goals

The ultimate research goal of this thesis is to develop a full working multiphysics simulation tool that combines thermal hydraulics and neutronics simulations into a single code base to perform steady-state and transient simulations for an MSFR reactor core. This is achieved by first developing a GPU-accelerated FM-LBM algorithm for thermal fluid simulation in the Julia-CUDA framework, and subsequently coupling this code to the in-house Phantom-S_N algorithm used for neutronics simulations. During this developing process, answers will be formulated to the following research questions:

1. ***FM-LBM model development:*** *How can the Julia-CUDA framework be utilized to develop a novel GPU-accelerated FM-LBM simulation tool that effectively models thermal hydraulics and precursor transport in an MSFR reactor core?*
    - *How can the DDF approach be employed to model the velocity, temperature, and precursor fields using a single LBM algorithm?*
    - *How can we leverage the flexibility of Julia-CUDA to implement optimization techniques from the literature, such as the use of shared memory, to enhance the algorithm's performance on the GPU?*
2. ***Coupling mechanism:*** *How can the newly developed thermal-hydraulics FM-LBM simulation tool be coupled with the existing Phantom-S_N code for neutronics simulations, which is written in the Fortran-90 programming language?*
    - *How can we create a unified simulation tool that utilizes and executes codes written in different programming languages (Julia-CUDA and Fortran-90)?*
    - *When are the different codes executed, and at what points does information need to be exchanged between them?*
    - *How is information transferred between the codes, and how do we manage the differential spatial discretization during the exchange of information?*
3. ***Comparison to benchmark:*** *How does the fully coupled multiphysics NTH tool compare to established benchmark results from the literature for steady-state and transient simulations of an MSFR reactor core?*
    - *What discrepancies exist between the simulation results and the benchmark study, and can these differences be explained using physical principles?*
    - *Can we conclude that the developed multiphysics simulation tool yields accurate results for MSFR core simulations in laminar flow conditions?*
4. ***Computational performance:*** *What is the computational performance of our multiphysics simulation tool?*
    - *Can we achieve significant improvements in the computational performance of the thermal-hydraulics code compared to other fluid simulation codes that utilize serial implementations?*
    - *How does the performance of the FM-LBM algorithm compare to existing literature focused on the optimization of the LBM algorithm through GPU computing?*

1.4. Thesis Outline

The structure of this report is as follows. Chapter 2 covers the theoretical background relevant to this project, including fundamental principles of fluid dynamics, thermal hydraulics, kinetic theory, nuclear reactor physics, and GPU computation. Chapter 3 details the numerical methods employed, first explaining the FM-LBM algorithm and its GPU implementation, followed by a discussion of the complete NTH tool, which includes neutronics and the coupling mechanism between the thermal hydraulics and neutronics codes. Chapter 4 presents the benchmark case for validating the FM-LBM thermal hydraulics code by simulating thermal fluid behavior in a side-heated square cavity. Chapters 5 and 6 focus on the Tiberga benchmark case used to validate the full multiphysics simulation tool under steady-state and transient conditions, respectively. Chapter 7 provides a forward looking discussion on possible additional features to be implemented in the multiphysics simulation tool. Finally, Chapter 8 concludes the report, summarizing the findings of this thesis and offering recommendations for future research.

2 Theory

This chapter presents several key physical principles relevant to the methods applied in this research. Section 2.1 introduces fundamental concepts in fluid dynamics, including conservation laws, and kinetic theory, which serve as the foundation for our thermal flow simulation code. Section 2.2 provides an overview of the neutronics in MSFR's, detailing essential formulas from nuclear reactor physics. Section 2.3 explores the multiphysics interactions within the MSFR reactor core, describing the coupling of various physical phenomena. Finally, Section 2.4 offers a technical background on parallel programming and the CUDA software package.

Throughout this section and the remainder of the report, vector notation and Einstein summation notation are used interchangeably, employing whichever provides more clarity or conciseness. In the case of Einstein notation, repeated indices imply a summation.

2.1. Fluid Dynamics

To capture the physical phenomenon of fluids in terms of mathematical formulas, we rely on the continuum approximation. The continuum approximation is based on the premise of a continuous mass distribution within the material under consideration [82]. This means that the discrete nature of individual atoms and their interactions are ignored. Instead, fluids are characterized solely by macroscopic variables such as pressure, velocity, and density. As a consequence, continuum physics gives rise to so-called conservation laws. These laws state that certain measurable quantities cannot change over time in an isolated physical system. These laws, in turn, can capture the complete evolution of a fluid system.

2.1.1. Conservation Laws

One of these conservation laws states that within a control volume, no mass can be created or destroyed. This implies that any change in mass within a particular control volume can only result from mass entering or leaving it. This principle leads to the so-called continuity equation, described by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0\,, \tag{2.1}$$

where ρ denotes the density of the control volume, and $\mathbf{u}$ its velocity [82]. A second important conservation law is conservation of momentum. Similar to mass conservation, momentum within a control volume is considered. In this case, any change in momentum must stem from momentum entering the system, stresses at the control volume's boundary, or body forces acting upon it. This yields the Cauchy momentum equation, formulated as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{1}{\rho} \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}\,. \tag{2.2}$$

Here, $\boldsymbol{\sigma}$ denotes the Cauchy stress tensor in units of N/m^2, and $\mathbf{f}$ denotes the specific bodyforce in units of N/kg [75]. Assuming that we are dealing with a Newtonian fluid, where viscosity is independent of any applied shear stresses, and assuming an incompressible flow in which the material density of each fluid element does not change with time (expressed simply as $\frac{D\rho}{Dt} = 0$, with $\frac{D}{Dt}$ denoting the material derivative), the momentum equation simplifies considerably. This leads us to the well-known Navier-Stokes Equations (NSE) for incompressible Newtonian fluids, represented as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}\,, \tag{2.3}$$

where ν denotes the kinematic viscosity of the fluid. Together, the continuity and the NSE form a complete set of equations for describing of incompressible Newtonian fluids [82].

2.1.2. Thermal Hydraulics

In addition to mass and momentum, a conservation law can also be formulated for energy. In this case, the energy within a control volume can only change due to energy entering or leaving the volume (in the form of heat or work) or through energy loss or generation occurring within the volume. By assuming that the rate of change of the internal energy is proportional to the rate of change of its temperature and by applying Fourier's law, which states that the rate of heat transfer is proportional to the negative gradient of temperature [65], the energy balance equation can be rewritten into the heat equation, given by

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T = \frac{\kappa}{\rho C_p} \nabla^2 T + \frac{q}{\rho C_p} \,. \tag{2.4}$$

Here T denotes the temperature, κ the thermal thermal conductivity, C_p the specific heat capacity, and q a temperature source or sink term [71]. In this equation, the time rate of temperature change (first term) is described by three processes, namely convection, diffusion, and external sources (second to fourth term respectively).

Note that the heat equation is coupled to the Navier-Stokes equations through the velocity field $\mathbf{u}$ in the convection term. To describe a two-way coupled thermal flow, where the velocity field influences the temperature field and vice-versa, the temperature must be coupled back to the momentum equation. In the case of a buoyancy-driven flow, such a formulation can be obtained using the Boussinesq approximation. This approximation states that only variations in the density ρ are considered and that all other fluid properties are held constant [100]. Furthermore, the variations in density are modeled using a reference density and a correction term that is linearly dependent on temperature, formulated as

$$\rho = \rho(T_0)[1 - \beta_{th}(T - T_0)] \,. \tag{2.5}$$

Here β_{th} denotes the thermal expansion coefficient in units of K^{-1}, which relates temperature deviations to the expansion of the fluid, and T_0 is a reference temperature. Additionally, the Boussinesq approximation states that density variations are only important in the buoyancy force term in the NSE. As a consequence, the NSE can be rewritten as [100]

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho_0} \nabla (p - \rho_0 \mathbf{g} \cdot \mathbf{z}) + \nu \nabla^2 \mathbf{u} - \mathbf{g} \beta_{th} (T - T_0) \,. \tag{2.6}$$

Together with the continuity equation (2.1) and the heat equation (2.4) these equations are called the Boussinesq equations, and they form the bases for the description of incompressible buoyancy-driven Newtonian fluids.

2.1.3. Dimensionless Numbers

Dimensionless numbers are a powerful tool in fluid mechanics that enable the evaluation of fluid behavior using ratios between key fluid and system characteristics [74]. Their significance can be demonstrated by performing a non-dimensional analysis of the NSE from Equation 2.3. This is achieved by substituting the variables in the equation with their dimensionless counterparts, defined as

$$\tilde{\mathbf{x}} = \frac{\mathbf{x}}{L} \,, \qquad \tilde{t} = \frac{t}{T} \,, \qquad \tilde{\mathbf{u}} = \frac{\mathbf{u}}{U} = \frac{\mathbf{u}T}{L} \,, \qquad \tilde{p} = \frac{p}{P} = \frac{pT^2}{\rho L^2} \,, \qquad \text{etc.} \tag{2.7}$$

Here, L, T, and U represent the characteristic length, time, and velocity scales of the fluid and the problem domain, respectively. By introducing these non-dimensional variables, the NSE can be rewritten as

$$\frac{\partial \tilde{\mathbf{u}}}{\partial \tilde{t}} + \tilde{\mathbf{u}} \cdot \nabla \tilde{\mathbf{u}} = -\frac{1}{\tilde{\rho}} \nabla \tilde{p} + \frac{1}{Re} \tilde{\nu} \nabla^2 \tilde{\mathbf{u}} + \tilde{\mathbf{f}} \,. \tag{2.8}$$

In this form, we have introduced the Reynolds number, a dimensionless quantity defined as

$$Re = \frac{UL}{\nu} \,. \tag{2.9}$$

This dimensionless version of the NSE reveals the significance of the Reynolds number in evaluating fluid behavior. Specifically, a high Reynolds number indicates that the convective term (second term on the left) dominates over the diffusive term (second term on the right). Since the convective term is non-linear, this dominance results in chaotic and irregular flow patterns, ultimately leading to turbulence. In contrast, a low Reynolds number implies that the diffusive term dominates, resulting in a smooth and stable flow, known as laminar flow.

This example highlights the power of dimensionless numbers. They can be easily calculated from system and fluid properties, yet they provide significant insights into fluid behavior. In addition to this, another key advantage of dimensionless numbers is that they allow us to compare flows that appear to operate on different physical scales. This is achieved through the principle of dynamic similarity, which states that if the governing dimensionless numbers of two systems are identical, their fluid behaviors will be the same [74].

In addition to the Reynolds number, several other dimensionless numbers are used in this research. Table 2.1 provides a comprehensive overview of these dimensionless numbers, including their mathematical definitions and physical interpretations.

Table 2.1: This section provides an overview of the dimensionless numbers used in this research. It includes their mathematical definitions in terms of fluid and system characteristics, along with a physical interpretation of each dimensionless number and its application in evaluating fluid behavior. In this context, L, U, and ΔT represent the characteristic length, velocity, and temperature scales of the system, respectively. The symbols ν, α, D, and β_{th} refer to the fluid's viscosity, thermal diffusivity, molecular diffusivity, and thermal expansion coefficient. Additionally, c_s denotes the speed of sound in the fluid, and g represents Earth's gravitational acceleration.

Dimensionless number	Definition	Physical interpretation
Reynolds	$Re = \frac{UL}{\nu}$	Measures the ratio between inertial and viscous forces acting on the fluid, and determines whether the fluid will be laminar or turbulent.
Prandtl	$Pr = \frac{\nu}{\alpha}$	Measures the ratio between the viscosity of a fluid and its thermal conductivity. It compares the rate at which momentum diffuses through a fluid versus the rate at which heat diffuses.
Schmidt	$Sc = \frac{\nu}{D}$	Measures the ratio between the viscosity of a fluid and its mass diffusivity. It compares the rate at which momentum diffuses through a fluid versus the rate at which mass (or species) diffuses.
Rayleigh	$Ra = \frac{\beta_{th} L^3 g \Delta T}{\nu\alpha}$	Measures the ratio between buoyancy forces and viscous and thermal diffusion effects. It quantifies the balance between buoyancy and diffusive forces acting on the fluid.
Nusselt	$Nu = \frac{hL}{\kappa}$	Measures the ratio between total heat transfer (convection + conduction) and conductive heat transfer.

2.1.4. Kinetic Theory

Fluids can be understood from different perspectives. In the previous section, the continuum approximation was used to describe a fluid on a macroscopic scale, using fluid quantities such as pressure and velocity. However Newtonian fluids can also be described on a microscopic scale, by tracking the interaction between individual particles or molecules using Newtonian mechanics. Kinetic theory takes the middle route and focuses on the in-between mesoscopic scale. Rather than describing individual particles, Kinetic theory studies how particle densities move within the fluid [55]. These particle densities, also called particle distributions, are given by $f(\mathbf{r}, \boldsymbol{\xi}, t)$, which denotes the density of particles moving with velocity $\boldsymbol{\xi} = (\xi_x, \xi_y, \xi_z)$ at position $\mathbf{r} = (x, y, z)$ and time t. Furthermore, these particle distributions can be related to macroscopic variables through their moments, which are calculated as [55]

$$\begin{aligned} \rho(\mathbf{r},t) &= \int f(\mathbf{r},\boldsymbol{\xi},t)\mathrm{d}^3\xi\,, \\ \rho(\mathbf{r},t)\mathbf{u}(\mathbf{r},t) &= \int \boldsymbol{\xi} f(\mathbf{r},\boldsymbol{\xi},t)\mathrm{d}^3\xi\,, \\ \rho(\mathbf{r},t)E(\mathbf{r},t) &= \frac{1}{2}\int |\boldsymbol{\xi}|^2 f(\mathbf{r},\boldsymbol{\xi},t)\mathrm{d}^3\xi\,. \end{aligned} \tag{2.10}$$

Kinetic theory investigates the evolution of these particle densities over time. As f is a function of position $\mathbf{r}$, particle velocity $\boldsymbol{\xi}$, and time t, its derivative with respect to time is equal to

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \left(\frac{\partial f}{\partial t}\right)\frac{\mathrm{d}t}{\mathrm{d}t} + \left(\frac{\partial f}{\partial r_i}\right)\frac{\mathrm{d}r_i}{\mathrm{d}t} + \left(\frac{\partial f}{\partial \xi_i}\right)\frac{\mathrm{d}\xi_i}{\mathrm{d}t}\,. \tag{2.11}$$

By replacing df/d$t = \Omega(f)$, dt/d$t = 1$, dr_i/d$t = \xi_i$, and dr_i/dt = f$_i$ (the latter being the specific body force in units of N/Kg) and using vector notation we arrive at the Boltzmann equation, given by

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \cdot \nabla f + \nabla_{\boldsymbol{\xi}} f \cdot \mathbf{f} = \Omega(f) \,. \tag{2.12}$$

This equation explains how particle densities change over time, resembling an advection process [55]. The first two terms on the left show how the densities move through advection with a velocity $\boldsymbol{\xi}$, while the third term represents how an external force affects this velocity. On the right side, the collision operator $\Omega(f)$ serves as a source term, reflecting how particle collisions redistribute densities locally. Formally, the collision operator is a nonlinear integral operator that accounts for all changes in the distribution function f resulting from collisions between particles. Kinetic theory states that, due to the random nature of collisions within a fluid, the particle distributions converge over time towards a so-called equilibrium distribution $f^{eq}(\mathbf{r}, \boldsymbol{\xi}, t)$. This equilibrium distribution, known as the Maxwell-Boltzmann distribution [64], is isotropic in velocity space around the macroscopic velocity $\mathbf{u}$, and has the property $\Omega(f^{eq}) = 0$.

Through a so-called Chapman-Enskog analysis, the Navier-Stokes equations can be recovered from the Boltzmann equation. This process entails expanding the particle densities as a perturbative series, and then systematically equating terms of similar orders between the distribution function and the collision operator expansion [18]. By decomposing the particle distribution functions into an equilibrium and a non-equilibrium part, and using the relationships between the moments of the particle distributions given in Equation 2.10, this process results in the same conservation laws as described in Section 2.1.1, proving that the mesoscopic description of fluids through the evolution of particle densities correctly simulates the dynamics described by the Navier-Stokes equations.

2.2. Nuclear Reactor Physics

At the heart of nuclear reactor physics lies the principle that energy can be harnessed through the splitting of fissile Actinides like uranium-235 and plutonium-239. This splitting process, known as nuclear fission, generates energy from a difference in binding energy between the pre-split and post-split configuration of subatomic particles, resulting in a mass discrepancy between the two states. As described by Einstein's famous formula, this mass difference is converted into energy through the equation $E = mc^2$. Nuclear fission events can occur spontaneously in a decay process but can also be induced by irradiating the fissile isotopes with neutrons. In the latter case, the fissile nuclide first absorbs a neutron, resulting in a new compound state. This heavier nuclide has an increased instability, triggering a fission event. Although this process is inherently stochastic, and should therefore be described in a statistical framework, most fission events yield two new nuclei along with several neutrons. Within a nuclear reactor, the neutrons produced from initial fission events trigger subsequent fissions, initiating a chain reaction. When this chain reaction is regulated, it can sustainably generate energy by continuously burning up Actinides [30].

This regulation of the nuclear chain reaction is crucial for ensuring safe and efficient energy production in a nuclear reactor. For example, if too few neutrons are generated in the nuclear process, the chain reaction cannot sustain itself, resulting in the gradual shutdown of the reactor over time. Conversely, when too many neutrons are produced, the number of fission events will grow exponentially, introducing significant safety issues, as nuclear reactor cores can only transport limited energy. This leads to the definition of the multiplication factor k, calculated as the number of neutrons in one generation divided by the number of neutrons in the preceding generation. When $k = 1$, the reactor operates at a critical state, where each neutron, on average, spawns exactly one new neutron after fission, leading to a stable reactor core. In other cases when $k < 1$ or $k > 1$ the reactor is in a sub-critical or super-critical state respectively. In these cases the neutron population is a function of time, leading to an unstable reactor core. Therefore, nuclear reactor control is concerned with keeping the multiplication factor at unity during normal operating conditions [30].

To effectively monitor and control the multiplication factor, it is important to accurately assess the neutron population within the reactor core. This necessitates the use of neutronic calculations, employing a stochastic framework to compute average neutron densities by evaluating various neutron-nucleus interactions. Such an analysis results in the neutron transport equation, which will be elaborated on the next section. Key to these calculations is the use of nuclear macroscopic cross-sections Σ_x, which quantify the likelihood of specific nuclear reactions occurring [118]. These macroscopic cross-sections are derived from the product of microscopic cross-sections σ_x, which are material and energy-dependent parameters, and the atomic number density N of the material. By multiplying the macroscopic cross-section by the neutron density and velocity, we obtain the reaction rate of x (in terms of reactions/cm^3/sec), where x denotes the nuclear reaction type. These reaction types include scattering, fission, absorption, among others.

2.2.1. Neutron Transport

Similar to the conservation equations for fluid dynamics and thermodynamics described in Section 2.1, a balance equation for neutron transport can be derived. Following an analogous approach, this is achieved by analyzing a control volume and equating the time rate of change of the neutron population to the various processes that influence neutron transport into and out of the volume. This results in the so-called neutron transport equation (NTE), described as

$$\begin{aligned}\frac{1}{\mathrm{v}}\frac{\partial\varphi}{\partial t}+\hat{\mathbf{\Omega}}\cdot\nabla\varphi+\Sigma_t(\boldsymbol{r},E)\varphi(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)=&\int_{4\pi}\mathrm{d}\hat{\mathbf{\Omega}}'\int_0^\infty \mathrm{d}E'\Sigma_s(E'\to E,\hat{\mathbf{\Omega}}'\to\hat{\mathbf{\Omega}})\varphi(\boldsymbol{r},E',\hat{\mathbf{\Omega}}',t)\\&+S(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)\,.\end{aligned}\tag{2.13}$$

Here, $\varphi(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)$ denotes the angular neutron flux, which serves as a measure for the expected number of neutrons crossing a small unit area d^3r around $\boldsymbol{r}$, with an energy in the range E to $E+\mathrm{d}E$, and moving with a velocity pointing in the direction $\hat{\mathbf{\Omega}}$ within a solid angle $\mathrm{d}\hat{\mathbf{\Omega}}$, at time t. Furthermore, v describes the speed of the neutrons, $\Sigma_t(\boldsymbol{r},E)$ the total cross-section of the material, $\Sigma_s(E'\to E,\hat{\mathbf{\Omega}}'\to\hat{\mathbf{\Omega}})$ the differential scatter cross-section of neutrons scattering from energy E' and angle $\mathbf{\Omega}'$ to E and $\mathbf{\Omega}$, and finally $S(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)$ acts as a source term [61]. Moving from left to right, the different terms of the equation denote the time rate of change of the neutrons, the transport of neutrons across the boundaries of the control volume, the loss of neutrons due to absorption or collision, the number of in-scattering neutrons from other energy levels and velocity directions, and finally a source term. The source term can be further decomposed into several contributors, one being the source of neutrons from fission reactions, given by

$$S_f(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)=\frac{\chi(E)}{4\pi}\int_{4\pi}\mathrm{d}\hat{\mathbf{\Omega}}'\int_0^\infty \mathrm{d}E'\nu(E')\Sigma_f(E')\varphi(\boldsymbol{r},E',\hat{\mathbf{\Omega}}',t)\,.\tag{2.14}$$

Here $\chi(E)$ denotes the energy spectrum of the emitted fission neutrons (acting as a distribution function), $\nu(E)$ the average number of neutrons emitted in a fission event as a function of energy, and $\Sigma_f(E)$ the fission cross-section as a function of energy. This formula denotes the production of neutrons with energy E by integrating over all fission events from neutrons with different energies and velocity directions. Note, that the formula assumes isotropic emission of the neutrons produced from fission, indicated by the $1/4\pi$ term.

2.2.2. Nuclear Reactor Kinetics

Equation 2.13 assumes that all neutrons are emitted directly after the fission event, in reality however, this is not the case. While most neutrons are released directly after fission, some of the produced fission nuclei, also emit neutrons through a decay process. These nuclei are called precursors, and their emission of neutrons takes a considerably longer time than the neutrons emitted in the fission event itself [30]. Therefore, a distinction should be made between neutrons obtained from the fission event (prompt neutrons) and neutrons originating from precursor decay (delayed neutrons) in the transport equation. In terms of mathematics, we say that a fraction β_i of the fission neutrons originate from the decay of precursor species C_i. This means that the remaining $1-\sum_i\beta_i$ fraction of neutrons are prompt. consequently, the NTE can be completed by adding a precursor source term for the delayed neutrons, turning Equation 2.13 into

$$\begin{aligned}\frac{1}{\mathrm{v}}\frac{\partial\varphi}{\partial t}+\hat{\mathbf{\Omega}}\cdot\nabla\varphi+\Sigma_t(\boldsymbol{r},E)\varphi(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)=&\int_{4\pi}\mathrm{d}\hat{\mathbf{\Omega}}'\int_0^\infty \mathrm{d}E'\Sigma_s(E'\to E,\hat{\mathbf{\Omega}}'\to\hat{\mathbf{\Omega}})\varphi(\boldsymbol{r},E',\hat{\mathbf{\Omega}}',t)\\&+\frac{(1-\beta)\chi^p(E)}{4\pi}\int_{4\pi}\mathrm{d}\hat{\mathbf{\Omega}}'\int_0^\infty \mathrm{d}E'\nu(E')\Sigma_f(E')\varphi(\boldsymbol{r},E',\hat{\mathbf{\Omega}}',t)\\&+\frac{\chi^d(E)}{4\pi}\sum_i\lambda_iC_i(\mathbf{r},t)+S^{ext}(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)\,.\end{aligned}\tag{2.15}$$

Here, $\beta=\sum_i\beta$ is the complete fraction of neutrons originating from precursor decay, λ_i denotes the decay constant of precursor species C_i, and χ^p and χ^d denote the neutron emission spectrum of the prompt neutrons and delayed neutrons respectively [61]. Similar to the prompt neutrons, the delayed neutrons are also assumed to be emitted isotropically by again introducing the $1/4\pi$ term. Note finally that an extra external source term $S^{ext}(\boldsymbol{r},E,\hat{\mathbf{\Omega}},t)$ is added for completeness.

While equation 2.15 correctly describes the neutron population of a reactor core, it does require a description of the precursor transport. This is especially important in the case of an MSFR, where the precursors will experience convective transport due to the velocity field of the liquideous fuel-salt. This precursor transport can be described using an advection-convection equation, given by

$$\frac{\partial C_i}{\partial t}+\nabla\cdot(\mathbf{u}C_i)+\lambda_iC_i(\mathbf{r},t)=\nabla\cdot(D_i\nabla C_i)+\beta_i\int_{4\pi}\mathrm{d}\hat{\mathbf{\Omega}}'\int_0^\infty \mathrm{d}E'\nu(E')\Sigma_f(E')\varphi(\boldsymbol{r},E',\hat{\mathbf{\Omega}}',t)\,.\tag{2.16}$$

Here, $\mathbf{u}$ denotes the velocity field of the fuel-salt, and D_i is known as the molecular diffusivity of the precursor family C_i in the fuel salt [30]. The usage of the indexation in C_i indicates that the equation only describes the transport of a single precursor family. From left to right, the terms are identified as the time rate of change, a convection term, a loss term due to decay, an advection term, and a fission source term. Note that the latter is similar to the fission term in Equation 2.15 but weighted by β_i for the fraction of precursors C_i produced in fission events. Together, Equations 2.15 and 2.16 capture the full neutronics of an MSFR nuclear reactor core.

2.3. Multiphysics of an MSFR Reactor Core

As discussed in the introduction of this report, in the design of the MSFR, the molten salt serves a dual purpose as both the fuel and the coolant. Consequently, simulating the reactor core requires accounting for both thermal fluid dynamics and neutronics, making it a multiphysics problem. Although thermal fluid dynamics and neutronics calculations operate on different time and length scales and are governed by separate physical laws, they cannot be computed independently due to their strong coupling through various processes. Most of these interactions have been mentioned in previous sections, but they are summarized here to emphasize the importance of integrating these coupling processes into the simulation code. Figure 2.1 provides a schematic overview of the different physical quantities and the processes through which they are interrelated.

Firstly, as discussed in Section 2.1, velocity and temperature are two-way coupled through the convection term in the heat equation and the buoyancy force in the NSE. Moreover, the velocity field is also coupled to the precursor field through the convective transport of precursor species. However, there is no direct process coupling the precursor density back to the velocity field. The precursor density is also directly two-way coupled with the neutron flux. As mentioned in Section 2.2, precursors introduce delayed neutrons through radioactive decay, while new fission reactions act as a source for precursors. Finally, there is a direct two-way coupling between temperature and neutron flux. Fission reactions generate heat, which serves as a power source in the heat equation. Conversely, temperature influences the neutron flux through changes in nuclear cross-sections. These changes are due to Doppler shifts, which can broaden resonance peaks in neutron-nucleus interactions, as well as density feedback effects.

These direct interactions also highlight the indirect influences among these quantities. For instance, an increase in neutron flux leads to more fission events, which raises the power output and, consequently, the temperature. The temperature rise then affects the velocity field via buoyancy forces. This altered velocity field modifies the convective transport of precursor species, which, in turn, influences the neutron flux. This creates a feedback loop where each component affects the others, illustrating the complex interdependencies within the system.

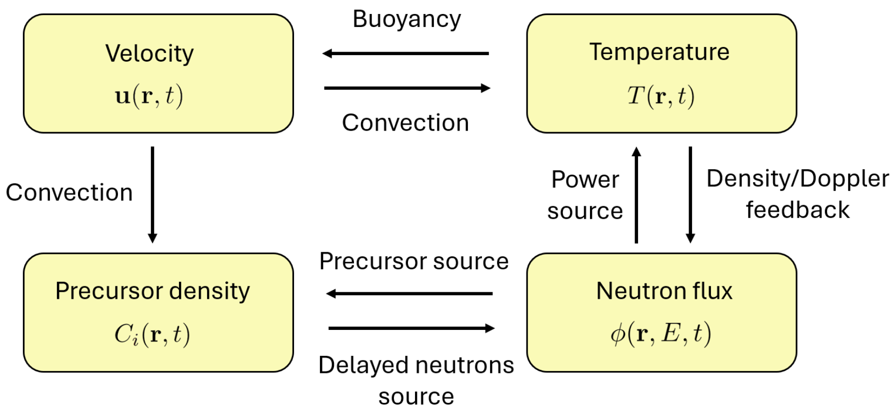

Figure 2.1: Schematic overview of the key physical quantities in MSFR reactor core simulations and the processes coupling these fields. This research employs the NSE to model velocity, the heat equation for temperature, the advection-convection equation for the precursor density, and the NTE for neutron flux.

2.4. Parallel Programming

The direct numerical simulation of thermal fluids using the NSE is notoriously computationally expensive [35]. To address this challenge, this research focuses on simulating thermal flows using a kinematic representation, leading to the adoption of the Lattice Boltzmann Method (LBM), a simulation technique that streamlines the simulation of thermal fluids by decomposing the process into a sequence of repetitive calculations. [55]. These calculations are performed locally at each grid point within the spatial discretization of the problem domain and are independent of one another. This independence enables the parallel execution of these calculations, significantly reducing the computational burden associated with fluid simulations.

In the case of LBM, the parallelized calculations are computationally small enough to be performed on a Graphics Processing Unit (GPU). A GPU contains thousands of cores that, although individually less powerful than a conventional CPU core, can perform calculations simultaneously in parallel. Originally designed to accelerate graphics rendering for video games, GPUs are now widely used for high-performance computing (HPC), a practice known as general-purpose computing on GPUs (GPGPU). This technique finds applications in various fields, including machine learning [63, 77, 87], medical imaging [5, 32, 67], and scientific computing [25, 49, 78].

2.4.1. CUDA Programming Language

Parallel programming involves writing algorithms that allow different segments of code to execute simultaneously across multiple processor cores. In this research, the Compute Unified Device Architecture (CUDA) programming language is used to implement such an algorithm. CUDA, developed by NVIDIA, is a parallel computing platform that allows developers to leverage NVIDIA GPUs for general-purpose processing [27]. Within this framework, pieces of code that should run independently on each core can be defined. These pieces of code, known as kernel functions, are sent to the GPU to run independently on each core. Generally speaking, this leads to the following procedure for parallel programming using CUDA

1. Transfer input variables from CPU memory to GPU memory.
2. Allocate GPU memory for simulation output.
3. Perform calculations on the GPU by sending kernel functions from the CPU to the GPU.
4. Transfer the simulation results from GPU memory back to CPU memory.

When using CUDA for parallel programming, several challenges related to GPGPU must be considered. Firstly, the computational cores of a GPU are generally less powerful than those of a conventional CPU core. As a result, it is crucial to minimize the complexity of computations performed within kernel functions. This can be achieved by adopting modular code design, where each kernel function performs a single, well-defined task, thereby reducing the computational load within each kernel. Additionally, the complexity of the calculations should be minimized by avoiding costly operations like loops whenever possible. Lastly, using single-precision floating-point numbers is recommended, as this in theory can halve the computational burden compared to double-precision floats.

Secondly, in the CUDA framework, only a limited set of operations is supported within a kernel function. While standard operations like addition, multiplication, and division are available, along with basic constructs such as loops, more advanced operations like matrix manipulations are not natively provided. Consequently, operations such as inner products and matrix multiplications must be manually implemented using loops.

Lastly, GPU programming with kernel functions relies on the assumption that calculations and memory addresses accessed by the kernels are independent. When this independence is not maintained, race conditions may arise, where multiple kernels on different cores access or modify the same memory location simultaneously. In such situations, CUDA does not generate an error, but the results can become distorted, making debugging challenging. To prevent this issue, it is crucial to use available features such as atomic operations and synchronization functions to ensure proper coordination among kernels and avoid data conflicts.

2.4.2. GPU Hardware Architecture

The CUDA programming language is designed exclusively for general-purpose computing on NVIDIA GPUs. Therefore, it is important to discuss the hardware architecture of NVIDIA GPUs, as this structure has certain implications for the software abstractions used in these GPUs. As illustrated in Figure 2.2, NVIDIA GPUs are organized in a layered architecture designed to optimize memory handling and processing efficiency.

At the top layer, the GPU is subdivided into streaming multiprocessors (SMs), which are individual processing units responsible for the parallel execution of tasks, and a block of global memory accessible by all computing cores. Within each SM, additional specific structures can be identified at the second layer. Each SM contains its own block of shared memory, accessible only by the cores within that specific SM. Additionally, each SM includes a warp scheduler, a component dedicated to scheduling the execution of the CUDA cores, which will be discussed in more detail in a later section. Finally, the SM contains several thousand CUDA cores that perform the actual computations. At the finest layer, each CUDA core contains the necessary hardware to perform computations along with its own block of register memory, which is used solely by that single computational unit.

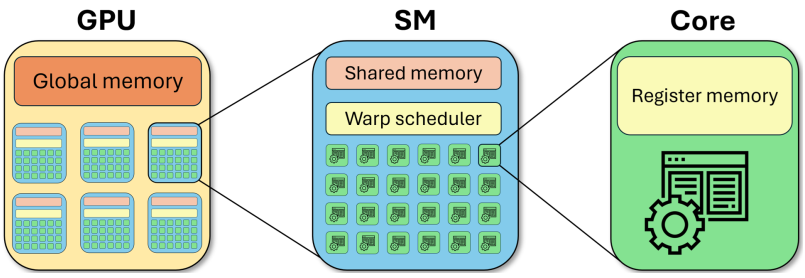

Figure 2.2: Hardware architecture of an NVIDIA GPU. This figure depicts the layered structure of the GPU, optimized for efficient memory management. At the top layer, the GPU is divided into multiple SMs and a block of global memory. Each SM, in turn, contains its own block shared memory and several thousand CUDA cores. At the most granular level, each CUDA core functions as an individual processing unit with its own dedicated register memory.

2.4.3. GPU Memory Hierarchy

The layered structure of the NVIDIA GPU allows for different types of memory, leading to a so-called memory hierarchy. As shown in Figure 2.2, different types of memory can be accessed by different groups of computational cores. For example, global memory can be accessed by all cores, while shared memory is limited to cores within the same SM. Although it might seem disadvantageous to have memory accessible by only a subset of cores, this design actually enhances memory access speed. For example, as the shared memory block is situated closer to the CUDA cores, it takes less time to read and write data. The same principle applies to register memory, while its accessibility is limited to a single computational core, its proximity allows for extremely fast data transfers. In addition to the memory access speed, the different types of memory also have varying capacities. Register memory has the smallest capacity as it is dedicated to a single core, while global memory offers the largest storage due to its broad accessibility. Consequently, this memory hierarchy creates a trade-off between accessibility, capacity, and access speed, which must be carefully balanced when developing code for simulations. Optimal storage locations should be chosen based on maximizing transfer speed while adhering to accessibility and capacity constraints. Within the CUDA programming framework, there are two more types of memory—distinct memory addresses within global memory. To provide a complete overview, a full summary of the memory hierarchy is given in Table 2.2.

2.4.4. CUDA Software Abstractions

In CUDA programming, the problem space is organized into a grid for parallel computation, with calculations performed at each grid point. Efficient scheduling of kernel launches across these grid points is managed through specific software abstractions within the CUDA framework, as illustrated in Figure 2.3.

Table 2.2: Comprehensive overview of the different memory addresses in an NVIDIA GPU. The GPU's layered architecture results in a hierarchical memory structure, with each type of memory offering distinct advantages and disadvantages in terms of accessibility, capacity, and access speed. Consequently, when developing code, it is essential to consider optimal storage locations to balance these factors effectively.

Memory type	Description
Global memory	Main storage of the GPU, is accessible for all CUDA cores as well as the CPU. Large memory storage, however slow information transfer.
Constant memory	Small amount of memory within global memory, that is read-only. Again it is accessible for all CUDA cores, however, the fact that it is read-only allows for fast reading. CPU can write to this piece of memory before the kernel launches.
Shared memory	Relatively small amount of memory specific to a single SM. Only cores located on the SM can access this memory, however, they can do so at a high speed.
Register memory	Smallest amount of memory, specific to a single CUDA core. As it is located on the core itself it is the fastest type of memory.
Local memory	Located within the global memory block, and specifically assigned to a single CUDA core. This memory is used when the register memory is fully occupied. Compared to register memory, the data transfer is very slow, however the capability is much larger.

Initially, the grid is divided into blocks, with each block assigned to a single SM. This subdivision determines how portions of the problem space are distributed among the SMs. Each block is further divided into warps, each consisting of 32 threads. A thread is the smallest unit of computation, performing operations for a single grid point. Warps group 32 threads together for efficient scheduling, a process that will be discussed further in the next section. CUDA allows the specification of both the number of blocks and the number of threads per block, which are crucial for simulation efficiency. While the total number of threads must cover the entire problem domain, the configuration of blocks and threads per block significantly impacts simulation performance.

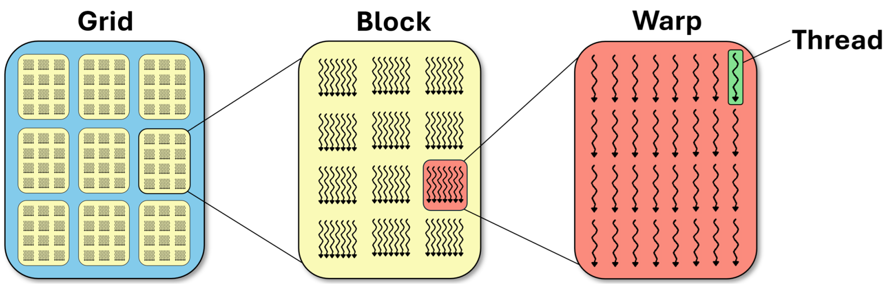

Figure 2.3: Overview of the software abstractions used in CUDA programming. While the number of threads in a warp is fixed at 32, CUDA allows for the specification of both the number of blocks and the number of threads per block. Optimizing the number of blocks and threads per block is essential for achieving balanced thread distribution across SMs, which maximizes computational performance.

Optimizing the number of blocks and threads per block is essential for performance. Since multiple blocks are assigned to a single SM and the number of SMs is limited, the goal is to balance the distribution of threads across the SMs. Achieving this often requires trial and error, involving testing various configurations and evaluating their impact on simulation speed to determine the most effective setup.

2.4.5. Warp Scheduling Process

The warp scheduler, as illustrated for a single SM in Figure 2.4, oversees the scheduling of calculations within an SM. Multiple blocks are assigned to each SM, and the warp scheduler manages the execution of these blocks by extracting warps from them and distributing the threads to the CUDA cores. When 32 cores are available, the warp scheduler assigns a new warp from the block to these cores. Once a block is fully processed, the scheduler proceeds to the next block.

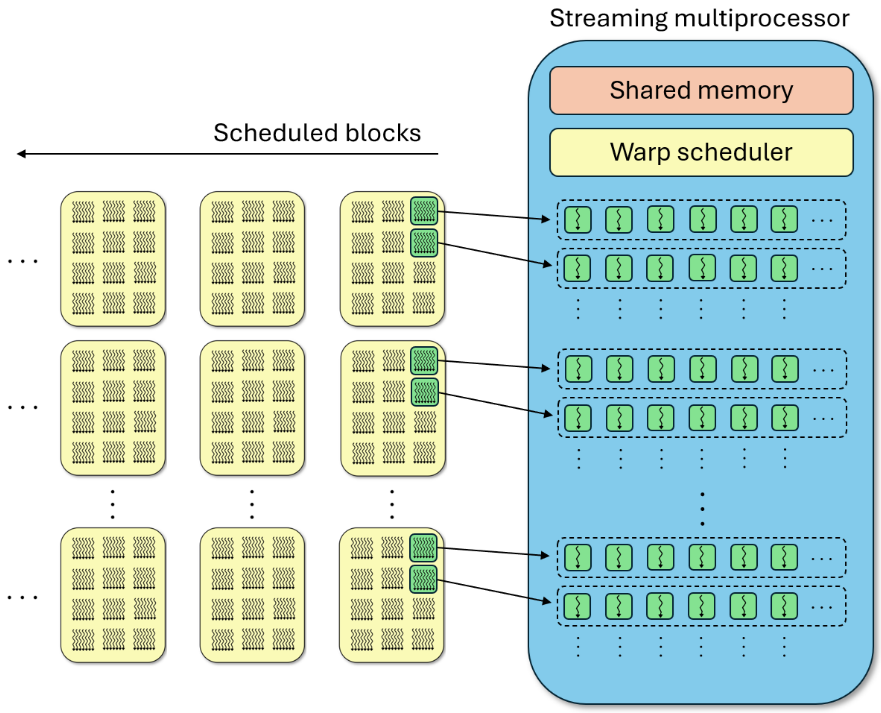

Figure 2.4: Illustration of the warp scheduling process on a single SM. The warp scheduler manages all warps from the blocks assigned to that SM, scheduling them in full rather than scheduling individual threads.

This scheduling process has two significant implications. First, shared memory can only be allocated for threads within the same block. The warp scheduler handles all warps within a block, ensuring that these threads have access to the shared memory within the same SM, while warps from different blocks may be allocated to different SMs. Second, since the warp scheduler schedules entire warps rather than individual threads, it is most efficient to define the number of threads per block as a multiple of 32. If the number of threads is not a multiple of 32, the warp scheduler will include inactive threads in the final warp. For instance, with 35 threads in a block, the second warp will contain 29 inactive threads. These inactive threads occupy CUDA cores that could have been used for active computations, leading to inefficient resource utilization.

3 Numerical Method

This chapter presents the numerical techniques and algorithms applied in this research. To simulate the complex multiphysics of an MSFR reactor core, a combination of two algorithms is employed. Section 3.1 provides a detailed description of the Filter Matrix Lattice Boltzmann Method (FM-LBM), which is utilized to simulate thermal flows and precursor transport. Moreover, Section 3.2 details how GPU acceleration is integrated into the FM-LBM algorithm to achieve parallel computing, alongside various techniques aimed at optimizing its performance. Section 3.3 introduces the Phantom-S_N algorithm, an in-house algorithm used for performing neutronics simulations. When combined with the FM-LBM algorithm, these two simulation tools capture the full multiphysics of the reactor core. Finally, Section 3.4 explores the coupling between the algorithms, explaining the mathematical techniques used to overcome the differential spatial discretizations of the two approaches, and detailing the process of information exchange between the two codes.

3.1. Thermal Fluid Simulation

To simulate thermal fluid dynamics, this research employs the Filter-Matrix Lattice Boltzmann Method (FM-LBM). This algorithm extends the traditional Lattice Boltzmann Method (LBM), providing greater stability compared to conventional LBM algorithms. LBM is a widely used numerical technique that has gained increasing scientific interest since the 1980s [55]. While originally developed for solving fluid mechanics problems [3, 22], LBM has found applications in a variety of other fields, including quantum mechanics [90, 89] and image processing [17, 50].

3.1.1. Lattice Boltzmann Method

The Lattice Boltzmann Equation (LBE) is derived from the Boltzmann equation for fluid kinetics through discretization over velocity, space, and time. This involves dividing the problem space into lattice points and the velocity space into a set of discrete velocities $\mathbf{c}_i$. These discrete velocities are specifically chosen such that particle densities move from one lattice point to another at each time step. Consequently, particle densities at each lattice point can be represented by a discrete set of distribution functions $\{f_i\}$, where each distribution function f_i is the product of the continuous particle density function moving in the direction $\mathbf{c}_i$ and a weight specific to that direction, expressed as $f_i(\mathbf{x}, t) = w_i f(\mathbf{x}, \mathbf{c}_i, t)$. By neglecting the force term in Equation 2.12 this discretization leads to the LBE, given by

$$f_i(\mathbf{x} + \mathbf{c}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = \Delta t \, \Omega_i(f) \, . \tag{3.1}$$

This equation describes the evolution of the particle densities $f_i(\mathbf{x}, t)$ moving in the direction $\mathbf{c}_i$ at each time step Δt. Similar to the relations provided in Equation 2.10, these distribution functions enable the calculation of macroscopic quantities at each lattice point by summing over the different velocity directions. For example, at an arbitrary lattice point, the density and the velocity of the fluid can be calculated using

$$\rho = \sum_i f_i \, , \qquad \rho \mathbf{u} = \sum_i f_i \mathbf{c}_i \, . \tag{3.2}$$

Note that, Equation 3.1 is not uniquely defined. First of all, we haven't said anything about the velocity set $\{\mathbf{c}_i\}$ and corresponding weights $\{w_i\}$. This is because multiple possibilities exist for these so-called schemes. By convention these schemes are denoted as $\mathrm{D}d\mathrm{Q}q$, where d denotes the dimensionality of the problem space and q indicates the number of velocity directions. To illustrate differences between such schemes, Figure

3.1 gives a visual representation of some of the possible schemes in 2D and 3D. Note that, more possible schemes exist in both 2D and 3D problem spaces.

The second parameter in Equation 3.1 that is not uniquely defined is the collision operator $\Omega(f)$. As described in Section 2.1.4, this operator accounts for the redistribution of particle densities after collision events. Furthermore, in the limit, it should relax the particle density functions towards the equilibrium distribution, given by the Maxwell-Boltzmann distribution. In the discrete framework, this equilibrium distribution is given by

$$f_i^{eq} = w_i \rho \left(1 + \frac{\mathbf{u} \cdot \mathbf{c}_i}{c_s^2} + \frac{(\mathbf{u} \cdot \mathbf{c}_i)^2}{2c_s^4} - \frac{\mathbf{u} \cdot \mathbf{u}}{2c_s^2} \right). \tag{3.3}$$

Here, ρ and $\mathbf{u}$ denote the macroscopic density and velocity, which can be obtained through the relationships given in Equation 3.2, w_i denotes the weight corresponding to velocity direction $\mathbf{c}_i$, and c_s is the lattice speed of sound, which is defined through the isothermal equation of state $p = c_s^2 \rho$. Chapman-Enskog analysis confirms that by representing the equilibrium distribution in the form of Equation 3.3, the discrete framework of the Lattice Boltzmann equation correctly solves the force-free Navier-Stokes equations in the limit [55].

Different collision operators exist to guide particle distributions towards equilibrium, and the choice of this operator distinguishes various Lattice Boltzmann algorithms. The simplest among them, the Bhatnagar-Gross-Krook (BGK) operator [8], relaxes distribution functions towards equilibrium using a single relaxation parameter, τ, and is defined as

$$\Omega_{\mathrm{BGK}}(f) = -\frac{1}{\tau}(f_i - f_i^{eq}). \tag{3.4}$$

Despite its simplicity and ease of implementation, the BGK collision operator's performance is unsatisfactory, often leading to unstable results [66]. Hence, more sophisticated collision operators such as the two-relaxation-times operator (TRT) [41], the multi-relaxation-times operator (MRT) [16], and Filter-Matrix operator (FM) [121, 123] have been introduced. This research uses the latter, the specifics of which will be discussed in a later section.

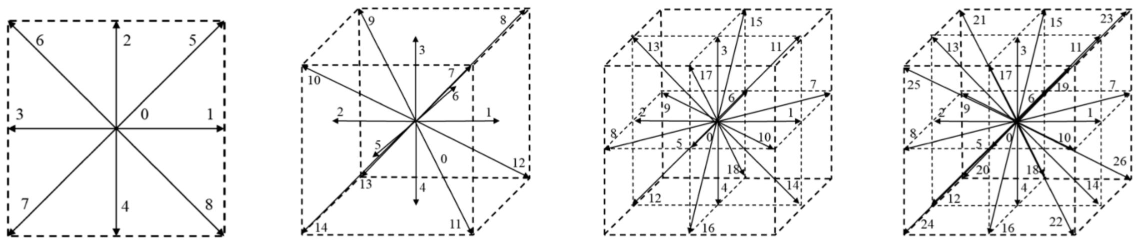

Figure 3.1: Examples of velocity schemes used in LBM simulations, characterized by their dimensionality, d, and the number of velocity directions, q. As can be seen from the figure, a higher q indicates greater interdependence between adjacent lattice points. This figure is obtained from [91].

Streaming and Collision

A closer look at the LBE reveals that the evolution of the density functions can be simulated using a combination of two steps.

1. Collision step: The redistribution of the density functions on the lattice node through the collision operator,

$$f_i^*(\mathbf{x}, t) = f_i(\mathbf{x}, t) + \Delta t \, \Omega_i(f). \tag{3.5}$$

   Here, $f_i^*(\mathbf{x}, t)$ denotes the post-collision distribution functions.

2. Propagation step: The streaming of particle distributions to neighboring lattice nodes,

$$f_i(\mathbf{x} + \mathbf{c}_i \Delta t, t + \Delta t) = f_i^*(\mathbf{x}, t). \tag{3.6}$$

These steps, visually represented in Figure 3.2, illustrate one of the key strengths of LBM, its simplicity. Fluid simulation can be effectively performed by simply alternating between these two straightforward steps.

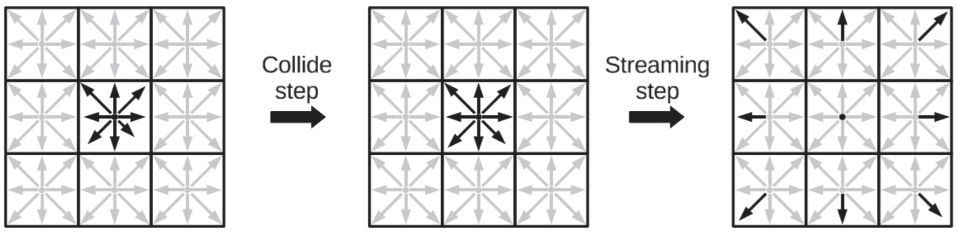

Figure 3.2: Visual representation of the LBM algorithm, which consists of two steps: collision and propagation. Fluid simulations can be performed by alternating between these two simple processes. This figure is adapted from [72].

3.1.2. Filter-Matrix Lattice Boltzmann Method

The Filter-Matrix Lattice Boltzmann Method (FM-LBM) can be derived from the LBE by redefining the spatial and temporal dimensions using a staggered grid approach [121]. This involves offsetting the space dimension by half a grid spacing and the time dimension by half a time step. As a result, Equation 3.1 is reformulated as

$$f_i\left(\mathbf{x}+\frac{\mathbf{c}_i\Delta t}{2}, t+\frac{\Delta t}{2}\right) = f_i\left(\mathbf{x}-\frac{\mathbf{c}_i\Delta t}{2}, t-\frac{\Delta t}{2}\right) + \Delta t\,\Omega_i(f)\,. \tag{3.7}$$

Using a Taylor expansion around $f_i(\mathbf{x},t)$ and by filling in the staggered representation of the LBE given in Equation 3.7, we obtain

$$f_i\left(\mathbf{x}\pm\frac{\mathbf{c}_i\Delta t}{2}, t\pm\frac{\Delta t}{2}\right) = f_i(\mathbf{x},t) \pm \frac{\Delta t}{2}\Omega_i(f) + \mathcal{O}\left(\Delta t^2\right)\,. \tag{3.8}$$

Subsequently, Chapman-Enskog analysis should be applied to Equation 3.8. As described in Section 2.1.4, this involves decomposing the particle density functions into an equilibrium part, given by Equation 3.3, and a non-equilibrium part. This analysis allows for the derivation of specific expressions for both the particle distributions f_i and the collision operator Ω_i in terms of macroscopic variables. These expressions ensure that the staggered formulation of the LBE correctly simulates incompressible flows within the framework of the NSE, and are given by

$$f_i(\mathbf{x},t) = \rho\omega_i\left[1+\frac{\mathbf{c}_i\cdot\mathbf{u}}{c_s^2}+\frac{1}{2}\left(\frac{(\mathbf{c}_i\cdot\mathbf{u})^2}{c_s^4}-\frac{\mathbf{u}\cdot\mathbf{u}}{c_s^2}\right)-\nu\left(\frac{(\mathbf{c}_i\cdot\nabla)(\mathbf{c}_i\cdot\mathbf{u})}{c_s^4}-\frac{\nabla\cdot\mathbf{u}}{c_s^2}\right)\right], \tag{3.9}$$

$$\Omega_i(f) = \frac{\rho w_i}{c_s^2}\left[(\mathbf{c}_i\cdot\nabla)(\mathbf{c}_i\cdot\mathbf{u}) - c_s^2\nabla\cdot\mathbf{u} + \mathbf{c}_i\cdot\mathbf{f}\right]\,. \tag{3.10}$$

Here, ν denotes the kinematic viscosity of the fluid, and $\mathbf{f}$ represents the specific body force, both of which can be found in the NSE formulated in Equation 2.3. By substituting these relationships back into Equation 3.8, the staggered particle distributions can be expressed as

$$\begin{aligned} f_i\left(\mathbf{x}\pm\frac{\mathbf{c}_i\Delta t}{2}, t\pm\frac{\Delta t}{2}\right) = \rho\omega_i\Bigg[& 1+\frac{\mathbf{c}_i\cdot\mathbf{u}}{c_s^2}+\frac{1}{2}\left(\frac{(\mathbf{c}_i\cdot\mathbf{u})^2}{c_s^4}-\frac{\mathbf{u}\cdot\mathbf{u}}{c_s^2}\right) \\ & -\nu\left(\frac{(\mathbf{c}_i\cdot\nabla)(\mathbf{c}_i\cdot\mathbf{u})}{c_s^4}-\frac{\nabla\cdot\mathbf{u}}{c_s^2}\right) \\ & \pm\frac{\Delta t}{2}\left(\frac{(\mathbf{c}_i\cdot\nabla)(\mathbf{c}_i\cdot\mathbf{u})}{c_s^2}-\nabla\cdot\mathbf{u}+\frac{\mathbf{c}_i\cdot\mathbf{f}}{c_s^2}\right)\Bigg]\,. \end{aligned} \tag{3.11}$$

This representation of the particle distributions can be written more concisely as a matrix multiplication by introducing the so-called filter matrix E_{ik} and the solution vector $\alpha_k^{\pm}(\mathbf{x},t)$ as

$$f_i\left(\mathbf{x}\pm\frac{\mathbf{c}_i\Delta t}{2}, t\pm\frac{\Delta t}{2}\right) = \sum_k w_i E_{ik}\alpha_k^{\pm}(\mathbf{x},t)\,. \tag{3.12}$$

Here, the filter matrix contains terms corresponding to microscopic velocities $\mathbf{c}_i$, and the solution vector contains terms corresponding to macroscopic variables such as ρ, $\mathbf{u}$, ν, and $\mathbf{f}$. The exact composition of the filter matrix and the solution vector depends on the specific scheme used and will be elaborated on shortly for the

D3Q19 scheme. The relationship in Equation 3.12 can also be inverted by introducing the matrix E_{ki}, which is related to the filter matrix through $w_i E_{ik} = (E_{ki})^{-1}$, leading to

$$\alpha_k^{\pm}(\mathbf{x},t) = \sum_i E_{ki} f_i \left(\mathbf{x} \pm \frac{\mathbf{c}_i \Delta t}{2}, t \pm \frac{\Delta t}{2}\right) . \tag{3.13}$$

This concludes the derivation of the FM-LBM method. Fundamentally, the algorithm remains unchanged, still comprising a streaming and a collision step. However, in the filter matrix algorithm, the collision operator now involves two matrix multiplications. First, the solution vector $\alpha_k^-(\mathbf{x},t)$ is computed from the pre-collision particle distributions using Equation 3.13. Subsequently, the post-collision particle densities are obtained by transforming the solution vector $\alpha_k^-(\mathbf{x},t) \to \alpha_k^+(\mathbf{x},t)$, and using the matrix multiplication given in Equation 3.12.

D3Q19 Filter Matrix

As shown in Figure 3.1, various schemes exist for 3D fluid simulation, including the D3Q15, D3Q19, and D3Q27 schemes. The D3Q27 scheme allows for more interdependence between neighboring nodes than the D3Q15, resulting in greater stability and accuracy. However, the D3Q15 scheme requires significantly less computational power for each time iteration, as calculations are performed for only 15 directions instead of 27. Therefore, selecting a scheme involves a trade-off between stability, accuracy, and simulation speed. In this research, the D3Q19 scheme is employed to simulate the velocity field, as it strikes a balance between accuracy and computational efficiency. In the context of the D3Q19 scheme, the filter matrix E_{ki} is defined as

$$\begin{aligned} E_{ki} = \Big[& 1, c_{ix}, c_{iy}, c_{iz}, 3c_{ix}^2 - 1, 3c_{iy}^2 - 1, 3c_{iz}^2 - 1, \\ & 3c_{iy}c_{iz}, 3c_{ix}c_{iz}, 3c_{ix}c_{iy}, 3c_{ix}\left(c_{iy}^2 - c_{iz}^2\right), 3c_{iy}\left(c_{iz}^2 - c_{ix}^2\right), \\ & 3c_{iz}\left(c_{ix}^2 - c_{iy}^2\right), c_{ix}\left(3c_{iy}^2 + 3c_{iz}^2 - 2\right), c_{iy}\left(3c_{ix}^2 + 3c_{iz}^2 - 2\right), \\ & c_{iz}\left(3c_{ix}^2 + 3c_{iy}^2 - 2\right), 3\left(2c_{ix}^2 - c_{iy}^2 - c_{iz}^2\right)\left(|c_i|^2 - \frac{3}{2}\right), \\ & 3\left(c_{iy}^2 - c_{iz}^2\right)\left(|c_i|^2 - \frac{3}{2}\right), 3|c_i|^2\left(|c_i|^2 - 2\right) + 1\Big]^{\top} . \end{aligned} \tag{3.14}$$

Note that this matrix can be constructed at the beginning of the simulation solely from the velocity set $\{\mathbf{c}_i\}$. The corresponding 19-speed solution vector $\alpha_k^{\pm}$ is given by

$$\alpha_k^{\pm} = \begin{bmatrix} \rho \\ \rho(u_x \pm \Delta t f_x/2) \\ \rho(u_y \pm \Delta t f_y/2) \\ \rho(u_z \pm \Delta t f_z/2) \\ 3\rho u_x^2 + \rho(-6v \pm \Delta t)\partial_x u_x + \rho v \nabla \cdot \mathbf{u} \\ 3\rho u_y^2 + \rho(-6v \pm \Delta t)\partial_y u_y + \rho v \nabla \cdot \mathbf{u} \\ 3\rho u_z^2 + \rho(-6v \pm \Delta t)\partial_z u_z + \rho v \nabla \cdot \mathbf{u} \\ 3\rho u_y u_z + \rho(-3v \pm 0.5\Delta t)\left(\partial_y u_z + \partial_z u_y\right) \\ 3\rho u_x u_z + \rho(-3v \pm 0.5\Delta t)\left(\partial_x u_z + \partial_z u_x\right) \\ 3\rho u_x u_y + \rho(-3v \pm 0.5\Delta t)\left(\partial_x u_y + \partial_y u_x\right) \\ -0.8, k = 11, \ldots, 16 \\ -0.95, k = 17, 18, 19 \end{bmatrix} . \tag{3.15}$$

Here, the parameters $\alpha_{11-16}^{\pm}$ and $\alpha_{17-19}^{\pm}$ correspond to third- and fourth-order terms, respectively. These terms arise as non-physical artifacts due to the discretization of the LBE. To enhance the numerical stability of the algorithm, these parameters are set to -0.8 and -0.95, respectively. This combination effectively dampens the contribution of the non-physical terms, leading to more stable simulations. These parameters do not have any inherent physical significance and can therefore alternatively be equated to zero, without altering the physical outcome of the simulation. This filtering of the higher-order terms is where the filter matrix collision operator derives its name from. The enhanced numerical stability obtained from this filtering is the main advantage of the filter matrix collision operator over other LBM collision operators [121].

3.1.3. FM-LBM for Thermal Flows

Until now, we have discussed the application of the Lattice Boltzmann framework primarily for simulating velocity fields. However, the FM-LBM algorithm can also be applied to advection-diffusion equations, enabling the simulation of the heat equation shown in Equation 2.4. This extension allows us to solve for both temperature and velocity fields using FM-LBM. By coupling these simulations, the algorithm can be used to model thermally driven fluids within the Boussinesq framework.

At the basis of Thermal FM-LBM lies the introduction of a new set of distribution functions $\{g_i\}$, known as the temperature distribution functions. The evolution of these distributions is again described by the LBE through

$$g_i(\mathbf{x} + \mathbf{c}_i \Delta t, t + \Delta t) - g_i(\mathbf{x},t) = \Delta t \, \Omega_i(g) . \tag{3.16}$$

Similar to velocity simulations, the collision operator relaxes the thermal distribution functions towards an equilibrium distribution, given by

$$g_i^{eq} = w_i T \left(1 + \frac{\mathbf{u} \cdot \mathbf{c}_i}{c_s^2} + \frac{(\mathbf{u} \cdot \mathbf{c}_i)^2}{2c_s^4} - \frac{\mathbf{u} \cdot \mathbf{u}}{2c_s^2} \right) . \tag{3.17}$$

Finally, the macroscopic temperature, T, can be obtained at any lattice point by summing over the temperature distribution functions:

$$T = \sum_i g_i \, . \tag{3.18}$$

By employing a staggered grid for the spatial and temporal domains, the filter matrix collision operator can also be constructed for the thermal distribution functions. This results in a similar formulation of the filter matrix but introduces a new solution vector $\beta_k^{\pm}(\mathbf{x}, t)$ specific to the thermal distribution functions. Analogous to Equation 3.12, this solution vector is related to the thermal distributions through

$$g_i \left(\mathbf{x} \pm \frac{\mathbf{c}_i \Delta t}{2}, t \pm \frac{\Delta t}{2} \right) = \sum_k w_i E_{ik} \beta_k^{\pm}(\mathbf{x}, t) \, . \tag{3.19}$$

This relationship can also be inverted using the same method as in Equation 3.13. In the D3Q19 framework, the filter matrix is identical to the one given in Equation 3.14, while the solution vector $\beta_k^{\pm}$ is defined as

$$\beta_k^{\pm} = \begin{bmatrix} T \\ T u_x + \frac{-8\alpha \pm \Delta t}{8} \partial_x T \\ T u_y + \frac{-8\alpha \pm \Delta t}{8} \partial_y T \\ T u_z + \frac{-8\alpha \pm \Delta t}{8} \partial_z T \\ 0, k = 5, \ldots, 19 \end{bmatrix} . \tag{3.20}$$

Here, α is the thermal diffusivity, which is also found in the heat equation formulated in Equation 2.4. For thermal FM-LBM, the simulation process mirrors that of velocity FM-LBM, where the simulation consists of an alternation between a streaming step and a collision step. During the collision step, the solution vector $\beta_k^{-}(\mathbf{x}, t)$ is calculated from the pre-collision thermal distributions. Subsequently, the solution vector is transformed to $\beta_k^{+}(\mathbf{x}, t)$, which is then used to compute the post-collision thermal distributions.

The thermal and velocity fields are coupled during the collision step of the simulation. First, the thermal field is used to calculate the buoyancy force, as described in Equation 2.6. This force term is then applied in the collision step of the velocity field, specifically when transforming the solution vector $\alpha_k^{-}(\mathbf{x}, t) \to \alpha_k^{+}(\mathbf{x}, t)$. Additionally, the velocity field is used in the thermal collision step during the transformation of the solution vector $\beta_k^{-}(\mathbf{x}, t) \to \beta_k^{+}(\mathbf{x}, t)$. This coupling of the two fields within the collision step enables the simulation of thermally driven flows.

3.1.4. FM-LBM for Precursor Transport

Similar to heat transport, precursor transport is also described by an advection-diffusion equation, as presented in Equation 2.16. Consequently, precursor flow can also be simulated using the FM-LBM. The implementation of precursor FM-LBM parallels that of thermal FM-LBM and thus also requires introducing new sets of distribution functions. In the case of precursor simulation, the distribution sets are denoted as $\{h_i^j\}$. Note that an additional index j is introduced which represents the simulation of multiple precursor families. Thus, for each precursor species j, a new set of distribution functions is introduced.

Mathematically, the formulations of precursor FM-LBM remain consistent with those in Equation 3.16 to Equation 3.20 for thermal FM-LBM. However, in this case, the macroscopic quantity is the precursor density. This means a summation over the distribution functions leads to

$$C_j = \sum_i h_i^j \, . \tag{3.21}$$

Additionally, the solution vector for precursor family j is now denoted by $\gamma_k^{j\,\pm}$. This vector is analogous to the solution vector for thermal LBM in Equation 3.20, however, with the thermal diffusivity α replaced by the molecular diffusivity D_j for precursor family j in the fluid.

Finally, precursor FM-LBM follows the same standard procedure of alternating between streaming and collision steps. And because the collision steps of precursor FM-LBM and thermal FM-LBM are virtually identical, the precursor field is coupled to the velocity field when the solution vectors are transformed from $\gamma_k^{j\,-}(\mathbf{x}, t) \to \gamma_k^{j\,+}(\mathbf{x}, t)$. This coupling ensures the convective transport of precursor densities within the velocity field.

3.1.5. Boundary Conditions

In the LBM framework, boundary conditions are applied directly to the mesoscopic distribution functions f_i by manipulating the streaming schemes of lattice nodes situated closest to the boundary. The literature offers a wide range of boundary schemes [69, 124], which can be broadly categorized into two families: link-wise boundary conditions and wet-node boundary conditions, as illustrated in Figure 3.3.

In the case of wet-node boundary conditions, the computational boundary coincides with the physical domain boundary. This means lattice points represent the corners of computational cells, with boundary lattice nodes located directly on the physical boundary. This research, however, focuses exclusively on link-wise boundary conditions. In this case, the lattice points are shifted by half a grid spacing, placing them at the centers of computational cells. Consequently, a small gap of half a grid spacing arises between the computational and physical boundaries, placing the computational domain inside the physical domain. To implement the boundary streaming scheme on these boundary nodes, it is convenient to introduce so-called ghost nodes. These nodes lie just outside the physical domain, with the physical boundary located between the boundary nodes and the ghost nodes. As these nodes are outside the physical domain, they are also referred to as solid nodes, implying a solid containing the fluid.

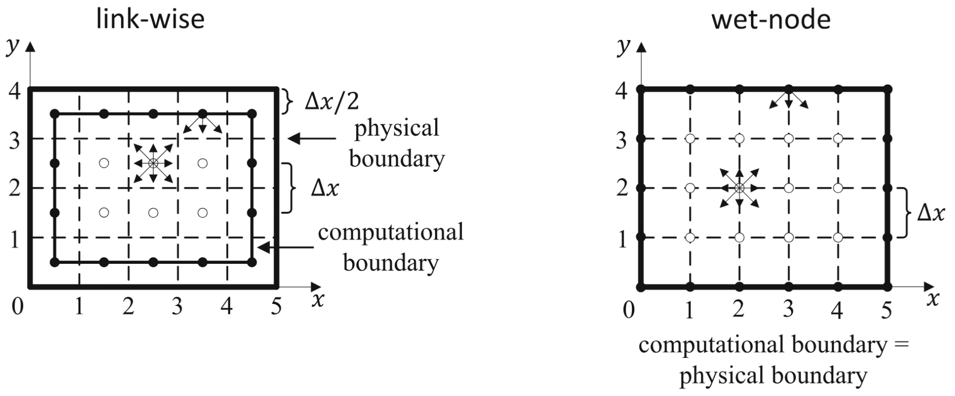

Figure 3.3: Comparison of link-wise and wet-node boundary conditions in LBM simulations. With link-wise boundary conditions, the computational boundary is shifted by half a grid spacing from the physical boundary, positioning the lattice nodes at the centers of the computational cells. In contrast, wet-node boundary conditions align the computational boundary with the physical boundary, placing the lattice nodes at the corners of the computational cells. This figure is obtained from [55].

In the remainder of this section, various types of boundary conditions will be discussed, along with their implications for the manipulation of the streaming schemes of the boundary nodes. These methods are summarized from [55]. The examples below will be illustrated using the D2Q9 grid given in Figure 3.1a, as this allows for more convenient visualizations. However, the corresponding formulas are generally applicable to all scheme variants, including those in 3D.

Periodic Boundary Conditions

Periodic boundary conditions are straightforward to implement by equating the incoming post-streaming distribution functions at one end of the domain with the outgoing pre-streaming distribution functions at the opposite end. This relationship is given by

$$f_i(\mathbf{x}_\mathrm{b}, t + \Delta t) = f_i^*(\mathbf{x}_\mathrm{b} + \mathbf{L}, t)\,, \tag{3.22}$$

where f_i^* and f_i represent the pre- and post-streaming distribution functions, respectively. Additionally, $\mathbf{x}_\mathrm{b}$ denotes a boundary lattice point. Figure 3.4 illustrates the implementation of periodic boundary conditions by introducing ghost nodes $\mathbf{x}_0$ and $\mathbf{x}_{N+1}$. In this setup, the pre-streaming distributions on ghost node $\mathbf{x}_0$ is initialized according to $f_i^*(\mathbf{x}_0, t) = f_i^*(\mathbf{x}_N, t)$ for the velocity directions $i = 1, 5, 8$. As a result, in terms of the post-streaming distribution functions f_i, we obtain $f_i(\mathbf{x}_1, t + \Delta t) = f_i^*(\mathbf{x}_N, t)$ for these directions. Conversely, for the distributions moving in the opposite direction, $i = 3, 6, 7$, we initialize the ghost nodes through $f_i^*(\mathbf{x}_{N+1}, t) = f_i^*(\mathbf{x}_1, t)$ and thus obtain for the post-streaming distributions the relation $f_i(\mathbf{x}_N, t + \Delta t) = f_i^*(\mathbf{x}_1, t)$.

Dirichlet Boundary Conditions

Implementing Dirichlet boundary conditions presents a more advanced challenge compared to periodic boundary conditions. First of all, due to the differential physical behavior between the NSE underlying the velocity fields, and the advection-convection equations underlying the temperature and precursor fields, it is essential

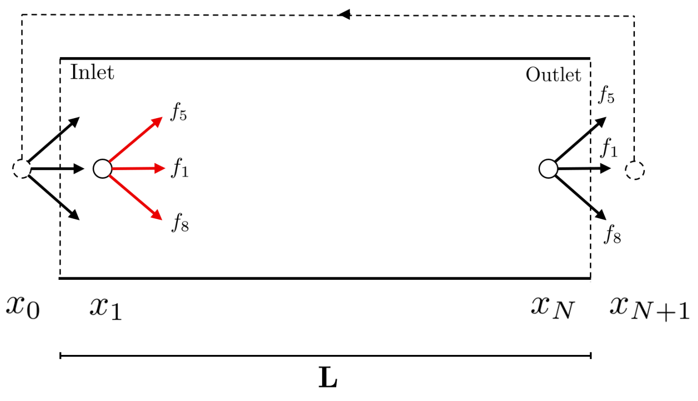

Figure 3.4: Visualization of periodic boundary conditions in LBM and its implementation using ghost nodes. The black arrows indicate pre-streaming distributions while the red arrows denote post-streaming distributions.

to differentiate between the streaming schemes for the distribution functions corresponding to the velocity field, f_i, and those for the temperature and precursor fields, g_i and h_i, respectively. This research uses the halfway bounce-back (HBB) method for the implementation of these Dirichlet boundary conditions. First, we focus on the implementation HBB approach for the velocity field, specifically for a no-slip boundary condition where the macroscopic fluid velocity at the physical wall, $\mathbf{u}_{\text{wall}}$, is set to zero. This condition is achieved by streaming the boundary nodes according to

$$f_i(\mathbf{x}_{\text{b}}, t + \Delta t) = f^*_{\bar{i}}(\mathbf{x}_{\text{b}}, t)\,. \tag{3.23}$$

Here, f^*_i and f_i represent the pre- and post-streaming distribution functions, respectively. Additionally, $\bar{i}$ denotes the opposite direction of i, implying $\mathbf{c}_{\bar{i}} = -\mathbf{c}_i$. This means that distribution functions moving from the boundary nodes to the solid nodes are reversed at the boundary, as illustrated in Figure 3.5. In terms of implementation using the ghost nodes, we can initialize the pre-streaming distribution functions on the ghost nodes with the reversed distribution functions through the relation $f^*_i(\mathbf{x}_{\text{b}} - \mathbf{c}_i\Delta t, t) = f^*_{\bar{i}}(\mathbf{x}_{\text{b}}, t)$, and then apply normal streaming.

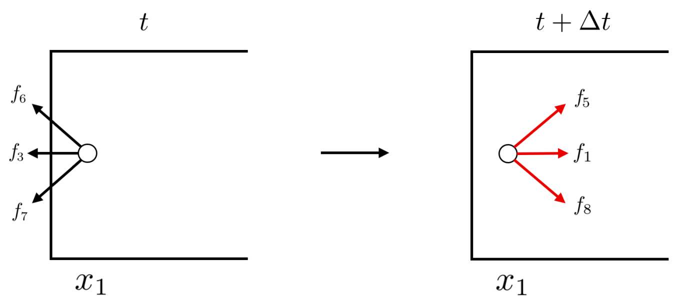

Figure 3.5: Visualization of Dirichlet boundary conditions in LBM and its implementation by reverting the distribution functions moving towards the solid nodes. The black arrows indicate pre-streaming distributions while the red arrows denote post-streaming distributions

For a moving wall, an extra contribution must be added after reversing, due to the momentum of the moving wall, transforming Equation 3.23 to

$$f_i(\mathbf{x}_{\text{b}}, t + \Delta t) = f^*_{\bar{i}}(\mathbf{x}_{\text{b}}, t) + 2w_i\rho\frac{\mathbf{c}_i \cdot \mathbf{u}_{\text{wall}}}{c_s^2}\,. \tag{3.24}$$

For the distribution functions describing an advection-diffusion equation, the anti-bounce-back (ABB) approach is used [55]. This method is similar to the half-way bounce-back (HBB) approach, where particle densities streaming into the wall are reflected back into the domain. The ABB method, however, introduces an additional negative sign in front of the pre-streaming distribution function. Moreover, when the macroscopic quantity corresponding to the distribution functions is specified to a non-zero value Q_{wall} at the physical boundary an additional term must be incorporated to enforce this requirement. This yields the complete ABB method, represented by

$$g_i(\mathbf{x}_{\text{b}}, t + \Delta t) = -g^*_{\bar{i}}(\mathbf{x}_{\text{b}}, t) + 2w_iQ_{\text{wall}}\,. \tag{3.25}$$

Note that here the distribution functions, g_i, for the temperature field is used, however, the formula will be identical for the distributions, h_i, of the precursor field.

Von Neumann Boundary Conditions

The Von Neumann boundary condition specifies that the normal gradient of a macroscopic quantity at a boundary is set to a constant value. In this study, we focus exclusively on the case where this gradient is zero. This condition is implemented by transforming the streaming schemes using mirror symmetry at the boundary, as described by the relation

$$f_i(\mathbf{x}_\mathrm{b} + \mathbf{c}_{i,t}\Delta t, t + \Delta t) = f_{\hat{i}}^*(\mathbf{x}_\mathrm{b}, t) \,. \tag{3.26}$$

Here, $\hat{i}$ denotes the mirrored direction of i relative to the boundary, which means that the normal velocity component is reversed, such that $\mathbf{c}_{\hat{i},n} = -\mathbf{c}_{i,n}$. Furthermore, $\mathbf{c}_{i,t}$ denotes the tangential velocity component relative to the boundary. Figure 3.6 illustrates how this boundary condition can be implemented using ghost nodes. First, the pre-streaming distribution functions are initialized on the ghost nodes using the symmetry conditions. Subsequently, normal streaming is applied. Note that no differentiation in the implementation of the Von Neumann boundary condition is needed for the different distribution functions describing the velocity, temperature, and precursor fields. Furthermore, for the velocity field, the Neumann boundary condition is also commonly referred to as the free-slip boundary condition.

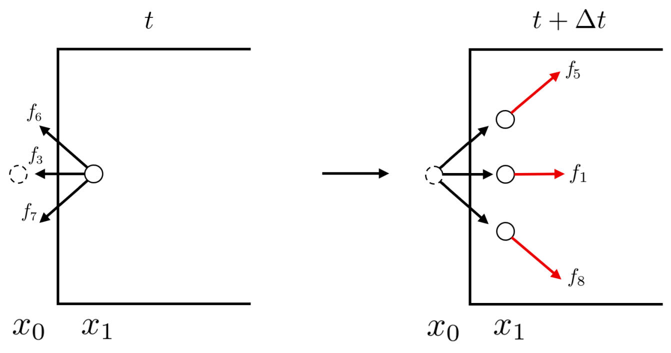

Figure 3.6: Visualization of Neumann boundary conditions in LBM and its implementation using ghost nodes. The black arrows indicate pre-streaming distributions while the red arrows denote post-streaming distributions

3.1.6. Unit Conversion

In LBM simulations, the simulation parameters are expressed in lattice units. For instance, the spatial dimension is determined by the number of grid points. When the problem domain is represented by a 100 × 100 grid, the spatial dimension is said to be 100 ls along each axis, where ”ls” denotes the lattice unit of distance. Similarly, the lattice unit of time, ”lt”, is measured by the time iterations of the LBM algorithm. For example, after five iterations, 5 lt has passed. Like physical units, Lattice units can also be combined to express the units of other variables. For instance, the lattice unit of velocity is given by ls/lt. In this research, whenever a variable Q is expressed in lattice units, it is provided with an asterisk superscript (e.g. Q^*). Conversely, whenever we want to specifically emphasize a variable is in physical units, it is represented with the subscript ”ph” (e.g. Q_{ph}).

When using simulation parameters expressed in lattice units, a few important considerations must be addressed. First, the parameters must be chosen to ensure that the algorithm simulates the same fluid problem as defined by the physical parameters. This is achieved by selecting the simulation parameters such that they yield the same values for the dimensionless numbers governing the fluid and system properties. For example, if a fluid problem is fully characterized by the Reynolds number, the simulation parameters should be selected to satisfy the relation

$$Re = \frac{U_{ph} L_{ph}}{\nu_{ph}} = \frac{U^* L^*}{\nu^*} \,. \tag{3.27}$$

Secondly, when switching between lattice units and physical units, conversion factors must be used. These factors are calculated as the ratio of the variable in physical units to its corresponding value in lattice units, expressed as

$$C_Q = \frac{Q_{ph}}{Q^*} \,. \tag{3.28}$$

For example, the conversion factor for distance is given by

$$C_x = \frac{L_{ph}}{L^*} = \frac{L_{ph}}{N} \,, \tag{3.29}$$

where N denotes the number of grid points along a given axis. Additionally, as outlined by [55], conversion factors can be combined to derive conversion factors for other variables. For instance, combining the distance conversion factor from Equation 3.29 with the viscosity conversion factor, defined as

$$C_\nu = \frac{\nu_{ph}}{\nu^*}, \tag{3.30}$$

yields the conversion factor for time as

$$C_t = \frac{C_x^2}{C_\nu}. \tag{3.31}$$

3.2. GPU Accelerated Thermal Fluid Simulation

Due to the independent calculations of the streaming and collision steps at each lattice point, the FM-LBM algorithm provides a perfect candidate for parallel computing of thermal fluids. Since the streaming and collision operations are computationally small enough, a GPU is used to execute this code in parallel. GPU-accelerated FM-LBM allows the collision and streaming steps within the FM-LBM code to be computed simultaneously for all lattice points. Theoretically, this means the computation time for the simulation does not necessarily increase with larger problem spaces, as all calculations on the lattice points are performed in parallel. In practice, however, computation time does increase with the number of lattice points, primarily due to the higher memory overhead involved in reading and writing larger arrays. Larger problem spaces require more storage to accommodate additional lattice points, which introduces memory access costs. Nevertheless, the increase in computation time on the GPU is significantly lower than that on a CPU, where lattice points are processed sequentially [44].

3.2.1. Julia-CUDA

In this research, the programming language Julia-CUDA is used to implement the FM-LBM code on a GPU. Julia is a flexible dynamic language used for scientific computing, much like Python and MATLAB. However, Julia boasts computational performance comparable to statically typed languages like C and Java, thanks to its just-in-time compiler and high-performing type inference system [56]. Within Julia, the CUDA package can be downloaded to enable CUDA programming through a Julia interface. The CUDA programming language, extensively described in the theory section of this report, is used for parallel computing on a GPU.

There are several advantages to using Julia-CUDA for GPU computations over other programming languages. Firstly, scientific programming languages are often favored over more complex languages like C and Java, which require an intricate understanding of low-level programming concepts. Julia stands out as an ideal choice because it offers computational performance on par with these advanced languages while maintaining ease of implementation due to its high-level syntax. Additionally, Julia also offers a lot of flexibility and functionality for handling GPU hardware. For example, Julia allows the user to assign variables to different levels of memory within NVIDIA GPUs' memory hierarchy. This increased flexibility results in better performance and greater control over the computational tasks of a GPU. This combination of high performance, ease of use, and GPU functionalities, makes Julia a great choice for scientific and parallel computing applications.

3.2.2. LBM Kernel Functions

GPU calculations within the CUDA framework are executed through kernel functions, which are small pieces of code designed to run on each GPU core. In this research, six distinct kernel functions are introduced: an initialization kernel, a propagation kernel, a boundary condition kernel, and three collision kernels—one for each physical variable being solved (velocity, temperature, and precursor density). Along with these different kernel functions, various levels of parallelization are introduced. For example, the propagation kernel is parallelized across both the spatial and velocity dimensions. This is because the streaming step calculations along different velocity directions are symmetrical, involving only the reading of distribution functions from neighboring nodes. Additionally, the streaming steps for the velocity, temperature, and precursor distributions are combined into a single kernel function due to their overlapping computations.

In contrast, the collision kernel is parallelized only over the spatial dimensions. This limitation arises because the matrix multiplications within the collision steps utilize distribution function information across all velocity directions, ruling out further parallelization. Moreover, the calculations for the collision steps vary significantly between different types of distribution functions, necessitating separate kernel functions for the velocity, temperature, and precursor fields. Although the collision steps for the temperature and precursor distributions are potentially similar enough to be combined, they are still split into two distinct kernels. This is because, the precursor collision kernel is parallelized over both spatial dimensions and precursor families, as these calculations are identical for different precursor families. This additional parallelization layer ultimately enhances overall performance, however, it requires separate implementations for the temperature and precursor collision steps.

Lastly, the boundary condition kernel is implemented similarly to the propagation kernel, with parallelization across both spatial and velocity dimensions. It applies boundary conditions for the velocity, temperature, and precursor distributions using overlapping code. This kernel function is the only one that performs calculations on the ghost nodes, initializing the particle distributions that enter the fluid domain at these nodes before the propagation kernel is executed. This ensures that the boundary nodes read the correct distribution function values from the ghost nodes during the propagation step.

Figure 3.7 presents an overview of the complete GPU-accelerated FM-LBM code, detailing the sequence of kernel function executions within each iteration. It also illustrates the standard parallel programming procedure using CUDA, as outlined in the theory section. This process involves initializing the problem space on the CPU, allocating GPU memory, transferring variables from CPU to GPU memory, executing kernel functions on the GPU for simulations, and then transferring the results back to CPU memory.

Note that, in this figure, the number of time iterations for the FM-LBM algorithm is set to a fixed value, N_t. When the iteration number t surpasses this value, the algorithm terminates and transfers the results back to the CPU. However, for simulations of stable flows progressing toward a steady-state solution, the number of iterations can be adjusted dynamically based on a convergence criterion, which will be discussed in the next chapter.

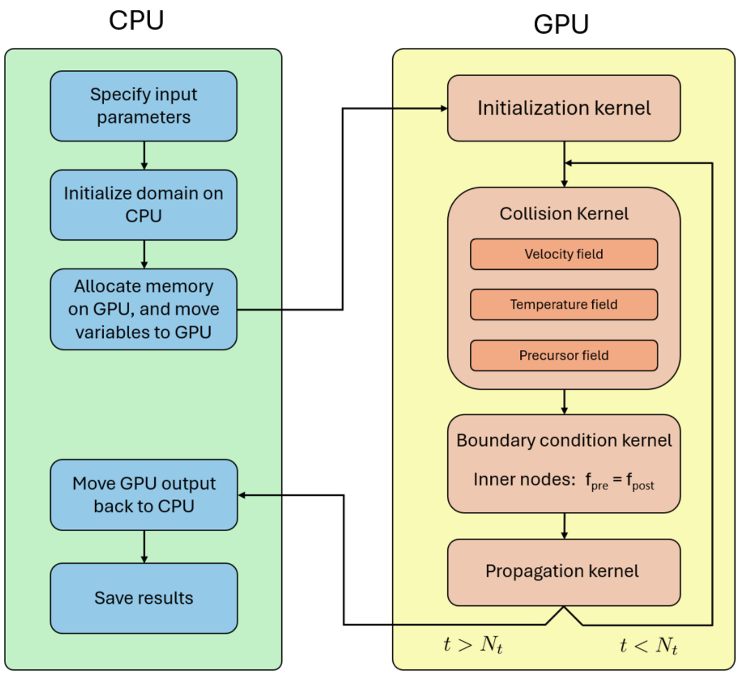

Figure 3.7: Overview of the GPU-accelerated FM-LBM algorithm. The figure illustrates the communication between the CPU and GPU, including the initialization of the problem space on the CPU, GPU memory allocation, transfer of variables from CPU to GPU, execution of kernel functions on the GPU, and the return of results to CPU memory. It also depicts the ordering of kernel functions within the main loop and the stopping criterion, where the loop terminates after N_t time iterations.

3.2.3. Race Conditions

In parallel algorithm implementations, it is crucial to address race conditions. These occur when multiple threads simultaneously access and modify the same memory address, potentially leading to corrupted simulation results without raising errors. This issue is particularly significant during the streaming step, where distribution functions are transferred to neighboring nodes. Since each thread targets a neighboring node and the warp scheduler's execution order is unpredictable, threads may read from and write to the same memory address in the array holding distribution function values. To prevent such race conditions in our code, we introduce two copies of the array storing the distribution functions. During the streaming step, the first array, `f_pre`, is read from, while the second array, `f_post`, is written to. Here, `f_pre` and `f_post` represent the pre- and post-streaming distribution functions, respectively. After the collision step, these arrays must be synchronized to ensure that `f_pre` at iteration `t+1` matches `f_post` at iteration `t`. This synchronization is handled by the boundary condition kernel, which now serves a dual purpose: for ghost nodes, it initializes distribution functions directed into the fluid domain, and for inner nodes, it updates the `f_pre` array to reflect the `f_post` array. The latter is also depicted in Figure 3.7 in the boundary condition kernel.

Another source of race conditions in LBM computations arises from unsynchronized GPU nodes between kernel launches. If the CPU initiates the next kernel launch before the previous one has finished, two kernels might execute simultaneously on the same lattice points, causing race conditions as they access the same variables. To avoid this, CUDA's built-in synchronization function, `CUDA.synchronize()`, should be used after each kernel launch to ensure the previous kernel has been completed before starting the next one.

3.2.4. Performance Gains

To fully leverage the GPU resources, several techniques have been introduced to the FM-LBM code to enhance its computational efficiency. Some of these techniques have already been discussed in the previous sections, such as the definition of the number of threads per block being a multiple of 32, as discussed in the theory section. But also introducing different levels of parallelization between kernel launches, as certain simulation steps allow for higher-level parallelization. As Julia-CUDA is flexible enough to introduce different numbers of blocks for different kernel launches, this differentiation in the level of parallelization allows for the faster execution of some of the kernels. Lastly, to reduce computational effort, the boundary condition kernel only considers particle densities moving into the fluid domain on the ghost nodes, disregarding other directions on these nodes that are irrelevant to the simulation.

In addition to these improvements, further optimizations have been made. As explained by [6], it is more efficient to store information in 1D arrays rather than multidimensional arrays, as 1D arrays incur less computational overhead on the GPU. Consequently, the 4D arrays containing the distribution functions and the 3D arrays containing the macroscopic variables are converted into long 1D arrays. The indexation through these arrays is facilitated by so-called index functions, which are defined as

$$\text{3D to 1D conversion:} \qquad idx = x + yN_x + zN_xN_y\,, \tag{3.32}$$

$$\text{4D to 1D conversion:} \qquad idx = x + yN_x + zN_xN_y + qN_xN_yN_z\,. \tag{3.33}$$

Here, (x, y, z) denote the spatial coordinates of the lattice points in the grid, q labels the different velocity directions of the scheme used, and N_x, N_y, N_z denote the number of lattice points in the x, y, z direction, respectively. Note that, in the index function for the 4D conversion, the velocity index is placed after the grid coordinates. This arrangement ensures that in a long 1D array, the distribution functions corresponding to the same velocity direction are stored next to each other, as illustrated in Figure 3.8. This configuration is implemented on purpose, as the ordering of the different distribution values in the 1D array affects the simulation speed of the GPU. This specific ordering improves simulation speed due to a process called memory coalescing within a GPU, where consecutive threads access consecutive memory addresses [24, 99]. By storing the distribution functions in this manner, the collision kernel benefits from increased speed during matrix multiplications, as consecutive threads handling adjacent lattice points in the domain simultaneously request information on the same velocity direction during the matrix multiplication.

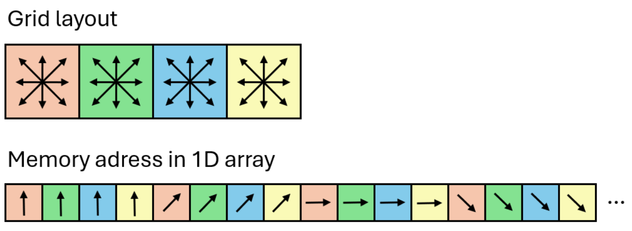

Figure 3.8: Illustration of the storage of particle distribution functions in 1D arrays. This specific ordering of distribution functions, where distributions corresponding to the same velocity direction are stored next to each other, optimizes GPU performance through memory coalescing.

Shared Memory

As already mentioned, the NVIDIA GPUs have a memory hierarchy, where memory closer to the individual cores is faster at the cost of lower capacity and more restrictive access. Julia-CUDA allows for low-level management of this memory, enabling the assignment of variables to either global memory or shared memory (with register memory reserved for constants defined within kernel functions). This makes the option for the usage of shared memory attractive, as it increases simulation speed. However, because lattice nodes are interconnected through the streaming step and shared memory is only accessible by threads within the same block, finding effective use cases for shared memory can be challenging.

In this research, a notable use case for shared memory is identified in the collision kernel considering the solution vector. Storing the complete solution vector for every lattice point is unnecessary. Instead, a solution vector can be generated for each lattice point within a block through shared memory, which persists for the duration of the kernel. This solution vector is employed in the filter matrix process of the collision step, which involves three stages: constructing the solution vector from the pre-collision distributions via matrix multiplication, transforming the solution vector, $\alpha_k^- \to \alpha_k^+$, and finally calculating the post-collision distributions from the transformed solution vector through another matrix multiplication. At the end of the kernel execution, only the relevant information, such as the velocity field or temperature field, is stored in global memory, while the remaining information of the solution vector is discarded. This approach has two key advantages. First, it minimizes global memory storage by only storing relevant solution vector information, optimizing information transfer as smaller arrays reduce latency in reading and writing. Second, it speeds up calculations within the kernel function, as the GPU accesses shared memory rather than global memory whenever information needs to be read from or written to the solution vector.

Moreover, the use of shared memory for the solution vector presents another opportunity for enhanced performance of the collision kernel. This is a consequence of the fact that, in kernel programming, matrix multiplications must be written as two nested for-loops due to the absence of a standard vector algebra package within the CUDA framework. In the collision kernel's matrix multiplications, three types of arrays are involved: the array containing the distribution functions for the entire problem space stored in global memory, the $q \times q$ filter matrix array stored in global memory, and the solution vector array stored in shared memory for lattice points within the same block. The latter is the quickest to access, given that it is stored in shared memory. The nested loops can be organized in two ways: iterating through the filter matrix rows in the outer loop and columns in the inner loop, or vice versa. Each method affects the constancy of the indices of the arrays being read from or written to, which are, the distribution functions array and the solution vector array. Specifically, for a $q \times q$ filter matrix, one index will iterate q^2 times while the other will iterate only q times, meaning one index remains constant more frequently than the other. This is illustrated in Figure 3.9.

Given that accessing the distribution function array in global memory is more costly than accessing the solution vector array in shared memory, the nested loops should be organized to minimize iterations through the distribution function array. For the first matrix multiplication in the collision kernel, where the solution vector is derived from the pre-collision distribution functions, $\alpha_k^- = \sum_i E_{ki} f_i$, the configuration on the left in Figure 3.9 is optimal. In this configuration, the *outer* loop iterates over the *columns* of the filter matrix, while the *inner* loop iterates over its *rows*. This arrangement results in only q iterations through the distribution function array (once per element) while iterating q^2 times through the solution vector array (q times per element). Given that information exchange with the solution vector array in shared memory is quicker, this configuration optimizes the matrix multiplication.

Conversely, for the second matrix multiplication in the collision kernel, where the post-collision distribution functions are derived from the transformed solution vector, $f_i^* = \sum_k w_i E_{ik} \alpha_k^+$, the configuration on the right in Figure 3.9 should be used. In this case, the *outer* loop iterates over the *rows* of the filter matrix, while the *inner* loop iterates over its *columns*. This again ensures that the distribution function array is accessed only q times (once per element), while the solution vector array is accessed q^2 times (q times per element). Again, this configuration leverages the faster access to the solution vector array in shared memory, optimizing the matrix multiplication.

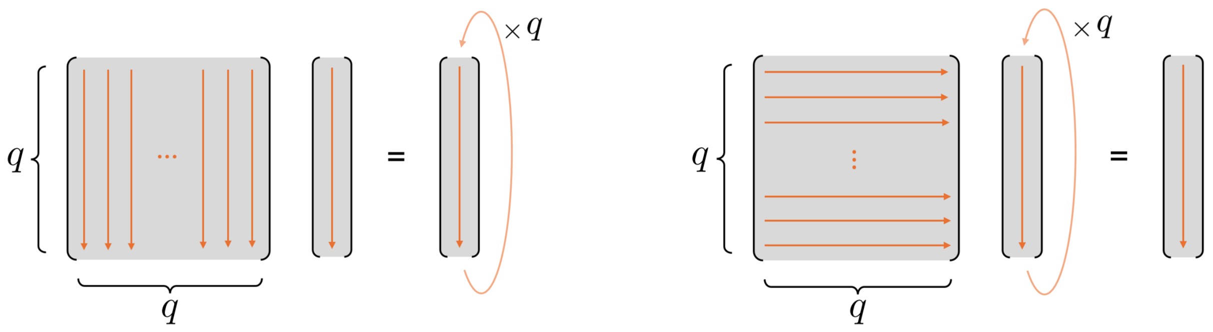

Figure 3.9: Different configurations of nested for loops in matrix multiplication. The order of these loops affects which of the two arrays is iterated over most frequently. When one array has higher access latency, it is more computationally advantageous to use a configuration that minimizes iterations over that array.

As a concluding remark, it is important to highlight that the performance gains achieved through the use of different levels of parallelization across various kernels, the application of shared memory for the solution vector in the collision kernel, and the optimal configuration of matrix multiplications to minimize iterations through the large distribution array stored in global memory are, as far as we are aware, unique to this research.

3.3. Neutronics Simulation

In addition to simulating thermal fluid dynamics, the neutronics of the reactor core must also be modeled to capture the complete multiphysics behavior of an MSFR reactor core. While the FM-LBM algorithm can simulate neutronics using the diffusion approximation [111, 112], this research takes a different approach by directly solving the full NTE, without relying on the diffusion approximation. This is achieved through the use of an in-house code, Phantom-S_N, which solves the S_N-equations via direct numerical simulation using a discontinuous Galerkin finite element method (DG-FEM) approach [54]. Unlike the FM-LBM code, which is implemented in Julia-CUDA and optimized for GPU execution, Phantom-S_N is written in Fortran-90 [28] and designed to run on a CPU.

3.3.1. S_N-Equations

The S_N-equations can be derived from the full formulation of the NTE, including delayed neutrons as given in Equations 2.15, through a series of approximations that discretizes the equation. The following section outlines the step-by-step derivation, adapted from [61].

First, the multigroup approximation is applied, where the energy variable is divided into G energy groups. The NTE is then integrated over each energy group, resulting in G coupled equations, denoted as

$$\begin{aligned}\frac{1}{\mathrm{v}_g}\frac{\partial\varphi_g}{\partial t}+\mathbf{\Omega}\cdot\nabla\varphi_g+\Sigma_{t,g}\varphi_g=&\sum_{g'=1}^{G}\int_{4\pi}\Sigma_{s,g'\to g}\left(\mathbf{\Omega}'\cdot\mathbf{\Omega}\right)\varphi_{g'}\mathrm{d}\mathbf{\Omega}'\\&+\frac{(1-\beta)\chi_g^p}{4\pi}\sum_{g'=1}^{G}\bar{\nu}_{g'}\Sigma_{f,g'}\phi_{g'}+\frac{\chi_g^d}{4\pi}\sum_i\lambda_iC_i\,,\end{aligned}\tag{3.34}$$

where φ_g represents the angular neutron flux and ϕ_g denotes the scalar neutron flux for energy group g. This formula is known as the multi-group neutron transport equation (MG-NTE), which represents G coupled equations, one for each energy group g. The equations are coupled through the scattering term, where the cross-section $\Sigma_{s,g'\to g}$ describes the probability of neutrons scattering from energy group g' to energy group g. Furthermore, the nuclear cross-sections are replaced with energy-group average values, which are calculated using

$$\Sigma_{x,g}=\frac{\int_{E_g}^{E_{g-1}}\Sigma_x(E)\varphi(E)\mathrm{d}E}{\int_{E_g}^{E_{g-1}}\varphi(E)\mathrm{d}E}\,.\tag{3.35}$$

Here, E_g denotes the lower energy bound of the energy group g (Note, that the energy group index is reversed by convention, meaning $E_{g-1}>E_g$). Additionally, the discrete ordinates approximation is applied to the angular variable to further discretize the MG-NTE. This means, the angular variable is discretized, limiting the calculations to a discrete set of angles, denoted as $\{\mathbf{\Omega}_n\}$. This allows the integration over the angular directions to be replaced by a summation over the discrete ordinates, weighted by a factor w_n for each ordinate

$$\int_{4\pi}f\left(\mathbf{\Omega}\right)\mathrm{d}\mathbf{\Omega}\approx\sum_{n=1}^{N}w_nf\left(\mathbf{\Omega}_n\right)\,.\tag{3.36}$$

Finally, the scattering term is discretized using functional expansion, which reformulates the it as a series of spherical harmonics $Y_{l,m}(\Omega)$, defined as

$$Y_{l,m}(\Omega)=Y_{l,m}(\theta,\phi)=\sqrt{\frac{(2l+1)}{4\pi}\cdot\frac{(1-m)!}{(1+m)!}}P_l^m(\cos\theta)e^{im\phi}\,,\tag{3.37}$$

with θ the polar and ϕ the azimuthal angle corresponding to the solid angle Ω, and $P_l^m(x)$ the associated Legendre polynomials. As a result, by including the discretization over the angular variable and the scattering term, the MG-NTE can now be rewritten as

$$\frac{1}{\mathrm{v}_g}\frac{\partial \varphi_{n,g}}{\partial t} + \Omega_n \cdot \nabla \varphi_{n,g} + \Sigma_{t,g}\varphi_{n,g} = \sum_{g'=1}^{G}\sum_{l=1}^{L}\sum_{m=-l}^{l} \Sigma^l_{s,g'\to g} Y^*_{l,m}\left(\Omega_n\right)\tilde{\phi}^m_{l,g'} + \frac{(1-\beta)\chi^p_g}{4\pi}\sum_{g'=1}^{G}\bar{\nu}_{g'}\Sigma_{f,g'}\phi_{g'} + \frac{\chi^d_g}{4\pi}\sum_i \lambda_i C_i\,. \tag{3.38}$$

Here, $Y^*_{l,m}$ denotes the complex conjugate of $Y_{l,m}$, defined as $Y^*_{l,m} = (-1)^m Y_{l,-m}$, and $\tilde{\phi}^m_{l,g}$ denotes the expansion coefficients, which can be related back to the angular neutron flux through the relation

$$\tilde{\phi}^m_{l,g} = \sum_{n'=1}^{N} w_{n'} Y_{l,m}(\Omega_{n'})\varphi_{n',g}\,. \tag{3.39}$$

The expression formulated in Equation 3.38 are referred to as the S_N-equations. This expression represents a set of coupled equations corresponding to each combination of energy group g and angular ordinate n. These equations serve as the foundation of the Phantom-S_N algorithm.

3.3.2. Phantom-S_N

Phantom-S_N employs a DG-FEM algorithm to solve the S_N-equations. This approach involves discretizing the spatial variable using a mesh, where variables within each mesh element are approximated by a superposition of linearly independent basis functions, denoted as $h_i(\mathbf{r})$. The angular flux - for each energy group and angular ordinate - and the precursor density of species j within each mesh element can then be expressed as

$$\varphi_{n,g}(\mathbf{r}) = \sum_i \varphi_i^{n,g} h_i(\mathbf{r}) \qquad\qquad C_j(\mathbf{r}) = \sum_i C_i^j h_i(\mathbf{r})\,, \tag{3.40}$$

In this context, $\varphi_i^{n,g}$ and C_i^j represent the coefficients associated with the basis functions. These coefficients are uniquely defined for each mesh element and are the unknowns to be solved by the Phantom-S_N algorithm.

By substituting the expressions from Equation 3.40 into the S_N-equations and applying the Galerkin method, the S_N-equations are transformed into a set of coupled linear equations for each combination of energy group and angular ordinate. When these equations are solved for a specific mesh order, the corresponding matrices can be organized into a block lower triangular form. Stated more clearly, in this configuration, each block corresponds to the unknowns within a single mesh element. If the unknowns of different mesh elements are solved in a specific order — starting from the boundary elements and moving in the direction of the neutron flux specified by $\mathbf{\Omega}_n$ — the block lower triangular form naturally arises, due to boundary conditions being known for incoming angular fluxes. Adopting this approach allows for an efficient solution of the system of coupled linear equations, as the block lower triangular structure enables a step-by-step solution process, where each step depends only on the results from previous steps. This sequential solving method is referred to as a transport sweep. For a more detailed explanation of the Phantom-S_N algorithm see [54, 57, 79].

3.4. Multiphysics Simulation Tool

In the current setup, the FM-LBM algorithm, implemented in Julia-CUDA, is used to solve for the velocity, temperature, and precursor fields, while the Phantom-S_N code, written in Fortran-90, calculates the neutron flux shape. This coupling scheme is depicted in Figure 3.10, which is a modified version of Figure 2.1. The updated figure includes labels that clearly indicate which part of the problem is simulated by the FM-LBM algorithm and which is handled by the Phantom-S_N algorithm. From the figure it is clear that some coupling between the fields is already integrated into the FM-LBM algorithm. This includes the convection terms and the buoyancy force, both incorporated within the collision kernel of the FM-LBM. Additionally, the figure highlights the remaining processes that couple the FM-LBM and Phantom-S_N codes. These include the power source and temperature feedback mechanisms, which couple the neutron flux to the temperature field, and the precursor source (as a fission product) and delayed neutron source linking the neutron flux to the precursor field.

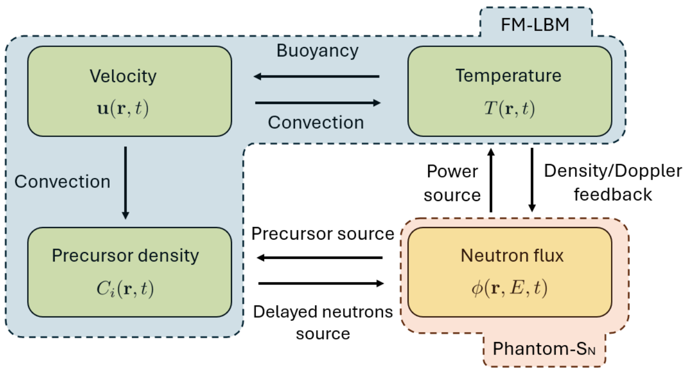
FM-LBM
Buoyancy
Velocity $\mathbf{u}(\mathbf{r},t)$
Temperature $T(\mathbf{r},t)$
Convection
Convection
Power source
Density/Doppler feedback
Precursor source
Precursor density $C_i(\mathbf{r},t)$
Neutron flux $\phi(\mathbf{r},E,t)$
Delayed neutrons source
Phantom-SN

Figure 3.10: Extended schematic overview of key physical quantities in MSFR reactor core simulations from Figure 2.1. This figure highlights which quantities are computed by the FM-LBM and Phantom-S_N codes, as well as the remaining processes that couple these codes.

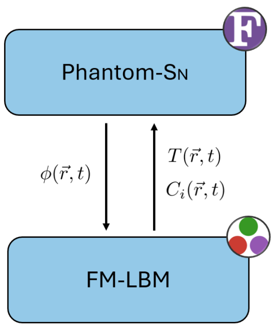

Figure 3.11: Abstract overview of the coupling between the FM-LBM and Phantom-S_N codes. This figure illustrates the information that is transferred between the two codes. The icons on the top right indicate the programming language the two codes are written in, being Fortran-90 and Julia.

3.4.1. Coupling Terms

Figure 3.11 provides a more abstract overview of the coupling between the FM-LBM and Phantom-S_N codes. It also outlines the variables exchanged between them, which are used to calculate the necessary coupling terms shown in Figure 3.10. Below is a summary of how these coupling terms are derived from the respective fields.

First, the precursor density generates a delayed neutron source in the NTE. In the code, this is managed by transferring the precursor density, calculated by the FM-LBM algorithm, to the Phantom-S_N code. Phantom-S_N then uses this density to compute the delayed neutron source as $\frac{\chi^d}{4\pi}\sum_i \lambda_i C_i$. Second, through fission interactions, the neutron flux leads to the production of new precursors as a fission product. In the code, this is handled by passing the scalar neutron flux to the FM-LBM code, after which the precursor source can be calculated as

$$Q_{C_i} = \beta_i \sum_g \nu_g \Sigma_{f,g} \phi_g \,. \tag{3.41}$$

Additionally, the scalar flux can also be used to calculate the power source for the heat equation through

$$q = E_{\textit{fiss}} \sum_g \Sigma_{f,g} \phi_g \,, \tag{3.42}$$

where $E_{\textit{fiss}}$ denotes the energy released from a single fission event. Lastly, the NTE couples back to the temperature field by transferring the temperature field from the FM-LBM algorithm to Phantom-S_N, which uses it to incorporate temperature feedback effects by adjusting the nuclear cross-sections. These feedback effects consist of two parts. First, the density feedback resulting from fuel salt expansion is modeled by scaling the cross-sections as

$$\Sigma_x(T) \;=\; \Sigma_x(T_{\textit{ref}})\frac{\rho(T)}{\rho(T_{\textit{ref}})} \;\approx\; \Sigma_x(T_{\textit{ref}})\left[1-\beta_{th}(T-T_{\textit{ref}})\right] . \tag{3.43}$$

Here, the latter formulation applies the same approximation as Equation 2.5, where density variations are modeled using the reference density and a correction term that is linearly dependent on temperature. This means, the parameter β_{th} is the same thermal expansion coefficient used to compute the buoyancy force. The second process involves Doppler effects, which broaden interaction peaks and subsequently modify nuclear cross-sections. As described in [7, 108], when Doppler effects are also taken into account next to density feedback, the combined feedback effect can be captured by adjusting the nuclear cross-sections according to

$$\Sigma_x(T) = \left[\Sigma_x(T_{\textit{ref}}) + \beta_r \ln\left(\frac{T}{T_{\textit{ref}}}\right)\right]\frac{\rho(T)}{\rho(T_{\textit{ref}})} \,. \tag{3.44}$$

Here, β_r is a feedback coefficient distinct from the thermal expansion coefficient β_{th}.

3.4.2. Steady-State Simulation

The Phantom-S_N algorithm performs steady-state calculations by solving a $k_{\textit{eff}}$ eigenvalue problem using the power method [68, 105]. This section outlines the formulation of the $k_{\textit{eff}}$ eigenvalue problem, the workings of the power method, and its implementation in the current situation where neutronics and precursor flow calculations a separated into two distinct algorithms.

By omitting the energy group and angle ordinate indexations and neglecting the time derivative of the angular flux (as we focus on the steady-state solution), we can reformulate Equations 3.38 and 2.16 concisely using operator notation as

$$A^t_{\varphi\varphi}\varphi = A^s_{\varphi\varphi}\varphi + A^f_{\varphi\varphi}\varphi + A_{\varphi C}C\,, \tag{3.45}$$

$$A^{\mathbf{u}}_{CC}C_i = A^f_{C\varphi}\varphi\,. \tag{3.46}$$

Here, Equation 3.45 represents the NTE, while Equation 3.46 describes the precursor advection-convection equation. The subscripts of the operators indicate interactions between the neutron flux and the precursor fields, and the superscripts specify the type of process the operator is associated with (t = transport, s = scattering, f = fission, $\mathbf{u}$ = advection/convection). As commonly described in the literature [30, 61], the k_{eff} eigenvalue problem is obtained by dividing the fission terms by the eigenvalue k_{eff}. Using matrix notation, Equations 3.45 and 3.46 can then be reformulated as

$$\begin{bmatrix} A^t_{\varphi\varphi} - A^s_{\varphi\varphi} & -A_{\varphi C} \\ 0 & A^{\mathbf{u}}_{CC} \end{bmatrix} \begin{bmatrix} \varphi \\ C_i \end{bmatrix} = \frac{1}{k_{\mathit{eff}}} \begin{bmatrix} A^f_{\varphi\varphi} & 0 \\ A^f_{C\varphi} & 0 \end{bmatrix} \begin{bmatrix} \varphi \\ C_i \end{bmatrix}. \tag{3.47}$$

which can be written down concisely as

$$\mathbf{M}\boldsymbol{\gamma} = \frac{1}{k_{\mathit{eff}}}\mathbf{F}\boldsymbol{\gamma}\,. \tag{3.48}$$

This formulation demonstrates that we are dealing with an eigenvalue problem, where k_{eff} is the eigenvalue and $\boldsymbol{\gamma} = [\varphi \;\; C_i]^T$ is the eigenvector. Phantom-S_N addresses this eigenvalue problem numerically using the power method. This algorithm starts by initializing the eigenvalue $k^{(0)}_{\mathit{eff}}$ and the solution vector on the RHS of Equation 3.48, $\boldsymbol{\gamma}^{(0)}$, with arbitrary values (or ideally values close to the actual solution). It then proceeds iteratively, calculating the next solution vector $\boldsymbol{\gamma}^{(l+1)}$ using the recurrence relation

$$\mathbf{M}\boldsymbol{\gamma}^{(l+1)} = \frac{1}{k^{(l)}_{\mathit{eff}}}\mathbf{F}\boldsymbol{\gamma}^{(l)}\,. \tag{3.49}$$

After each iteration, the eigenvalue is updated according to

$$k^{(l+1)}_{\mathit{eff}} = \frac{\left\|\boldsymbol{\gamma}^{(l+1)}\right\|}{\left\|\boldsymbol{\gamma}^{(l)}\right\|}\,, \tag{3.50}$$

where $\|.\|$ denotes the Euclidean L^2 norm. The power method ensures that this iterative procedure eventually converges to the eigenvector associated with the eigenvalue of largest magnitude [10]. In reactor physics, this means that the calculated eigenvalue k_{eff} corresponds to the fundamental mode of the reactor core, which is the dominant mode governing the overall behavior of the neutron population [30]. The question, however, is how to solve Equation 3.49 for a given $\boldsymbol{\gamma}^{(l)}$ and $k^{(l)}_{\mathit{eff}}$ using the combined FM-LBM and Phantom-S_N codes. This process consists of three steps:

1. First, calculate $C_i^{(l+1)}$ from $\boldsymbol{\gamma}^{(l)}$ using the lower expression from Equation 3.47. This is done by exporting $\varphi^{(l)}$ from Phantom-S_N to the FM-LBM code, calculating the precursor source term $A^f_{C\varphi}\varphi^{(l)}$, and subsequently running the FM-LBM algorithm from Figure 3.7 until convergence is achieved. This step basically calculates the steady-state solution of the advection-convection equation for the precursor field for the given precursor source $A^f_{C\varphi}\varphi^{(l)}$.
2. The resulting precursor field $C_i^{(l+1)}$ is transferred back to Phantom-S_N and used to calculate the dealyed neutron source ($A_{\varphi C}C_i^{(l+1)}$) in the upper expression of Equation 3.47. Phantom-S_N uses this information to update the fission source (RHS) to $\varphi^{(l)} + A_{\varphi C}C_i^{(l+1)}$.
3. Finally, Phantom-S_N solves the upper equation of Equation 3.47, considering only the diagonal term ($A^t_{\varphi\varphi} - A^s_{\varphi\varphi}$) in matrix $\mathbf{M}$ and using the updated fission source. This step basically corresponds to solving the steady-state NTE for a fixed delayed neutron source $A_{\varphi C}C_i^{(l+1)}$.

After these steps, the new $k^{(l+1)}_{\mathit{eff}}$ is calculated using Equation 3.50, and the power method proceeds to the next iteration by updating $\boldsymbol{\gamma}^{(l)} = \boldsymbol{\gamma}^{(l+1)}$. This step basically

It is important to emphasize that this discussion focuses solely on the information exchange between the two codes concerning precursor transport. However, as illustrated in Figure 3.10, the coupling between the NTE and the temperature field must also be considered. This can be easily implemented by calculating the power source alongside the precursor source when $\varphi^{(l)}$ is exported to the FM-LBM code. As the FM-LBM code runs and solves for the velocity, temperature, and precursor fields, the power source directly influences the temperature calculations. This, in turn, indirectly affects the precursor field calculations due to the interdependencies within the FM-LBM algorithm. On the other hand, when the FM-LBM results are returned to Phantom-S_N, the algorithm provides not only the updated precursor field $C_i^{(l+1)}$ but also the temperature field. This temperature field is then used to scale the nuclear cross-sections before Phantom-S_N executes the final step (step 3). Consequently, a two-way coupled system is established for both precursor and heat transport.

One final remark should be made concerning the current implementation of the power method in combination with thermal hydraulics. In the current approach, the temperature and velocity fields, are solved alongside the precursor field as part of the FM-LBM calculations. These results subsequently influence the precursor transport and nuclear cross-sections within the power method calculations. As a consequence, the matrices $\mathbf{M}$ and $\mathbf{F}$ of the eigenvalue problem defined in Equation 3.48 are effectively redefined at each iteration of the power method (implying that these matrices should technically also include a superscript denoting the iteration step). This means that during each iteration, only a single step of the power method is solved, after which the eigenvalue problem is redefined. This approach works because the power method does not require dependence between solution vectors across iterations. The method can begin with any initial solution vector, and while convergence is achieved faster when the initial vector is closer to the true solution, each iteration essentially "forgets" the previous solution vector. In the current implementation, the eigenvalue problem is redefined at each iteration, and the power method restarts. As the initial solution vector of the redefined problem becomes progressively closer to the true solution vector, eventually convergence is achieved within a single iteration of the power method. An alternative method would involve introducing a second outer loop. In this approach, the velocity and temperature fields are held constant during each iteration of the outer loop, allowing the power method to solve the eigenvalue problem to full convergence. Subsequently, new temperature and velocity fields are calculated in the next iteration of the outer loop, leading to a redefinition of the eigenvalue problem and restarting the power method. While this alternative ensures the matrices $\mathbf{M}$ and $\mathbf{F}$ remain constant during each power method iteration, solving the full power method repeatedly is significantly more computationally intensive. In our approach, where temperature and velocity are solved next to the precursor fields, we essentially take a shortcut to accelerate progress toward the steady-state solution.

3.4.3. Transient Simulation

In transient simulations, the time derivative must be reintroduced into the NTE. In Phantom-S_N, time discretization is implemented using a backward differing scheme. For instance, when employing a first-order backward differing scheme, Equation 3.45 is reformulated as

$$\frac{1}{\mathrm{v}_g}\frac{\varphi^{n+1}-\varphi^{n}}{\Delta t} + A^t_{\varphi\varphi}\varphi^{n+1} = A^s_{\varphi\varphi}\varphi^{n+1} + A^f_{\varphi\varphi}\varphi^{n+1} + A_{\varphi C}C^{n+1}\,. \tag{3.51}$$

Here, n denotes the time index, and Δt represents the time step. For a given delayed neutron source term, $A_{\varphi C}C^{n+1}$, Phantom-S_N solves the NTE using a similar approach to the steady-state case, but now includes an additional source term, $\varphi^n/(\Delta t \mathrm{v}_g)$, which depends on the neutron flux from the previous time step. For higher-order backward differing schemes, this additional source term becomes more complex.

The transient simulation algorithm can now be summarized as follows. First, the neutron flux, φ^0, is initialized. From this initial condition, the algorithm iterates over time steps, performing the following steps in each iteration:

1. Export the precursor source, $A^f_{C\varphi}\varphi^n$, from Phantom-S_N to the FM-LBM algorithm.
2. Compute the precursor density, C^{n+1}, for the next time step by iterating the FM-LBM algorithm N_t times. The total simulation time of the FM-LBM algorithm corresponds exactly to the time step of Phantom-S_N, such that $\Delta t = C_t N_t$, where C_t is the time conversion factor described in Section 3.1.6.
3. Export the delayed neutron source, $A_{\varphi C}C^{n+1}$, back to the Phantom-S_N algorithm.
4. Compute the neutron flux, φ^{n+1}, for the next time step using Equation 3.51.

These steps demonstrate that the simulation alternates between the FM-LBM and Phantom-S_N algorithms. Each algorithm advances the solution by a time step, Δt, with the output of one algorithm serving as the source term for the other in the subsequent iteration. Here, the time step Δt specifically refers to the time step used in the Phantom-S_N algorithm. The corresponding number of LBM time steps, N_t, can be calculated using the lattice conversion factor for time as $N_t = \frac{\Delta t}{C_t}$. To ensure consistency, the simulation parameters of the LBM algorithm should be chosen such that dividing the Phantom-S_N time step by the lattice conversion factor results in an integer number of LBM time steps.

Just as in the previous section, we have only described the coupling between the codes concerning precursor transport. However, thermal coupling can be easily implemented by exchanging additional information when the codes are switched. Specifically, in the FM-LBM algorithm, the power source is calculated from the neutron flux alongside the precursor source. This power source is then used as a source term in the heat equation during the FM-LBM simulations. Additionally, the FM-LBM algorithm exports the temperature field to the Phantom-S_N algorithm, which uses this information to update the nuclear cross-sections before solving for the neutron flux in the next time step.

3.4.4. Galerkin Projection

Now that we understand what information is transferred between the FM-LBM and Phantom-S_N codes, as well as when it is transferred, we need to clarify how this transfer occurs. The primary reason we cannot directly use the variable values exchanged between the two codes is that their spatial discretizations differ significantly. As explained in Section 3.3, the Phantom-S_N algorithm employs FEM discretization for the spatial dimension. In this approach, the neutron flux values across the domain are determined by expressing them as a linear combination of basis functions, with the coefficients for these functions calculated for each mesh element, as detailed in Equation 3.40. In contrast, as detailed in Section 3.1, the FM-LBM algorithm considers particle densities only at a specific set of lattice nodes. Consequently, macroscopic quantities are available solely at these lattice points, which, in our implementation using link-wise boundary schemes, correspond to the centers of the computational cells. To obtain values at other locations within the computational cells, interpolation techniques must be applied between the lattice points.

This fundamental difference in handling the spatial variable poses no issue when transferring information from Phantom-S_N to the FM-LBM algorithm, as variables can be written out on the lattice points using the relations in Equation 3.40. However, challenges arise when transferring variables from the FM-LBM algorithm back to Phantom-S_N. In this case, values at the lattice points must be transformed into coefficients of the basis functions for each element. For the precursor densities calculated by the FM-LBM algorithm, this issue is addressed using the Galerkin projection method [34], which is derived through

$$\begin{aligned} C(\mathbf{r}) &= \sum_i C_i h_i(\mathbf{r})\,, \\ \int_{V_e} C(\mathbf{r}) h_j(\mathbf{r}) \mathrm{d}\mathbf{r} &= \int_{V_e} \sum_i C_i h_i(\mathbf{r}) h_j(\mathbf{r}) \mathrm{d}\mathbf{r}\,, \\ \sum_q w_q C(\mathbf{r}_q) h_j(\mathbf{r}_q) &\simeq \sum_i C_i \int_{V_e} h_i(\mathbf{r}) h_j(\mathbf{r}) \mathrm{d}\mathbf{r}\,. \end{aligned} \tag{3.52}$$

In the first step the precursor field is expressed as a linear combination of basis functions $h_i(\mathbf{r})$ and corresponding coefficients, just as in Equation 3.40. In the second step both sides of the equation are multiplied with a different basis function $h_j(\mathbf{r})$ and both sides are integrated over the mesh volume. This corresponds to a projection of the precursor field onto the basis functions $h_j(\mathbf{r})$. In the final step, Gaussian quadrature is employed to approximate the continuous integration over a mesh element on the LHS by summing the integrand at specific quadrature points, $\mathbf{r}_q$, each multiplied by a corresponding weight, w_q, unique to that quadrature point. This approximation is analogous to the use of angular ordinates, where integrals are also replaced by summations, as shown in Equation 3.36. When explicitly writing out both indexations i and j, the final line of Equation 3.52 can be expressed as a matrix system, $\mathbf{y} = \mathbf{A}\mathbf{x}$, with the specific components given by

$$\mathbf{y} = \begin{bmatrix} \sum_q w_q C(\mathbf{r}_q) h_1(\mathbf{r}_q) \\ \sum_q w_q C(\mathbf{r}_q) h_2(\mathbf{r}_q) \\ \vdots \\ \sum_q w_q C(\mathbf{r}_q) h_N(\mathbf{r}_q) \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \int_{V_e} h_1(\mathbf{r}) h_1(\mathbf{r}) \mathrm{d}\mathbf{r} & \int_{V_e} h_1(\mathbf{r}) h_2(\mathbf{r}) \mathrm{d}\mathbf{r} & \dots & \int_{V_e} h_1(\mathbf{r}) h_N(\mathbf{r}) \mathrm{d}\mathbf{r} \\ \int_{V_e} h_2(\mathbf{r}) h_1(\mathbf{r}) \mathrm{d}\mathbf{r} & \int_{V_e} h_2(\mathbf{r}) h_2(\mathbf{r}) \mathrm{d}\mathbf{r} & \dots & \int_{V_e} h_2(\mathbf{r}) h_N(\mathbf{r}) \mathrm{d}\mathbf{r} \\ \vdots & \vdots & \ddots & \vdots \\ \int_{V_e} h_N(\mathbf{r}) h_1(\mathbf{r}) \mathrm{d}\mathbf{r} & \int_{V_e} h_N(\mathbf{r}) h_2(\mathbf{r}) \mathrm{d}\mathbf{r} & \dots & \int_{V_e} h_N(\mathbf{r}) h_N(\mathbf{r}) \mathrm{d}\mathbf{r} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix}. \tag{3.53}$$

Here, N denotes the number of basis functions per element, $h_i(\mathbf{r})$ are the basis functions, $\int_{V_e}$ denotes the integration over a mesh element, and C_i are the coefficients of the precursor densities to be determined. Now, Galerkin projection is performed as follows: For each element, the precursor density values are first determined at the quadrature points of that element through interpolation in the FM-LBM spatial grid as illustrated in Figure 3.12. The interpolation techniques used will be discussed in the next section. With these interpolated

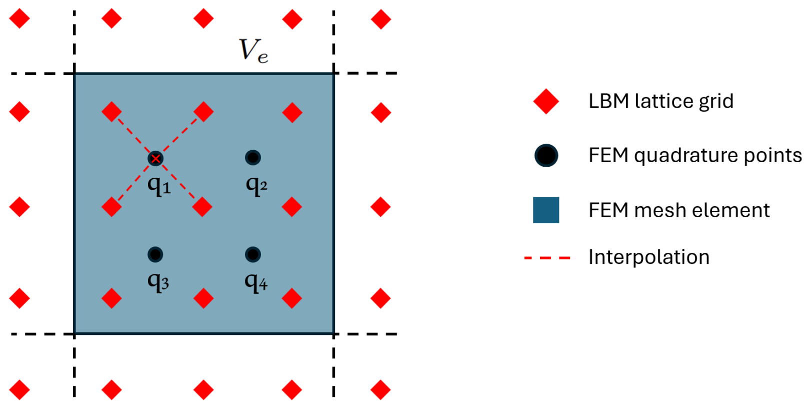

Figure 3.12: Overview of the differential spatial discretization in the FM-LBM and Phantom-S_N algorithms, along with the interpolation process within the LBM lattice grid to determine the physical variables at the quadrature points of the FEM mesh. When precursor densities are transferred from the FM-LBM algorithm to Phantom-S_N, the precursor field is first interpolated to compute the precursor densities at the quadrature points of each mesh element, denoted as $C(\mathbf{r}_q)$. Using this information, the vector $\mathbf{y}$ and matrix $\mathbf{A}$, as defined in Equation 3.53, are constructed. The coefficients of the basis functions for the FEM representation in Phantom-S_N are then obtained by solving the matrix system $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{x}$ represents the vector containing the coefficients.

values, the vector $\mathbf{y}$ and the matrix $\mathbf{A}$ can be constructed. The coefficients C_i are then determined for each element by solving for $\mathbf{x}$ in the matrix system $\mathbf{y} = \mathbf{A}\mathbf{x}$.

For the temperature field, a more straightforward approach is employed. Since the temperature field in the Phantom-S_N algorithm is used solely to adjust the nuclear cross-sections, there is no need to apply Galerkin projection to transform the FM-LBM values into FEM coefficients. Instead, we write the temperature values at the quadrature points through interpolation in the FM-LBM grid. The Phantom-S_N algorithm then uses these values to calculate the average temperature per mesh element as

$$T_{\textit{avg}} = \frac{1}{V_e}\int_{V_e} T(\mathbf{r})\mathrm{d}\mathbf{r} \simeq \frac{1}{V_e}\sum_q w_q T(\mathbf{r}_q)\,, \tag{3.54}$$

where V_e denotes the volume of the mesh element. This approach eliminates the need for all the FEM coefficients of the temperature field. This does mean, however, that nuclear cross-sections are not adjusted continuously throughout the spatial dimension. Instead, they are adjusted on a per-mesh-element basis.

3.4.5. B-Spline Interpolation

As explained in the previous section, the variables exported from the FM-LBM algorithm to Phantom-S_N must be written out on the quadrature points of the FEM mesh elements. Since these points do not coincide with the lattice grid of the FM-LBM algorithm, interpolation of the FM-LBM results is necessary to extract values at the quadrature points. To accomplish this interpolation, this research employs B-spline functions.

B-splines are piece-wise polynomial functions consisting of a series of base functions. More specifically, a B-spline of order p is defined as a linear combination of basis functions $B_{i,p}$ of degree p [42]. These basis functions are recursively defined through the zero-order basis functions, which in the case of a 1D application are defined as

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise.} \end{cases} \,. \tag{3.55}$$

Here, $\{t_i\}$ denotes the spline knots, which indicate the intervals of the piece-wise polynomial. This means within these knots, the polynomial is a continuous function. The basis functions of higher-order B-splines can be defined recursively from lower-order base functions through

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t)\,. \tag{3.56}$$

Figure 3.13 gives an overview of the zero-order, first-order, and second-order base functions in 1D, which are used for the creation of constant, linear, and quadratic splines. Spline functions are constructed by taking a linear combination of these basis functions with corresponding weights, resulting in a piecewise polynomial

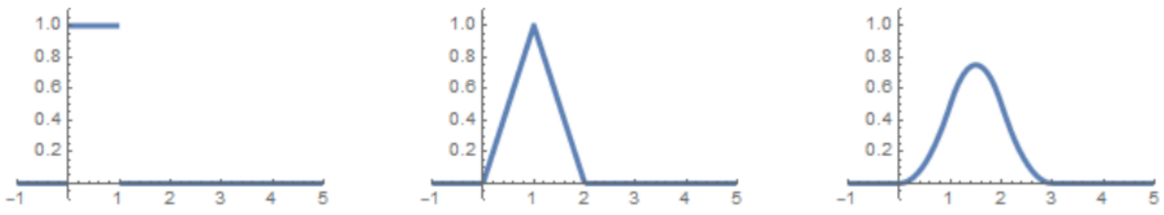

Figure 3.13: 1D zero-order, first-order, and second-order base functions for B-spline functions. These base functions are constructed using the expressions in Equations 3.55 and 3.56, and by using integer values as spline knots. This figure is adapted from from [31].

function. For a B-spline function of order $p \geq 1$ these piece-wise polynomials intersect at the knot points. Moreover, the derivatives of the piecewise polynomial up to order $p - 1$ are also equal at the knots [23].

In the context of interpolation on a lattice grid, B-splines can be utilized to fit a continuous function between the lattice points. This is achieved by placing the knots of the B-splines on the lattice nodes and selecting appropriate weights to ensure that the piecewise polynomial passes through all function values on the lattice grid. This method can also be extended to 2D and 3D by using basis functions appropriate for those dimensions. Figure 3.14 illustrates such interpolation using B-splines for both 1D and 2D. It is important to note that when zero-order splines are employed, the interpolation function returns the value of the nearest neighbor. In contrast, for higher orders, such as cubic splines ($p = 3$), the function between the lattice points becomes increasingly smooth. In this research, the Interpolations package created by [52] is used to implement the B-spline interpolation in Julia.

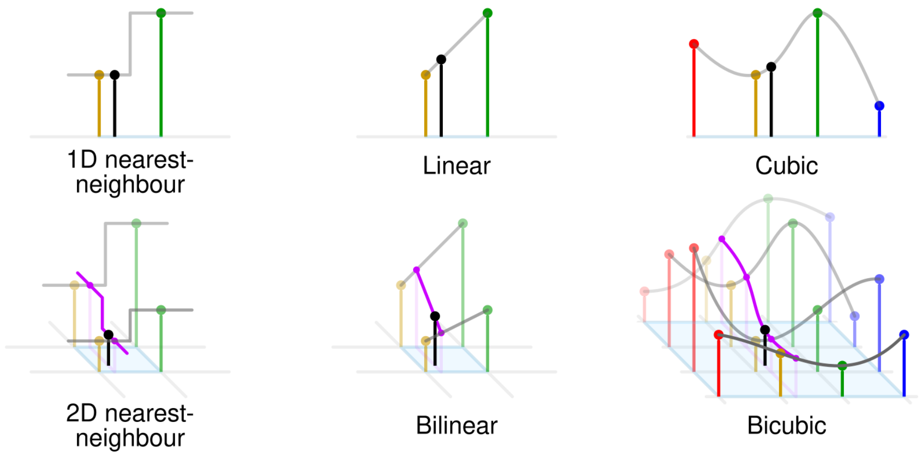

Figure 3.14: Illustration of interpolation in the FM-LBM scheme using B-splines for both 1D and 2D cases. The order of the basis functions determines the interpolation scheme, zero-order basis functions yield nearest neighbor interpolation, while higher-order basis functions produce smoother functions between the grid points of the lattice. This figure is obtained from [101].

3.4.6. Software Coupling

The previous sections outlined the interaction between the FM-LBM and Phantom-S_N algorithms, detailing what information is exchanged, when it is exchanged, and how the differences in their spatial discretizations are addressed. However, this discussion does not cover the specifics of their communication at the software level, particularly since the FM-LBM code is implemented in Julia, while the Phantom-S_N code is written in Fortran-90.

In our code implementation, Phantom-S_N is designated as the master operator. The primary reason for this is that the outer loop of the power method had already been implemented in Phantom-S_N at the start of this project. As a result, Phantom-S_N initiates and manages the execution of the FM-LBM algorithm. This is accomplished in Fortran-90 by making system calls to the Julia executable via the command line interface. Each time Julia is invoked, the Fortran-90 environment pauses and waits for the FM-LBM code to fully complete, ensuring that Phantom-S_N resumes operation only once the FM-LBM algorithm has finished running.

Furthermore, the exchange of information between the two codes is facilitated through I/O operations, where relevant data, such as variable values at quadrature points, is communicated via reading and writing to simple files. After each invocation of the FM-LBM algorithm, Julia saves the entire computational domain, including all distribution functions and macroscopic variables. This allows the full domain to be reloaded in subsequent iterations for continued flow development when an updated flux shape is provided by the Phantom-S_N code.

Lastly, all computations in this research were carried out on the DelftBlue supercomputer [1]. The DelftBlue supercomputer features both NVIDIA Tesla V100S GPU cores with 32 GB of video RAM and NVIDIA A100 GPU coress with 80 GB of video RAM, which were used for GPU accelerated FM-LBM algorithm. All performance results that will be mentioned for the FM-LBM algorithm were specifically measured using the NVIDIA A100 GPUs. Furthermore, all Phantom-S_N calculations were performed on the Intel Xeon CPU cores, supported by 250 GB of RAM, also available on the DelftBlue supercomputer.

4

Validation of Thermal Flow Simulation

Before testing the multiphysics coupling between the FM-LBM and Phantom-S_N codes, it is essential to first validate the accuracy of the FM-LBM code in simulating thermal flows. This intermediate benchmarking serves as a precautionary measure, ensuring that any errors encountered during multiphysics simulations are less likely to originate from the FM-LBM algorithm itself and more likely from the coupling between the two codes.

To validate the FM-LBM thermal flow code, we simulate the standard case of a side-heated 2D cavity, where natural convection drives the flow. In this validation case, only the precursor transport within the FM-LBM algorithm is excluded from testing. However, since the precursor transport is implemented in a manner almost identical to heat transport, we can reasonably assume the FM-LBM code functions correctly if it accurately simulates thermally driven flows.

Section 4.1 provides background on the benchmark setup, detailing the problem geometry, input parameters, and convergence criteria. Section 4.2 discusses the simulation results and compares them with established benchmark data. Finally, Section 4.3 includes a brief note on the performance of the FM-LBM code in terms of simulation speed.

4.1. Benchmark Setup

The simulation of natural convection in a side-heated 2D cavity is a well-established benchmark problem. In this study, the results obtained using our FM-LBM simulation tool are compared to those of [102], which employed a traditional finite difference method to solve the Boussinesq equations, as well as [122] and [11], who both utilized an FM-LBM algorithm. The benchmark geometry consists of a two-dimensional square cavity filled with air. No-slip boundary conditions are applied to the velocity field on all four walls. Furthermore, the left wall is cooled to a temperature $T_{Cold} < T_{ref}$, while the right wall is heated to $T_{Hot} > T_{ref}$. Here, T_{ref} represents the reference temperature, which is equal to $(T_{Cold} + T_{Hot})/2$. Finally, the top and bottom walls are adiabatic, meaning Neumann boundary conditions are applied to the temperature field. Figure 4.1 provides a schematic overview of the fluid domain and the associated boundary conditions.

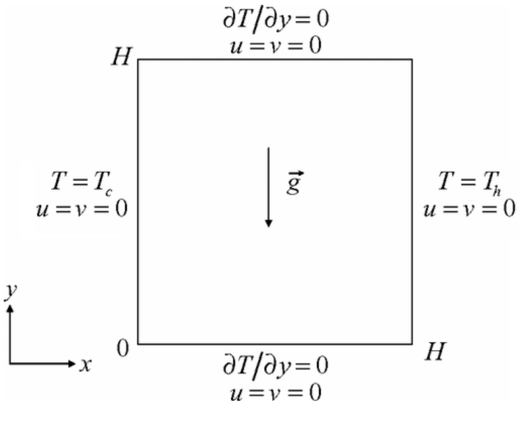

Figure 4.1: Problem domain of a side-heated 2D square cavity. No-slip boundary conditions are applied to the velocity field at all four walls. For the temperature field, Neumann boundary conditions are applied at the top and bottom walls, while the left wall is cooled to a temperature T_{Cold}, and the right wall is heated to a temperature T_{Hot}. This figure is obtained from [122].

In the case of natural convection within a side-heated cavity, the flow behavior is entirely governed by the Prandtl number (Pr) and the Rayleigh number (Ra), as defined in Table 2.1. Since air is the working fluid, the Prandtl number is fixed at $Pr = 0.71$. In the benchmark, the Rayleigh number is varied across three values, $Ra = 10^4$, 10^5, and 10^6, to analyze different temperature and velocity fields. By selecting the number of lattice points N (in an $N \times N$ grid), the viscosity ν^*, and the wall temperatures T^*_{Cold} and T^*_{Hot}, the remaining simulation parameters can be derived using the definitions of the dimensionless numbers provided in Table 2.1.

Table 4.1 presents an overview of the simulation parameters used in this research to replicate the benchmark results. Note that the parameter $\beta_{th} g^*$ denotes the product of the thermal expansion coefficient and the gravitational constant in lattice units. Since these terms only appear in the buoyancy force term, a single value is determined for their product based on the definition of the Rayleigh number.

Table 4.1: Overview of the FM-LBM parameters used for the simulation of a side-heated 2D square cavity along with the values in lattice units. The problem is fully defined by the Pr and Ra numbers. By selecting the number of lattice points along each axis, the temperatures of the walls, and the kinematic viscosity of the fluid, the remaining parameters can be calculated using the definitions of the dimensionless numbers given in Table 2.1.

Parameter	Simulation parameter	Unit
Ra	10^4, 10^5, 10^6	$-$
Pr	0.71	$-$
N	$101 - 501$	ls
T^*_{Hot}	20.0	lT
T^*_{Cold}	1.0	lT
ν^*	$1.85 \cdot 10^{-2}$	ls^{2}lt
α^*	$2.61 \cdot 10^{-2}$	ls^{2}lt
$\beta_{th} g^*$	$0.25 \cdot Ra\, N^{-3}$	ls lt^{-2}lT^{-1}

The FM-LBM algorithm is typically a transient solver. However, for simulations of stable flows progressing toward a steady-state, the number of iterations can also be determined based on a convergence criterion. The convergence criterion determines when the simulation has converged to the steady-state solution by comparing the fields at different time instances, and if so, terminates the simulation. In this benchmark study, we follow the criterion introduced by [122], in which the steady-state solution is achieved when

$$\max_i \left\{ \frac{\|\mathbf{u}(\mathbf{x}_i, t) - \mathbf{u}(\mathbf{x}_i, t - 500\Delta t)\|}{\|\mathbf{u}(\mathbf{x}_i, t)\|}, \sqrt{\left| \frac{T(\mathbf{x}_i, t) - T(\mathbf{x}_i, t - 500\Delta t)}{T(\mathbf{x}_i, t)} \right|} \right\} < 10^{-6}\,. \tag{4.1}$$

Here, $\mathbf{x}_i$ represents an arbitrary location on the lattice grid, Δt is the LBM time step, and $\|.\|$ is the Euclidean L^2 norm. This criterion implies that every 500 iterations, the macroscopic variables are calculated and compared to their values from 500 iterations earlier. If, at any lattice point, the relative change in these variables exceeds the threshold of 10^{-6}, the program has not yet fully converged.

4.2. Benchmark Results

Before comparing the full velocity and temperature fields with the benchmark studies, the grid dependence of the simulation results is examined. This is done using a similar approach to [122], where the Nusselt number at the cold boundary is analyzed with respect to the number of grid points used in the simulation. The Nusselt number at the cold boundary is calculated as

$$Nu_0 = -\frac{1}{\Delta T} \int_0^H \left. \frac{\partial T}{\partial x} \right|_{x=0} dy\,, \tag{4.2}$$

where, ΔT represents the temperature difference between the cold and hot walls. The second component of the state vector for the temperature field, as described in Equation 3.20, is used to compute the temperature gradient. The trapezoidal rule is then applied to numerically integrate this gradient.

Figure 4.2 illustrates the calculated Nusselt number at the cold wall for $Ra = 10^4$, 10^5, and 10^6 using various grid resolutions in the thermal flow simulations. The results reveal a clear grid dependence when the number of lattice points is insufficient. In this study, simulations were performed using grid sizes ranging from 101×101 to 501×501, with increments of 20 points along each axis between successive runs. Although the Nusselt number continues to decrease slightly for grid sizes larger than 501×501, indicating that the problem is not fully spatially converged, the relative change between $Nu_0(N = 501)$ and $Nu_0(N = 481)$ was found to be less than 0.05% across all Ra values. This level of precision is considered sufficient to minimize grid dependence and produce results that are consistent with benchmark data.

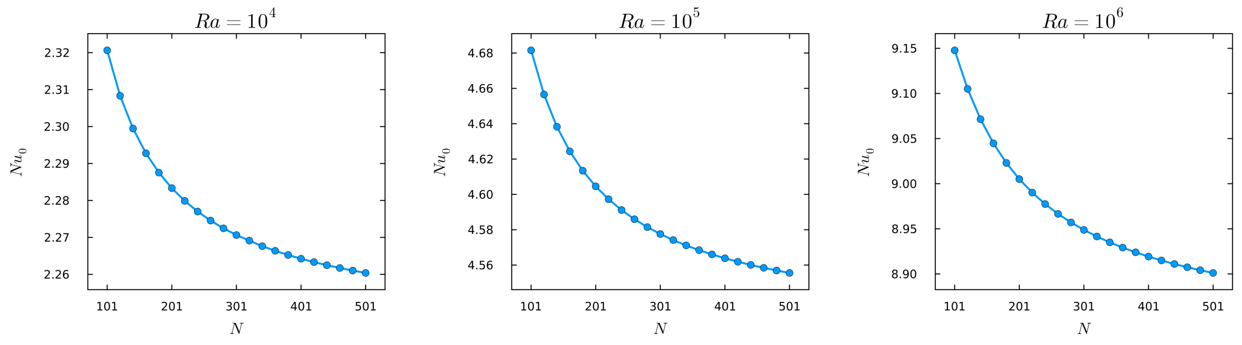

Figure 4.2: This figure illustrates the grid dependence of the simulation results for the side-heated square cavity. It does so by computing the Nusselt number at the cold wall using Equation 4.2 for various grid sizes. It is assumed that the Nusselt number converges for all Ra values when a grid size of 501×501 is used, thereby eliminating grid dependence from the simulation results.

Figure 4.3 presents the simulation result for the velocity and temperature fields for different values of Ra using a lattice grid of 501×501. The plots show that the simulated results qualitatively align well with those from the benchmark studies. However, a quantitative comparison is needed to fully evaluate the agreement between our simulation and the benchmarks. To enable this comparison, the velocity and spatial dimensions are rescaled to dimensionless values based on the simulation parameters, according to

$$\bar{\mathbf{u}} = \frac{N\mathbf{u}^*}{\alpha^*}\,, \qquad \bar{\mathbf{x}} = \frac{\mathbf{x}^*}{N}\,. \tag{4.3}$$

Using these rescaled variables, Table 4.2 summarizes the maximum horizontal velocity along the vertical centerline along with its corresponding y-coordinate, the maximum vertical velocity along the horizontal centerline along with its corresponding x-coordinate, and the Nusselt number at the cold wall. These values are directly compared to those reported in the benchmark studies.

Table 4.2 shows a strong agreement between the simulation results and the benchmark values, with relative errors below 1% for most parameters. Only the Nusselt number and the $\bar{y}_{max}$ parameter exhibit relative errors

Table 4.2: Quantitative comparison between simulation results and benchmark results of the side-heated 2D square cavity using a lattice grid of 501×501. This table shows the Nusselt number at the cold wall computed using Equation 4.2, the maximum horizontal velocity along the vertical centerline along with its corresponding y-coordinate, and the maximum vertical velocity along the horizontal centerline along with its corresponding x-coordinate. The table shows good aligment between simulation and benchmark results, with discrepancies for most parameter under 1%.

	Study	Nu_0	$\bar{u}_{x,max}$	$\bar{y}_{max}$	$\bar{u}_{y,max}$	$\bar{x}_{max}$
Ra = 10^4	[102]	2.238	16.178	0.177	19.617	0.881
	[122]	2.245	16.183	0.178	19.627	0.882
	[11]	2.232	16.189	0.175	19.631	0.880
	Current	2.260	16.189	0.177	19.637	0.881
	Error [%]	0.66 - 1.24	0.00 - 0.07	0.00 - 1.13	0.03 - 0.10	0.00 - 0.11
Ra = 10^5	[102]	4.509	34.73	0.145	68.59	0.934
	[122]	4.521	34.74	0.144	68.62	0.935
	[11]	4.543	34.74	0.147	68.55	0.933
	Current	4.556	34.84	0.145	68.62	0.935
	Error [%]	0.29 - 1.03	0.29 - 0.32	0.00 - 1.38	0.00 - 0.10	0.00 - 0.21
Ra = 10^6	[102]	8.817	64.63	0.150	219.36	0.962
	[122]	8.819	64.91	0.148	220.20	0.960
	[11]	8.890	65.05	0.150	220.54	0.963
	Current	8.901	64.87	0.151	220.56	0.963
	Error [%]	0.12 - 0.94	0.06 - 0.37	0.66 - 1.98	0.01 - 0.54	0.00 - 0.31

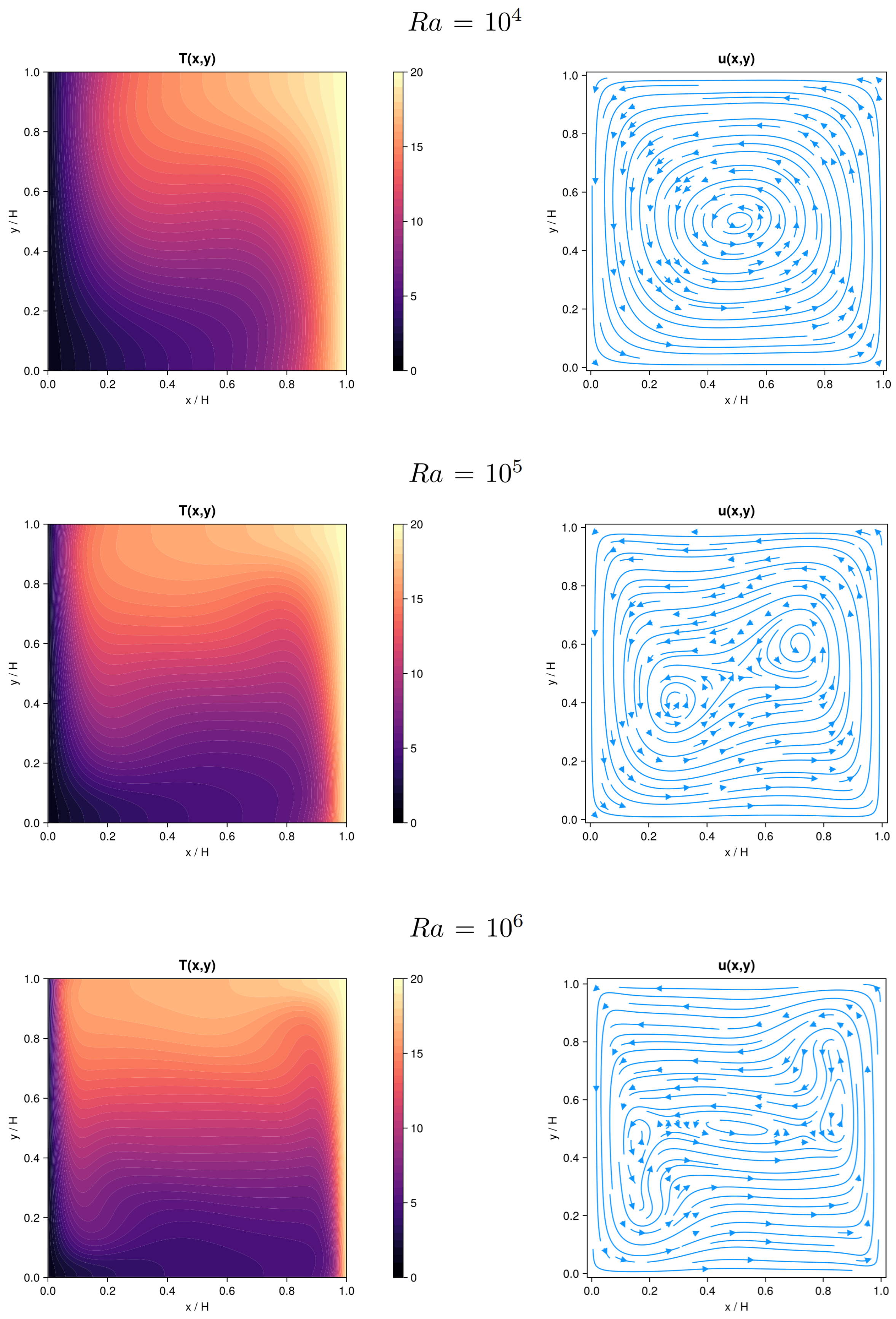

Figure 4.3: Simulation results of the side-heated 2D square cavity. This figure shows a streaming plot for the velocity field along with a heat map of the temperature field for each simulated Ra number. Additionally, all results are obtained using a Pr number of 0.71.

exceeding 1%. In the case of $\bar{y}_{max}$, the discrepancy arises from the small absolute value of the coordinate itself, with the absolute error being comparable to that of $\bar{x}_{max}$, and deviations appearing only at the third decimal place. For the Nusselt number, the simulation results consistently overshoot the benchmark values. This discrepancy can likely be attributed to the Nusselt number not being fully converged. As illustrated in Figure 4.2, using even larger grid sizes would likely yield results that more closely approximate the benchmark values.

4.3. Performance Analysis

A common practice in the field of LBM research to assess the simulation speed of an algorithm is to measure the number of cell evaluations per second, more commonly referred to as the number of lattice updates per second (LUPS). Due to the substantial size of these numbers, it is often more convenient to express this metric in terms of millions of lattice updates per second (MLUPS), which is defined as

$$\text{MLUPS} = \frac{N_{\text{Grid}} N_T}{T} \times 10^{-6} \,. \tag{4.4}$$

Here, N_{Grid} denotes the number of grid points, N_T represents the number of LBM time iterations performed in simulation, T is the total simulation time, and the factor 10^{-6} scales the value from LUPS to MLUPS. As indicated by the formula, a lattice update involves the complete execution of all simulation steps (propagation, collision, etc.) for a single LBM time iteration at a single lattice point.

Figure 4.4 shows the simulation speed of the FM-LBM algorithm for thermal fluid computation, comparing the performance of single versus double precision floating-point numbers. As outlined in Section 3.4.6, these measurements were conducted using an NVIDIA A-100 GPU. For clarity, a three-dimensional cubic problem domain was used in these measurements, meaning the total domain consists of N^3 lattice points. In this setup, the additional axis has periodic boundary conditions, effectively simulating the side-heated 2D square cavity problem but with an added computational load.

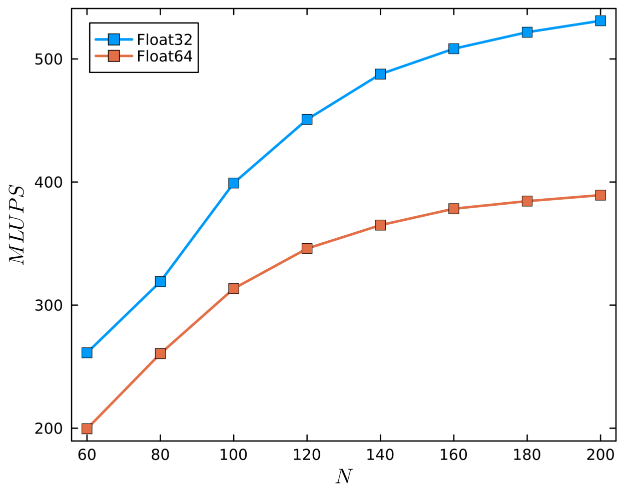

Figure 4.4: Simulation speed of the thermal FM-LBM algorithm, expressed in MLUPS, comparing performance between single and double precision floating-point numbers. Note that, in this context, a lattice update refers to the complete execution of all kernel functions for both the temperature and velocity fields at a single lattice point during one LBM time iteration. Furthermore, the simulation speeds are measured for different cubic domain sizes, where N represents the number of lattice nodes along a single axis (e.g., $N = 200$ corresponds to a total of 200^3 lattice points).

Figure 4.4 shows that our simulations achieve approximately 390 MLUPS when using double precision floating-point numbers. When compared to studies optimizing the LBM algorithm using similar methods (excluding multi-GPU implementations such as [116]), our performance is within a reasonable range. For instance, [98, 99] report simulation speeds up to 1200 MLUPS using a D3Q19 scheme. However, their studies focus solely on the velocity field, while our results include computations for both the velocity and temperature fields, effectively doubling the computational load. Additionally, those studies use a simpler SRT approach for the collision step with the BGK operator, whereas our research employs the FM-LBM algorithm. This method is significantly more computationally intensive due to the matrix multiplications involved in the collision kernel, further explaining the difference in performance.

Two additional observations can be made from Figure 4.4. First, switching from double precision to single precision floating-point numbers yields less performance improvement than expected. Since double precision calculations typically require approximately twice the resources and computing power, one might anticipate a near twofold increase in performance. However, the number of MLUPS only increases from around 390 to 530 for the maximum domain size of $N^3 = 200^3$. Second, the number of MLUPS rises with domain size. In principle, the computation time for a single LBM iteration should remain constant, irrespective of the number of lattice points, due to the GPU's parallel processing capabilities. However, this scaling behavior has its limits. Beyond a certain point, the GPU's theoretical advantage diminishes, and increasing the number of lattice points no longer results in higher MLUPS. This limitation primarily stems from the finite memory bandwidth of the GPU, which governs the data transfer rate between global memory and processing cores. As more processing cores attempt to access the memory simultaneously, the bandwidth eventually becomes saturated.

As described by [103], the finite memory bandwidth imposes an upper limit on the achievable lattice evaluations per second. This theoretical maximum can be calculated by dividing the memory bandwidth of the GPU with the number of reads and writes to memory per lattice node per LBM time iteration as

$$\text{MLUPS}^{th} = \frac{B}{10^6 \cdot 8 \cdot n}\,. \tag{4.5}$$

Here, MLUPSth is the theoretical upper limit of MLUPS, B denotes the memory bandwidth of the GPU in bytes per second, 10^6 scales LUPS to MLUPS, 8 represents the byte size of double precision floating-point numbers, and n is the number of reads and writes per lattice node per iteration. For the NVIDIA A100 GPU, the reported memory bandwidth is approximately 2 TB/s [76]. In our implementation of the FM-LBM algorithm for both velocity and temperature fields, the total number of reads and writes to global memory per iteration is approximately 400. Substituting these values, the theoretical maximum MLUPS we can achieve in our implementation is MLUPS$^{th} = 625$ for double precision computations. It is important to note that this upper limit accounts only for the finite bandwidth of global memory and excludes other factors. For instance, the number of reads and writes to shared memory, estimated at 1500, is not considered. Furthermore, overheads from the CPU, are also neglected. These include the time it takes to perform kernel launches, and the delay time from the synchronization of the CUDA cores between consecutive kernel launches. Nonetheless, the calculated upper limit illustrates that we are in reasonable range with our realized number of MLUPS. Furthermore, it explains the stagnation of MLUPS with increasing domain size and why MLUPS approaches a constant value at large domain sizes.

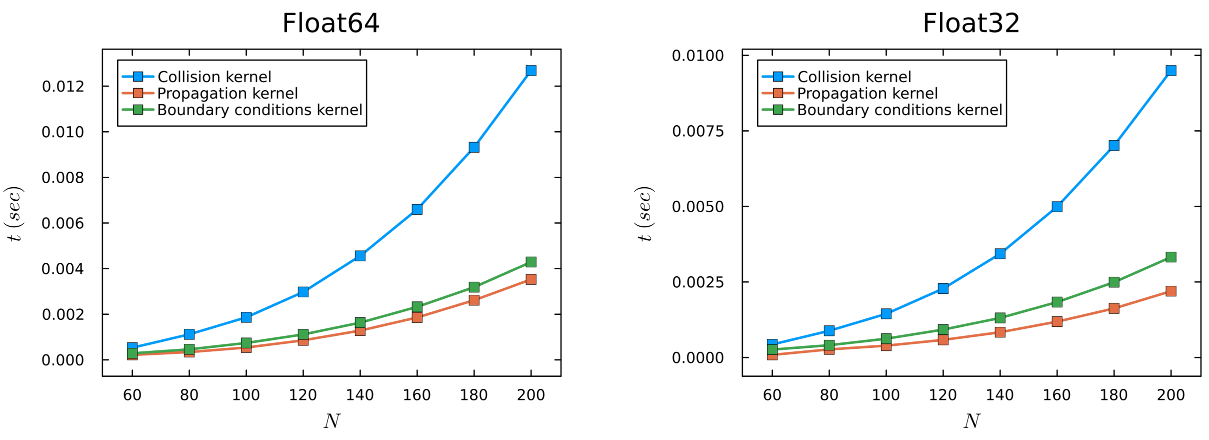

Figure 4.5: Execution time of various kernel functions for simulating both velocity and temperature fields, comparing single and double precision floating-point numbers. The execution times are measured for different cubic domain sizes, where N represents the number of lattice nodes along a single axis (e.g., $N = 200$ corresponds to a total of 200^3 lattice points).

The effect of memory bandwidth saturation is further demonstrated in Figure 4.5, which presents the average execution times of various kernel functions for different grid sizes. The execution times increase with domain size, following a cubic relationship. This scaling behavior is expected, as doubling the number of lattice nodes along each axis results in an eightfold increase in total lattice nodes, thereby necessitating a proportional increase in global memory reads and writes. Since the finite memory bandwidth is likely the bottleneck for simulation speed in our case, this cubic scaling is reflected in Figure 4.5. Additionally, the figure highlights that the collision kernel is the most computationally demanding part of the simulation. This is attributed to the two matrix multiplications performed within the kernel, which are resource-intensive for CUDA cores. In contrast, the propagation and boundary condition kernels primarily involve reading neighboring node values, making them less demanding. Furthermore, the collision kernel accounts for the highest number of memory operations, including all 1500 shared memory reads and writes and approximately half of the global memory reads and writes.

5
Validation of Steady-State Multiphysics Coupled Simulation

Having confirmed that the FM-LBM algorithm accurately models thermal flows, the next step is to validate its multiphysics coupling with the Phantom-S_N algorithm. To benchmark the fully coupled system, the Tiberga benchmark case [96] is employed. This benchmark study simulates the behavior of a fuel salt within a 2D lid-driven square cavity. The process begins by simulating individual variables independently, followed by progressively increasing the degree of coupling between the variables. This step-by-step approach helps isolate and identify potential errors within specific components of the multiphysics simulation tool.

Section 5.1 provides an overview of the benchmark study and includes a brief explanation of the convergence criteria employed. Section 5.2 details the methods used to quantify simulation errors and explains how simulation results are compared to the benchmark results. Sections 5.3 to 5.5 focus on steps 0.1 to 0.3 of the benchmark study, where the velocity, temperature, and neutron flux fields are solved independently. Sections 5.6 to 5.9 then cover steps 1.1 to 1.4, where these fields are progressively coupled, culminating into a fully coupled system at step 1.4. Finally, Section 5.10 provides a remark on the convergence speed of power method used in steady-state simulation.

It is important to note that throughout steps 0.1 to 1.4, all simulations are carried out using the power method described in Section 3.4.2. This means the results presented in this chapter correspond to steady-state conditions. In the next chapter, transient simulations will be carried out and compared to the benchmark results.

5.1. Benchmark Setup

Figure 5.1 illustrates the problem space of the benchmark, which consists of a two-dimensional, lid-driven square cavity measuring 2 m by 2 m. The velocity field is subject to no-slip boundary conditions on all four walls, with only the top wall moving at a non-zero velocity, denoted as U_{lid}. For both the temperature and precursor fields, von Neumann boundary conditions are applied at all four walls. A vacuum boundary condition is imposed on the neutron flux at each wall. Only density feedback effects are considered in the coupling between temperature and neutron flux, which means Doppler feedback effects are ignored. The cavity is filled with a LiF-BeF_2-UF_4 fuel salt. The properties of this salt are assumed to be uniform and temperature-independent throughout the domain. The atomic composition of the fuel salt is provided in Table 5.1. To facilitate the cooling of the salt, a volumetric heat sink is introduced, defined as

$$q(\mathbf{r}) = \gamma(T_{ref} - T(\mathbf{r})) \,, \tag{5.1}$$

where γ represents the volumetric heat transfer coefficient, and $T_{ref} = 900K$ is the reference temperature. The flow properties of this benchmark case are characterized by the Reynolds number (Re), Prandtl number (Pr), and Schmidt number (Sc) as defined in Table 2.1. Throughout the study, a Reynolds number of 40 is

Table 5.1: Atomic composition of the LiF - BeF_2 - UF_4 fuel salt used in the Tiberga benchmark study [96].

Isotope	^{6}Li	^{7}Li	^{9}Be	^{19}F	^{235}U
Atomic fraction (%)	2.11488	26.0836	14.0992	56.3969	1.30545

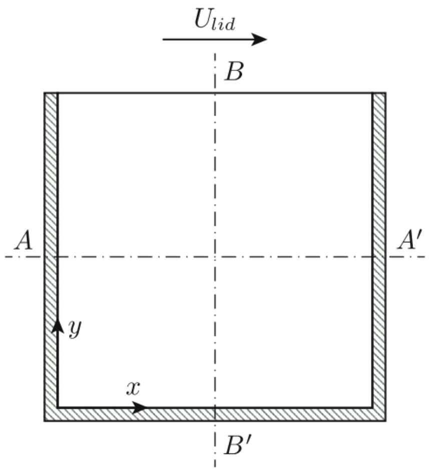

Figure 5.1: Problem domain of the Tiberga benchmark study. The domain consists of a lid-driven cavity that measures 2 m by 2 m. No-slip boundary conditions are applied to the velocity field on all four walls, with the top wall moving at a constant velocity U_{lid}. Von Neumann boundary conditions are imposed on the temperature and precursor fields at all four walls, while vacuum boundary conditions are applied to the neutron flux. Physical variables along centerlines AA' and BB' are analyzed to validate simulation results against the benchmark data [96].

maintained. The Prandtl number and Schmidt number are set to exceptionally high values of $3.075 \cdot 10^5$ and $2.0 \cdot 10^8$, respectively. As will be discussed later, these values are significantly reduced in this application to reduce the computational effort of the FM-LBM code.

For the neutron calculations, the JEFF-3.1 library [53] is employed at a reference temperature $T_{ref} = 900$ K, with neutronics data generated through the use of Serpent [60]. A comprehensive overview of the nuclear cross-sections and other relevant nuclear parameters from the benchmark is provided in Appendix A. The neutron flux is categorized into six energy groups, the upper energy bounds of which are specified in Table 5.2. Additionally, the precursor calculations incorporate eight precursor families. Their respective fractions and decay constants are detailed in Table 5.3 [96] .

Table 5.2: Upper bounds of the six energy groups as used in the Tiberga benchmark study [96] .

Group g	E_B (MeV)
1	$2.000 \cdot 10^1$
2	$2.231 \cdot 10^0$
3	$4.979 \cdot 10^{-1}$
4	$4.479 \cdot 10^{-2}$
5	$5.531 \cdot 10^{-3}$
6	$7.485 \cdot 10^{-4}$

Table 5.3: Fractions and decay constants of the eight precursor families as used in the Tiberga benchmark study [96].

Family i	λ_i (sec^{-1})	β_i (-)
1	$1.24667 \cdot 10^2$	$2.33102 \cdot 10^4$
2	$2.82917 \cdot 10^2$	$1.03262 \cdot 10^3$
3	$4.25244 \cdot 10^2$	$6.81878 \cdot 10^4$
4	$1.33042 \cdot 10^1$	$1.37726 \cdot 10^3$
5	$2.92467 \cdot 10^1$	$2.14493 \cdot 10^3$
6	$6.66488 \cdot 10^1$	$6.40917 \cdot 10^4$
7	$1.63478 \cdot 10^0$	$6.05805 \cdot 10^4$
8	$3.55460 \cdot 10^0$	$1.66016 \cdot 10^4$

Since the benchmark study also includes results generated with the Phantom-S_N algorithm for neutron flux simulation, the exact code specifications for Phantom-S_N are replicated from the benchmark case. This means the neutron flux is computed on a 50 $\times$ 50 structured square mesh, with a polynomial order of $p = 1$ for the basis functions. Additionally, the angular variable is discretized into six ordinates in each quadrant of the 2D problem space, meaning the S_6 equations are solved.

Finally, for convergence toward the steady-state solution, two distinct convergence criteria need to be defined. The first is for the power method, which serves as the outer loop in steady-state simulations. As explained in Section 3.4.2, the solution vector $\gamma^{(l)}$, which contains the coefficients for the neutron flux and precursor densities at power iteration l, converges to the true solution in the limit. In this research, the iterative process is terminated when

$$\frac{\left\|\gamma^{(l+1)} - \gamma^{(l)}\right\|}{\left\|\gamma^{(l+1)}\right\|} < 10^{-4}, \qquad \text{and} \qquad \left|\frac{k_{\mathit{eff}}^{(l+1)} - k_{\mathit{eff}}^{(l)}}{k_{\mathit{eff}}^{(l+1)}}\right| < 10^{-6}. \tag{5.2}$$

This indicates that the relative changes in the solution vector and the effective multiplication factor are less than 10^{-4} and 10^{-6}, respectively.

Additionally, within the outer loop of the power method, the FM-LBM algorithm is called at each iteration. The simulations carried out by this algorithm also progress toward a steady-state solution for the given precursor and power sources. Therefore, a convergence criterion must also be defined for the FM-LBM algorithm. This is done by extending the convergence criterion given in Equation 4.1 to include the eight precursor densities as

$$\max_{i,j}\left\{\frac{\|\mathbf{u}(\mathbf{x}_i,t)-\mathbf{u}(\mathbf{x}_i,t-500\Delta t)\|}{\|\mathbf{u}(\mathbf{x}_i,t)\|},\sqrt{\left|\frac{T(\mathbf{x}_i,t)-T(\mathbf{x}_i,t-500\Delta t)}{T(\mathbf{x}_i,t)}\right|},\sqrt{\left|\frac{C_j(\mathbf{x}_i,t)-C_j(\mathbf{x}_i,t-500\Delta t)}{C_j(\mathbf{x}_i,t)}\right|}\right\}<10^{-4}. \quad (5.3)$$

Note that the threshold of the FM-LBM algorithm has been slightly relaxed from 10^{-6}, as specified in Equation 4.1, to 10^{-4}. This adjustment was implemented to reduce the computation time required for the complete simulation tool to achieve a steady-state solution, without losing accuracy. Further explanations regarding this choice are provided in Section 5.10.

5.2. Error Quantification

To compare the simulation results with those of the benchmark, various quantities are evaluated along the horizontal and vertical centerlines, labeled AA' and BB' in Figure 5.1. For a consistent comparison of neutron flux in steady-state computations, the final neutron flux is normalized to a reference power of $P_{ref} = 1$ GW.

The Tiberga benchmark is a collaborative effort involving researchers from four different universities, each developing their own multiphysics simulation tools for MSFR core simulations. As a result, the benchmark includes the outcomes from four distinct simulation codes, enabling a comparative analysis of their outputs. In this study, the simulation results are assessed against the benchmark by analyzing the discrepancy in simulation variables, defined as

$$\varepsilon_Q = \sqrt{\frac{\sum_{i=1}^{N_p}(Q_c(\mathbf{x}_i)-Q_{avg}(\mathbf{x}_i))^2}{\sum_{i=1}^{N_p}Q_{avg}^2(\mathbf{x}_i)}}. \quad (5.4)$$

In this equation, N_p represents the number of evaluation points for the variable Q being compared to the benchmark. $Q_c(\mathbf{x}_i)$ denotes the results from the current simulation tool at evaluation point $\mathbf{x}_i$, while $Q_{avg}(\mathbf{x}_i)$ represents the average results from the benchmark study at evaluation point $\mathbf{x}_i$, calculated as the mean of the outputs from the various benchmark codes. When the evaluation points of the benchmark results do not align with the lattice grid, cubic B-spline interpolation, as described in Section 3.4.5, is applied.

5.3. Step 0.1: Velocity field

The first step of the benchmark is to simulate the velocity field independently of the other fields. Since the velocity field is driven by the upper lid, no external input is required for this simulation. Although this step may appear somewhat redundant, given that our FM-LBM has already demonstrated its ability to accurately simulate flow, it is still included for completeness.

The parameters relevant to the simulation of the velocity field are presented in Table 5.4 in both physical and LBM units. The conversion from physical to LBM units was performed using the LBM conversion factors as outlined in Section 3.1.6.

Table 5.4: Simulation parameter used for the computation of step 0.1 of the Tiberga benchmark case, which involves the independent simulation of the velocity field in the lid-driven square cavity. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values.

Parameter	Physical value	Physical unit	LBM value	LBM unit
Re	40	–	40	–
L	2.0	m	501	ls
ρ	$2.0 \cdot 10^3$	kg m^{-3}	1.00	lm ls^{-2}
ν	$2.5 \cdot 10^{-2}$	m^{2}s^{-1}	0.63	ls^{2}lt^{-1}
U_{lid}	0.5	m s^{-1}	0.05	ls lt^{-1}

Before presenting the simulation results, the grid dependence is first analyzed by evaluating the discrepancy with the benchmark results as a function of grid size. As described in the previous section this is done by comparing the velocity field on the horizontal (AA') and vertical (BB') centerlines to the benchmark results, where the discrepancies are quantified using Equation 5.4. Figure 5.2 illustrates the average discrepancy with the benchmark as a function of the number of grid points N per dimension, where the full domain consists of $N \times N$ lattice nodes in total.

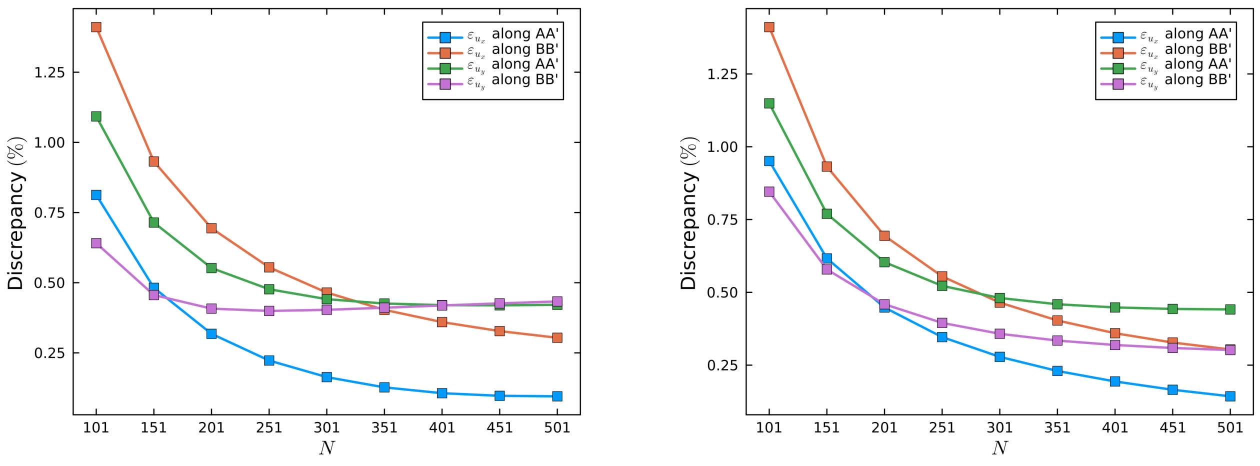

Figure 5.2: Average discrepancies of the horizontal and vertical components of the simulated velocity field along the horizontal and vertical centerlines as a function of grid size. The x-axis shows the number of gridpoints N along a single dimension, where in total there are $N \times N$ lattice points. The discrepancies are calculated by comparing our simulation results with the average The left plot includes the results from the PSI benchmark code, while they are omitted in the right plot due to deviations in these results with respect to the other benchmark codes. The discrepancies are determined by comparing our simulated velocity field to the ensemble mean velocity field, calculated by averaging the results from the various benchmark codes, as explained in Section 5.2.

Figure 5.2 contains two plots showing the discrepancy as a function of grid size: one with (left) and one without (right) the results from the PSI code in the benchmark. The PSI results are excluded in the second plot as they tend to deviate, particularly for the u_y variable along the vertical centerline, causing a slight increase in ε_{u_y} beyond $N = 201$, as seen in the left plot of Figure 5.2. In the right plot, the discrepancies converge at a grid size of 501×501, indicating that grid dependence has diminished from the simulation results beyond this point.

Figures 5.3 and 5.4 present the simulation results for the horizontal and vertical components of the velocity field, respectively, for a grid size of 501 $\times$ 501. Both figures compare the simulation results with benchmark data from various codes and illustrate the velocity field along the horizontal and vertical centerlines. Our simulation results show strong agreement with the benchmark data, with average discrepancies of 0.10% for u_x along AA', 0.26% for u_x along BB', 0.42% for u_y along AA', and 0.43% for u_y along BB'.

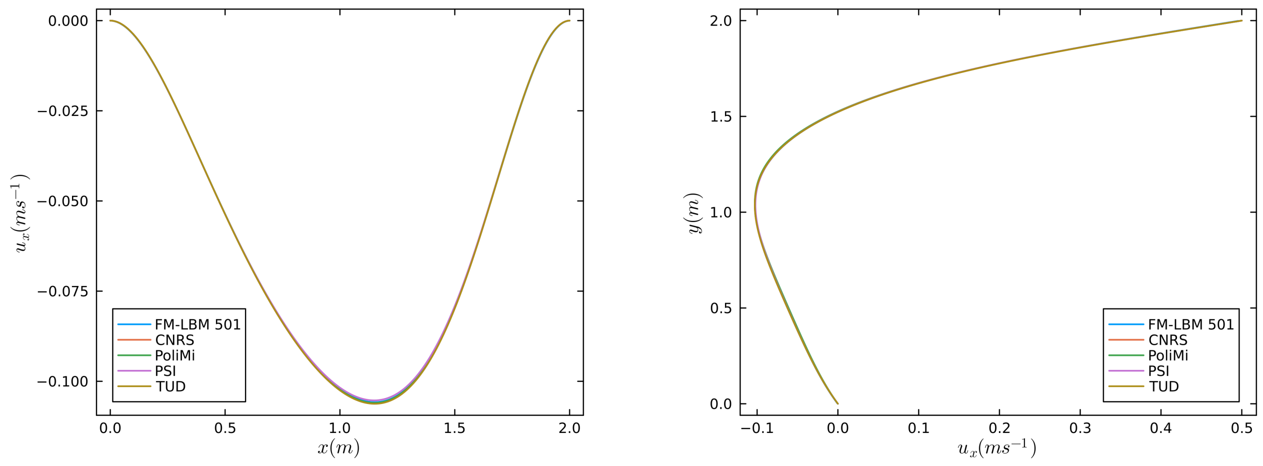

Figure 5.3: Simulation results of step 0.1 form the Tiberga benchmark. This figure shows the horizontal component of the velocity field along the horizontal and vertical centerlines along with the benchmark data. The results from this research are obtained using the FM-LBM algorithm on a simulation grid of 501×501.

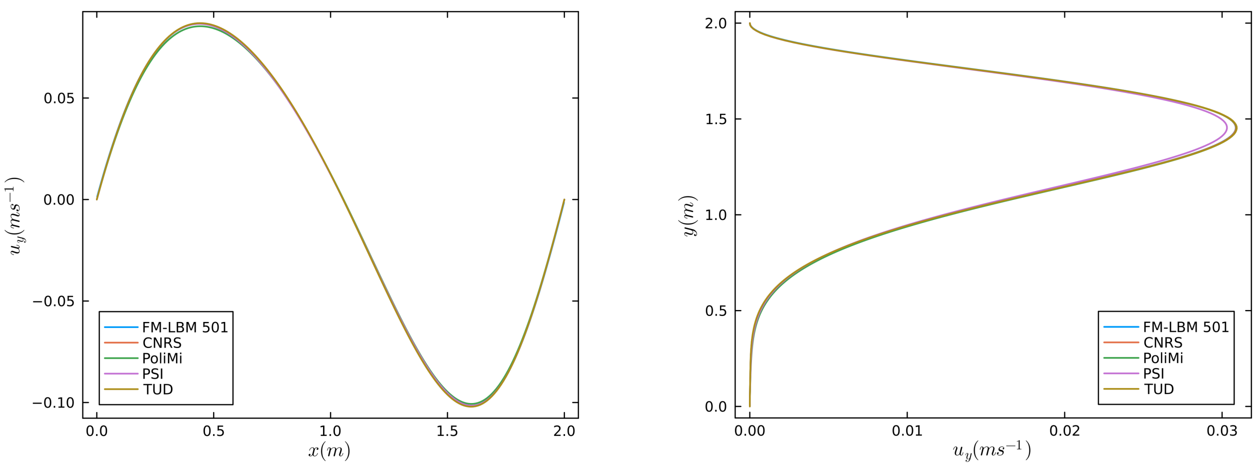

Figure 5.4: Simulation results of step 0.1 form the Tiberga benchmark. This figure shows the vertical component of the velocity field along the horizontal and vertical centerlines along with the benchmark data. The results from this research are obtained using the FM-LBM algorithm on a simulation grid of 501×501.

5.4. Step 0.2: Neutron flux

In step 0.2 of the Tiberga benchmark, the neutron flux shape is calculated independently from all other fields. As mentioned in the previous section, the benchmark study includes results generated with the Phantom-S_N algorithm. Specifically, the TUD-S6 results apply the same specifications used in this research, where the S_6-equations are solved. The primary objective of this step is to accurately replicate the TUD-S6 benchmark results. Given that Phantom-S_N is a sophisticated code with various configurations and use cases, successfully reproducing these results ensures that the code is being used correctly.

The relevant parameters for simulating the neutron flux are listed in Table 5.5. These parameters are provided in physical units only, as Phantom-S_N operates with SI units. The temperature is uniformly set to the reference value, resulting in constant nuclear cross-sections throughout the domain. Additionally, the neutron flux is normalized at the end of the power iteration such that the total reactor power matches its reference value.

Table 5.5: Simulation parameter used for the computation of step 0.2 of the Tiberga benchmark case, which involves the independent simulation of the neutron flux in the square cavity, with a uniform temperature distribution equal to the reference temperature.

Parameter	Physical value	Physical units
P_{ref}	$1.0 \cdot 10^9$	W
$T = T_{ref}$	900	K

Figure 5.5 shows the simulated fission rate density along the horizontal centerline AA', derived from the neutron flux, alongside the benchmark results. Due to the problem's symmetry, the results along the vertical centerline are identical. The results obtained in this research are in exact agreement with the TUD-S_6 benchmark results. Specific fission rate density values at key coordinates, as documented in the benchmark study, match those simulated here. These values are presented in Table 5.6 for completeness. Moreover, the same reactivity of $\rho = 578.1$ pcm observed in the TUD-S_6 benchmark is reproduced in this study. As a result, we can conclude that the neutronics simulations successfully replicates the benchmark results in the absence of flow, confirming the correct operation of the Phantom-S_N code.

5.5. Step 0.3: Temperature field

In step 0.3 of the Tiberga benchmark, the temperature field is simulated independently from the other fields. To facilitate thermal convection, the velocity field simulated in step 0.1 is imposed as a fixed velocity field. Additionally, the neutron flux shape calculated in step 0.2 is used to calculate a power source through Equation 3.42, which is also held constant throughout the simulation.

Although the code has already demonstrated its ability to simulate thermal flows, this step is again included for completeness. It also serves as the initial test to ensure the correct transfer of data from Phantom-S_N

Coordinate	$\int_E \Sigma_f \Phi \mathrm{d}E$ (m^{-2}s^{-1})
(0.00, 1.00)	$6.833 \cdot 10^{17}$
(0.25, 1.00)	$7.463 \cdot 10^{18}$
(0.50, 1.00)	$1.300 \cdot 10^{19}$
(0.75, 1.00)	$1.667 \cdot 10^{19}$
(1.00, 1.00)	$1.796 \cdot 10^{19}$
(1.25, 1.00)	$1.667 \cdot 10^{19}$
(1.50, 1.00)	$1.300 \cdot 10^{19}$
(1.75, 1.00)	$7.463 \cdot 10^{18}$
(2.00, 1.00)	$6.833 \cdot 10^{17}$

Table 5.6: Fission rate density at key coordinates along the AA' centerline calculated using the simulated neutron flux from step 0.2 of the Tiberga benchmark case. These results exactly match those from the TUD-S6 benchmark results.

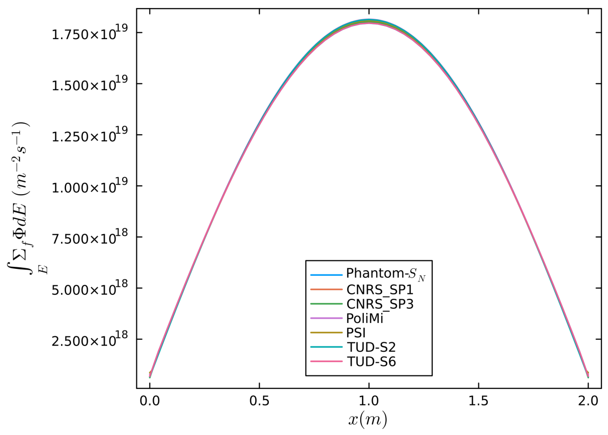

Figure 5.5: Simulated fission rate density along the horizontal centerline from Step 0.2 of the Tiberga benchmark case, alongside the benchmark results. In this step, the neutron flux is calculated independently of all other fields.

to the FM-LBM algorithm, as the neutron flux must be translated from the FEM basis function coefficients to values on the FM-LBM grid using the expressions in Equation 3.40.

The relevant parameters for simulating the temperature field are listed in Table 5.7. The thermal diffusivity, α, is calculated using the Pr number and the fluid properties detailed in Table 5.4. The reference temperature is copied from Table 5.5 and is included to provide its corresponding LBM value.

Table 5.7: Simulation parameter used for the computation of step 0.3 of the Tiberga benchmark case, which involves the independent simulation of the temperature field in the lid-driven square cavity. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values. Note the reduction of the Prandtl number in our FM-LBM simulations compared to the benchmark value. This reduction leads to a different physical value for the thermal diffusivity.

Parameter	Physical value	Physical unit	LBM value	LBM unit
Pr	$3.075 \cdot 10^5$	-	1000	-
γ	$1.0 \cdot 10^6$	W m^{-3}K^{-1}	$7.17 \cdot 10^{-3}$	lm ls^{-1}lt^{-3}lT^{-1}
C_p	$6.15 \cdot 10^6$	J m^{-3}K^{-1}	$1.10 \cdot 10^2$	lm ls^{-1}lt^{-2}lT^{-1}
T_{ref}	900	K	1.0	lT
α	Benchmark: $8.13 \cdot 10^{-8}$ Reduced Pr: $2.50 \cdot 10^{-5}$	m^{2}s^{-1}	$6.27 \cdot 10^{-4}$	ls^{2}lt^{-1}

As indicated in Table 5.7 and discussed earlier in Section 5.1, the Prandtl number has been significantly reduced compared to the value used in the benchmark case. This adjustment is made as a Prandtl number of $3.075 \cdot 10^5$ is unrealistically high for an MSFR core, which typically operates within a range of 1 to 35 [4]. But more importantly, these high Prandtl values significantly increase the computational effort of the FM-LBM algorithm. Higher Prandtl numbers demand increased viscosities (ν^*) to maintain stability in LBM simulations. In turn, this requires a larger simulation grid to preserve a constant Reynolds number, greatly increasing computational costs. Additionally, larger grids slow the convergence to steady-state solutions, making high Prandtl number simulations even more computationally intensive.

To understand the impact of reducing the Prandtl number on the simulation results, we must understand the meaning of the Prandtl number. This can be achieved by non-dimensionalizing the heat equation from Equation 2.4 as

$$PrRe[\tilde{\mathbf{u}} \cdot \tilde{\nabla}\tilde{T}] = -\tilde{\nabla}^2\tilde{T} + K(GrPr)^{1/4}(1-\tilde{T})\,. \tag{5.5}$$

In this equation, K is a constant related to the Nusselt number, which represents the ratio of total heat transfer to heat transfer via conduction, and Gr is the Grashof number, characterizing the ratio of buoyancy forces to viscous forces in the fluid. The tilde notation denotes non-dimensionalized variables, scaled to their characteristic values as outlined in Section 2.1.3.

From Equation 5.5, it is evident that increasing the Prandtl number enhances the relative influence of the convection and heat sink terms over diffusion. This implies that, for the high Prandtl number used in the benchmark case, diffusion effects can essentially be neglected. However, in our system, we already expect diffusion to be minimal due to relatively small temperature gradients in the domain, which is a result of the volumetric heat sink distributed uniformly throughout the domain. Therefore, it is expected that, even with lower Prandtl numbers, the convection and heat sink terms will dominate. As a result, further increasing the Prandtl number is unlikely to significantly affect the temperature field, as the diffusion term is already negligible. The only notable temperature variations are expected to arise from the shape of the power source, as depicted in Figure 5.5. However, beyond a certain Prandtl number, even the diffusion effects from these gradients will become negligible.

To validate our assumption that reduced Prandtl numbers can yield similar simulation results compared to the benchmark, the left graph in Figure 5.6 shows the average discrepancy in the simulated temperature along the centerlines for different Prandtl numbers on a 501×501 lattice grid. Additionally, the right graph of Figure 5.6 illustrates the grid dependence of the simulation results, similar to Figure 5.2, this figure shows the average discrepancy plotted against the number of grid points.

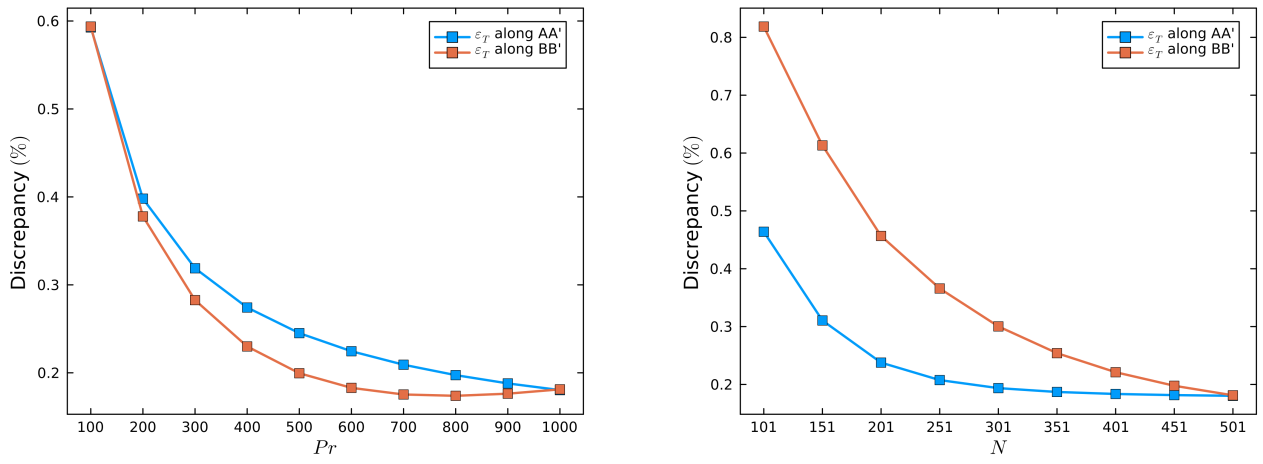

Figure 5.6: Average discrepancy with benchmark results from step 0.3 as a function of Prandtl number (left) and grid size (right). In the left figure, the simulation where performed using a grid size of 501×501. In the right figure, the simulation where performed using a Prandtl number of 1000. Furthermore, in the right plot, the x-axis denotes the number of lattice nodes per dimension, which means the total grid consists of $N \times N$ lattice points. The discrepancies are determined by comparing our simulated velocity field to the ensemble mean velocity field, calculated by averaging the results from the various benchmark codes, as explained in Section 5.2.

Figure 5.6 shows that the grid dependency has diminished for simulations using a grid size of 501×501 and higher. Additionally, the average discrepancy does not decrease further when Prandtl numbers greater than

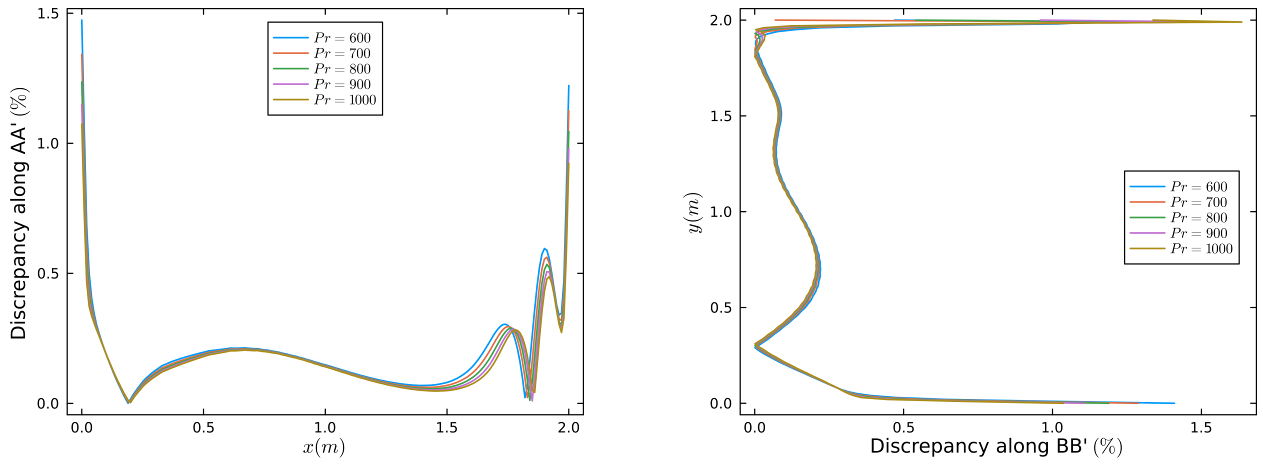

Figure 5.7: Discrepancy per evaluation point with benchmark results from step 0.3 as a function of Prandtl number. The left figure shows the discrepancy along the horizontal centerline, while the right figure shows the discrepancy along the vertical centerline. In both figures, the simulation where performed using a lattice grid of 501×501. The discrepancies are determined by comparing our simulated velocity field to the ensemble mean velocity field, calculated by averaging the results from the various benchmark codes, as explained in Section 5.2.

1000 are used. This confirms our assumption that the benchmark results can be accurately reproduced with reduced Prandtl numbers. To further analyze the effect of reduced Prandtl numbers on simulation errors, Figure 5.7 presents the average discrepancy at each evaluation point along the horizontal (left) and vertical (right) centerlines for various Prandtl values. These figures show that the average discrepancy is highest near the boundaries, which is a consequence of the temperature gradients being the steepest at the boundary due to the Neumann boundary conditions. Consequently, the approximation of using a reduced Prandtl number performs less effectively at the boundaries. Moreover, the figures also indicate that increasing the Prandtl number only slightly reduces simulation errors near the boundaries, while further from the boundary, the solution has already converged for relatively small Prandtl values.

Finally, Figure 5.8 presents the resulting temperature field along the horizontal (left) and vertical (right) centerlines for a simulation grid of 501×501 with a Prandtl number of 1000, compared to the benchmark results. The figure demonstrates good alignment between the current implementation and the benchmark data, with average discrepancies of 0.18% along the AA' centerline and 0.18% along BB'.

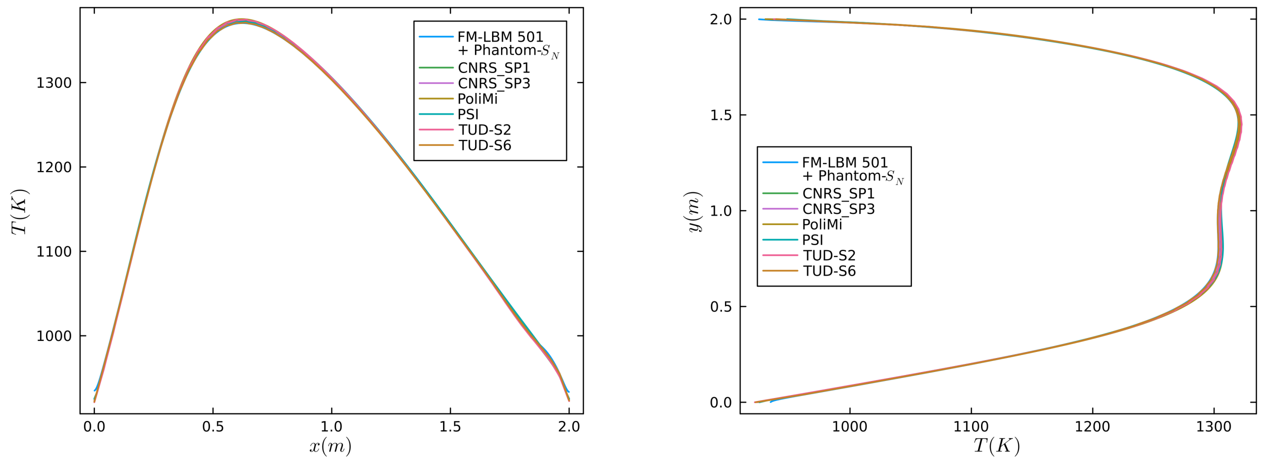

Figure 5.8: Simulated temperature field along the horizontal (left) and vertical (right) centerlines in step 0.3 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000, compared to the benchmark results. The label FM-LBM 501 + Phantom-S_N indicates that the simulations are performed using the FM-LBM algorithm on a 501×501 lattice grid with the power source being generated by Phantom-S_N.

5.6. Step 1.1: Circulating fuel

Step 1.1 of the Tiberga benchmark addresses precursor transport due to circulating fuel. In this step, the first coupling is established between neutronics and flow calculations. To simplify the problem, the velocity field remains fixed based on the results from step 0.1, and temperature effects are disregarded. This means, a uniform temperature field at the reference temperature is assumed across the domain, leading to constant nuclear cross-sections. As a result, this step only simulates neutron flux and precursor transport under the imposed velocity field. The primary objective of this step is to assess the reactivity change resulting from precursor transport. Additionally, it offers the first opportunity to test the two-way coupling between the FM-LBM code and the Phantom-S_N code for precursor transport.

The relevant parameters for simulating precursor transport are provided in Table 5.8. These should be complemented with the parameter listed in Table 5.4 and 5.5 for the velocity and neutron fields, respectively.

Table 5.8: Simulation parameter used for the computation of step 1.1 of the Tiberga benchmark case, which involves the simulation of precursor transport due to circulating fuel. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values. These parameter should be complemented with the parameter listed in Table 5.4 and 5.5 for the velocity and neutron fields to obtain the full list of parameters used in this simulation step. Note the reduction of the Schmidt number in our FM-LBM simulations compared to the benchmark value. This reduction leads to a different physical value for the molecular diffusivity.

Parameter	Physical value	Physical unit	LBM value	LBM unit
Sc	$2.0 \cdot 10^8$	-	1500	-
D	Benchmark: $1.25 \cdot 10^{-10}$ Reduced Sc: $1.67 \cdot 10^{-5}$	m^{2}s^{-1}	$4.18 \cdot 10^{-4}$	ls^{2}lt^{-1}

Similar to the Prandtl number adjustment in step 0.3, the Schmidt number is also significantly reduced, as shown in Table 5.8. This reduction follows the same rationale as for the Prandtl number: higher Schmidt numbers require larger viscosity values (ν^*) to maintain numerical stability, which in turn necessitate larger simulation grids to preserve a constant Reynolds number. The grid sizes associated with these very high Schmidt values lead to computational costs that are impractical for this research. Consequently, the Schmidt number is significantly lowered.

The impact of this reduced Schmidt number can again be analyzed through a non-dimensional approach, this time focusing on the advection-diffusion equation for precursor transport. By disregarding the fission source term in Equation 2.16, we can reformulate the advection-diffusion equation as

$$ScRe[\tilde{\mathbf{u}} \cdot \tilde{\nabla}\tilde{C}_i] = \tilde{\nabla}^2\tilde{C}_i - Da\tilde{C}_i\,. \tag{5.6}$$

Here, Da represents the Damköhler number, which relates the diffusion timescale to the decay rate of the precursors and increases with the Schmidt number. The tilde notation indicates non-dimensionalized variables, scaled by their characteristic values. As with the Prandtl number, increasing the Schmidt number amplifies the effects of the convection and the sink terms over diffusion. This implies that high Schmidt values, as in the benchmark case, effectively ignore diffusion effcts. Given the small precursor gradients observed in our simulations, diffusion effects are expected to be already small, hence, even with a reduced Schmidt number, convection and sink terms can dominate. As a result, we expect that further increasing the Schmidt number is therefore unlikely to notably impact the precursor field.

For simplicity, this non-dimensional analysis excludes the fission source term in Equation 2.16. Including this term would introduce a second Damköhler number that relates the diffusion timescale to the rate of precursor production from fission events, which also increases with the Schmidt number. However, this does not alter the conclusion that high Schmidt numbers effectively allow diffusion effects to be neglected.

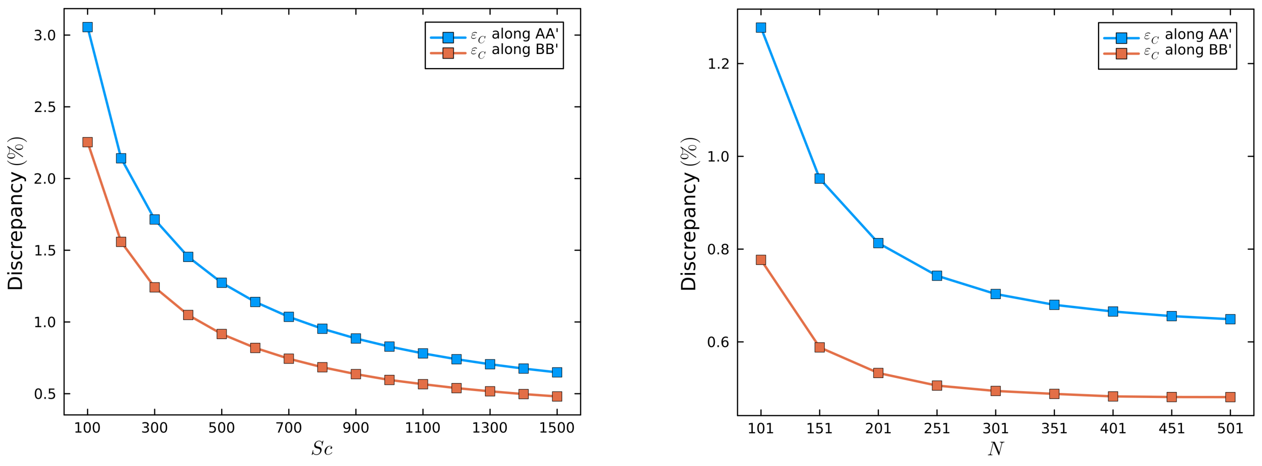

Figure 5.9: Average discrepancy with benchmark results from step 1.1 as a function of Schmidt number (left) and grid size (right). In the left figure, the simulation where performed using a grid size of 501×501. In the right figure, the simulation where performed using a Schmidt number of 1500. Furthermore, in the right plot, the x-axis denotes the number of lattice nodes per dimension, which means the total grid consists of $N \times N$ lattice points. The discrepancies are determined by comparing our simulated velocity field to the ensemble mean velocity field, calculated by averaging the results from the various benchmark codes, as explained in Section 5.2.

To validate our assumption that a reduced Schmidt number yields simulation results comparable to the benchmark, the left graph in Figure 5.9 presents the average discrepancy in the simulated delayed neutron source, calculated as $\sum_i \lambda_i C_i$, along the centerlines for various Schmidt numbers on a 501×501 lattice grid. This figure shows that a Schmidt number of 1500 results in an average discrepancy of approximately 0.5% on both centerlines, and further increasing the Schmidt number does not improve accuracy, confirming our assumption. Additionally, the right graph in Figure 5.9 illustrates the grid dependency of the simulation results, displaying the average discrepancy as a function of grid size. This figure illustrates that grid dependency diminishes for simulations with grid sizes of 501×501 and larger.

Similarly to step 0.3, Figure 5.10 presents the average discrepancy of the delayed neutron source at each evaluation point along the horizontal (left) and vertical (right) centerlines for various Schmidt numbers to identify where simulation errors are largest. These figures again show that the average discrepancy is highest near the boundaries, which is a consequence of the precursor gradients being the steepest at the boundary due to the Neumann boundary conditions. Consequently, using a reduced Schmidt number is less effective

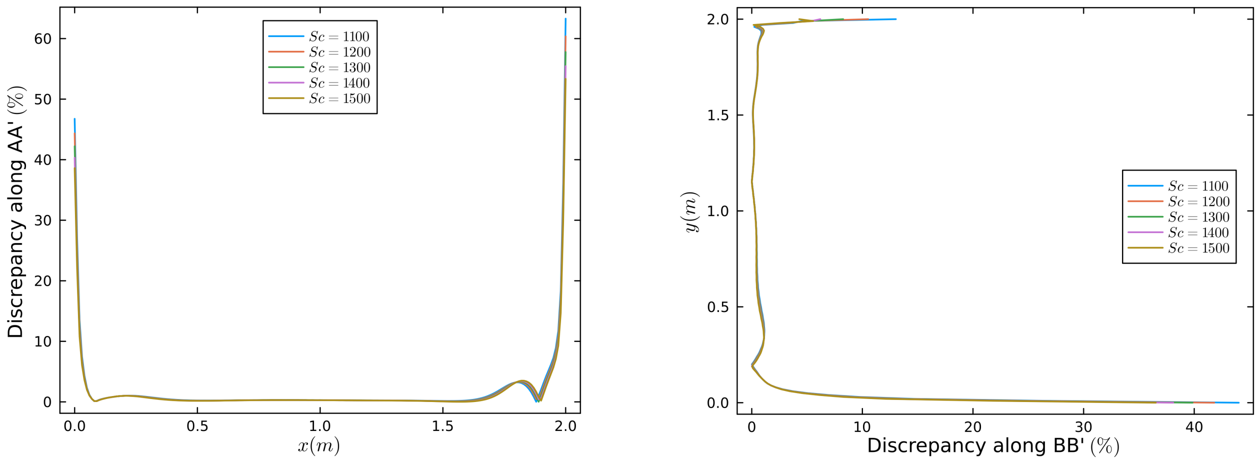

Figure 5.10: Discrepancy per evaluation point with benchmark results from step 1.1 as a function of Schmidt number. The left figure shows the discrepancy along the horizontal centerline, while the right figure shows the discrepancy along the vertical centerline. In both figures, the simulation where performed using a lattice grid of 501×501. The discrepancies are determined by comparing our simulated velocity field to the ensemble mean velocity field, calculated by averaging the results from the various benchmark codes, as explained in Section 5.2.

near the boundaries. Furthermore, the figures demonstrate that increasing the Schmidt number only slightly reduces simulation errors near the boundaries, while further from the boundaries, the solution has already converged for relatively small Schmidt values.

Figure 5.11 presents the simulated delayed neutron source along the horizontal (left) and vertical (right) centerlines for a simulation grid of 501×501 with a Schmidt number of 1500, compared to the benchmark results. The figure demonstrates good alignment between the current implementation and the benchmark data, with average discrepancies of 0.65% along the AA' centerline and 0.48% along BB'.

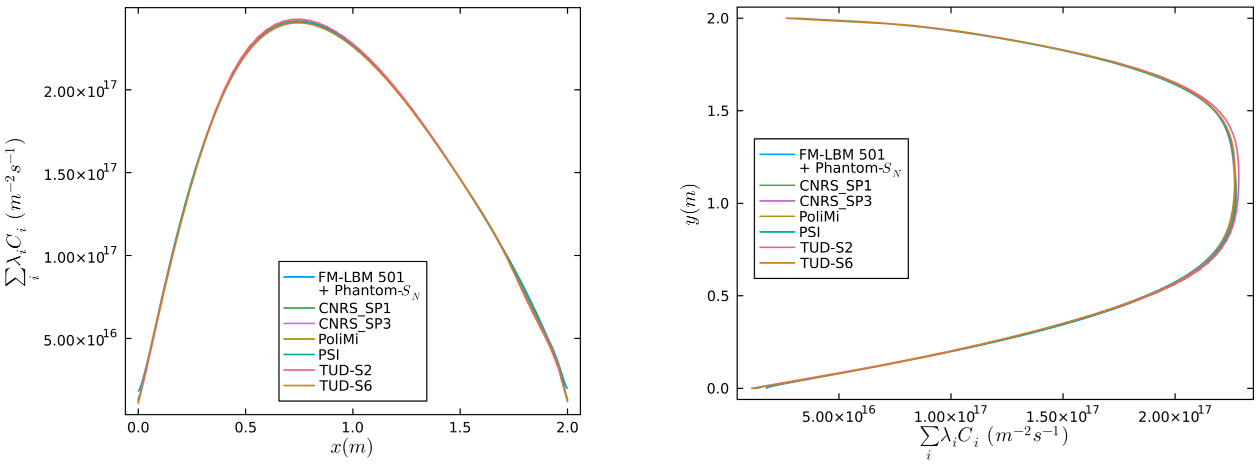

Figure 5.11: Simulated delayed neutron source along the horizontal (left) and vertical (right) centerlines in step 1.1 of the Tiberga benchmark case for a simulation grid of 501×501 with a Schmidt number of 1500, compared to the benchmark results.

Finally, the impact of precursor transport on reactivity is evaluated. In the benchmark study, this is done by calculating the reactivity difference between the current simulation and the simulation result from step 0.2, in which where the flux shape is computed independently of the flow field. Table 5.9 presents the effective multiplication factor, reactivity, and reactivity change relative to step 0.2, as calculated in this study, alongside benchmark values for comparison. The results demonstrate strong agreement with the benchmark, particularly in the reactivity change values relative to step 0.2. This agreement arises because the reactivity change metric inherently accounts for differential handling in the neutronics calculations. As previously discussed, the TUD code is the only simulation tool in this benchmark that solves the full NTE without relying on the diffusion approximation. Consequently, if diffusion-based codes introduce an error in the simulated reactivity compared to transport-based codes, this error is effectively canceled when calculating reactivity differences with respect to step 0.2. This error cancellation explains the greater consistency observed in the benchmark results for the reactivity change, $\rho_{1.1} - \rho_{0.2}$, compared to the absolute reactivity values of this simulation step. Furthermore, it also explains why our findings show the closest alignment with results from the TU-S_6 simulation code. This is due to the identical handling of the neutronics calculations, in which the Phantom-S_N algorithm solves the full NTE, without relying on the diffusion approximation, as the other simulation tools do.

Table 5.9: Effective multiplication factor, reactivity (in per cent mille), and reactivity change relative to step 0.2, as calculated in this study for step 1.1 of the Tiberga benchmark case, shown alongside benchmark values for comparison.

Code	k_{eff}	ρ (pcm)	$\rho_{1.1} - \rho_{0.2}$ (pcm)
FM-LBM 501 + Phantom-S_N	1.00516126	513.5	-64.7
CNRS-SP_1	1.00350021	348.8	-62.5
CNRS-SP_3	1.00291950	291.1	-62.6
PoliMi	1.00360495	359.2	-62.0
PSI	1.00349920	348.7	-63.0
TUD-S_2	1.00422377	420.6	-62.0
TUD-S_6	1.00520091	517.4	-60.7

5.7. Step 1.2: Power coupling

In step 1.2 of the Tiberga benchmark case, the coupling between thermal hydraulics and neutronics calculations is further enhanced. In addition to precursor transport, power coupling is now achieved. This means that the power output from the neutronics calculations is used as a source term in the heat equation, while the temperature field is used to adjust the nuclear cross-sections according to Equation 3.43. Consequently, this step achieves full two-way coupling between our FM-LBM algorithm and the Phantom-S_N code for both power and precursor flow. For simplicity, buoyancy effects are disregarded, and the imposed velocity field from step 0.2 is maintained in this step, isolating any potential errors in the simulation results to the power coupling. The primary objective of this step is to evaluate the reactivity change resulting from the combined coupling of the temperature field and neutron flux, in addition to the precursor coupling. The parameters relevant to this simulation step can be compiled from the parameters of the previous steps provided in Tables 5.4, 5.5, 5.7, and 5.8.

From this point on, given the findings in Figures 5.6 and 5.9, it is assumed that simulations converge for grid sizes of 501×501, a Prandtl number of 1000, and a Schmidt number of 1500 in the current benchmark setup. Consequently, in the remaining steps these values will be used, and no analysis will be performed on grid dependence and the convergence of the simulation results with respect to the Prandtl and Schmidt number.

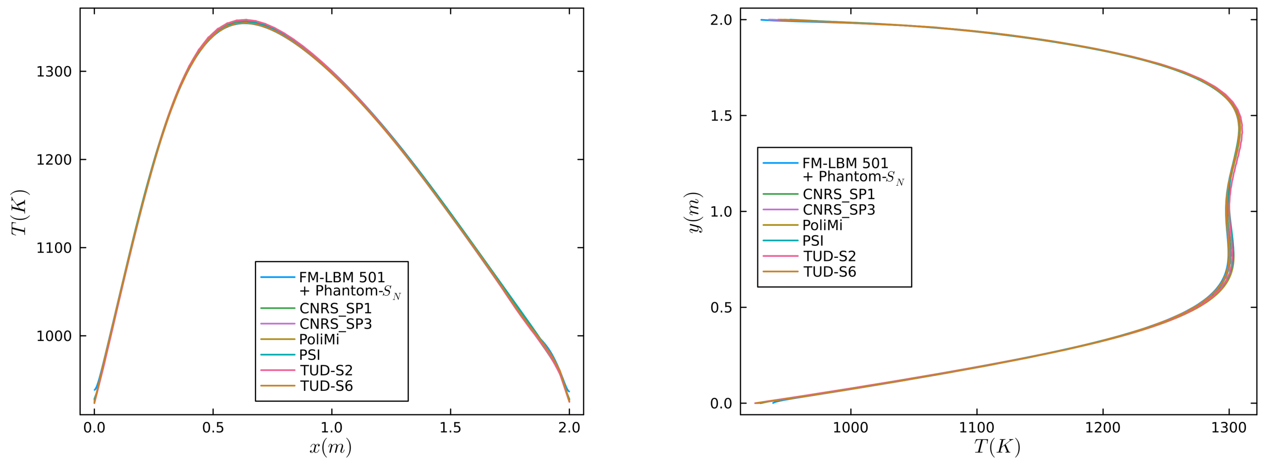

Figure 5.12: Simulated temperature field along the horizontal (left) and vertical (right) centerlines in step 1.2 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

The observables that will be compared to the benchmark results in this step are the temperature field, shown in Figure 5.12, and the differential fission rate density between the current simulation of the neutron flux and the results from step 0.2, shown in Figure 5.13. These results align well with the benchmark studies, showing

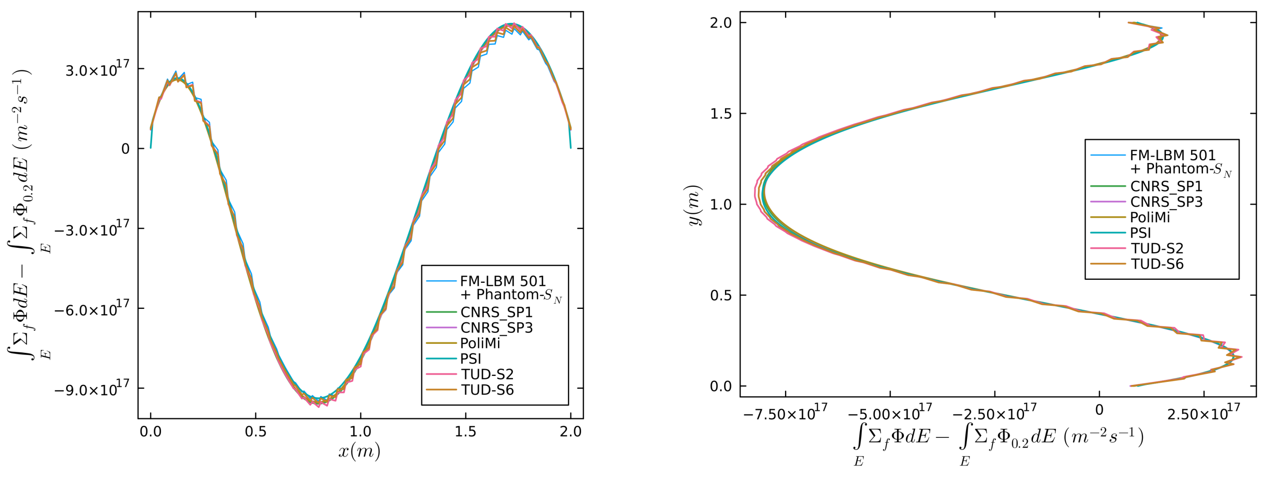

Figure 5.13: Differential fission rate density with the simulation result of step 0.2 along the horizontal (left) and vertical (right) center-lines in step 1.2 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

average discrepancies of 0.18% along AA' and 0.19% along BB' for the temperature field, and 3.49% along AA' and 1.23% along BB' for the differential fission rate density relative to step 0.2.

An important observation from Figure 5.13 is the ridged shape of the differential fission rate density. This pattern arises from the Phantom-S_N algorithm, which adjusts cross-sections based on average temperature values per element, unlike other codes that use a point-wise approach. As a result, this pattern is also seen in the TU Delft results, as these results also make use of the Phantom-S_N algorithm. The reason why this ridged shape is so clearly visible is because the differences in neutron flux shape between the current simulation and step 0.2 are relatively small, effectively providing a close-up view of the results from the Phantom-S_N algorithm. In contrast, the neutron flux shapes themselves are two orders of magnitude larger, forming an overall smooth curve. This ridged shape also accounts for the relatively high discrepancy observed in the differential fission rate density compared to previously reported discrepancies. This is consistent with findings from the benchmark study, which also reports larger discrepancies among different benchmark codes for the differential fission rate density.

Similar to step 1.1, Table 5.10 presents the reactivity change resulting from the combined effects of precursor and power coupling on the simulation results. It includes the effective multiplication factor, reactivity, and reactivity change relative to step 1.1, as calculated in this study, alongside the benchmark values for comparison. The simulated results show strong agreement with the benchmark, particularly for the reactivity change. As explained in the previous section, the greater consistency between benchmark results and our results for the reactivity change compared to the absolute reactivity, is due to the fact that this metric corrects for the differences in neutronics simulation. Where we acknowledge that our simulation tool and the TUD simulation code solve the full NTE, whereas the other simulation tools rely on the diffusion approximation. This also explain why our results most closely align with those of the TUD-S_6, given the identical treatment of the neutronics calculations.

Table 5.10: Effective multiplication factor, reactivity (in per cent mille), and reactivity change relative to step 1.1, as calculated in this study for step 1.2 of the Tiberga benchmark case, shown alongside benchmark values for comparison.

Code	k_{eff}	ρ (pcm)	$\rho_{1.2} - \rho_{1.1}$ (pcm)
FM-LBM 501 + Phantom-S_N	0.9940334	-600.2	-1113.7
CNRS-SP_1	0.9920320	-803.2	-1152.0
CNRS-SP_3	0.9914576	-861.6	-1152.7
PoliMi	0.9920458	-801.8	-1161.0
PSI	0.9920035	-806.1	-1154.8
TUD-S_2	0.9928061	-724.6	-1145.2
TUD-S_6	0.9939903	-604.6	-1122.0

5.8. Step 1.3: Buoyancy

In step 1.3 of the Tiberga benchmark case, the fully coupled system, now including buoyancy effects, is examined under the simplest conditions, specifically with $U_{lid} = 0.0$. This means that the imposed velocity field from step 0.1 is removed, and all fields are now solved in the simulation, subject to all coupling processes. Since the full coupling between thermal hydraulics and neutronics has already been validated in step 1.2, any discrepancies in the simulation results compared to the benchmark codes at this stage can be attributed to the inclusion of buoyancy effects in the thermal hydraulics code. This step represents the first opportunity to evaluate the performance of the complete multiphysics tool developed in this research. The parameters relevant to this simulation step are compiled by combining those provided in Tables 5.4, 5.5, 5.7, and 5.8, along with the parameters specifically associated with buoyancy effects given in Table 5.11.

Table 5.11: Simulation parameter used for the computation of step 1.3 of the Tiberga benchmark case, which involves the simulations using the full coupled system also including buoyancy effects. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values. These parameters should be complemented with the parameters given in Tables 5.4, 5.5, 5.7, and 5.8 to obtain the full list of parameters used in this simulation step.

Parameter	Physical value	Physical unit	LBM value	LBM unit
β_{th}	$2.0 \cdot 10^{-4}$	K^{-1}	$1.80 \cdot 10^{-1}$	lT^{-1}
g	9.81	m s^{-2}	$3.91 \cdot 10^{-4}$	ls lt^{-2}

Figures 5.14, 5.15, 5.16, and 5.17 present the simulation results for the horizontal and vertical components of the velocity field, the temperature field, and the delayed neutron source along the horizontal and vertical centerlines, respectively. These results correspond to a simulation grid of 501×501, with a Prandtl number of 1000 and a Schmidt number of 1500. The simulation shows strong agreement with the benchmark results with average discrepancies of 0.51% along AA' for the horizontal velocity component, 0.76% along AA' and 0.17% along BB' for the vertical velocity component, 0.14% along AA' and 0.20% along BB' for the temperature field, and 0.90% along AA' and 1.01% along BB' for the delayed neutron source. While the discrepancies for the delayed neutron source appear relatively large compared to the others reported, this level of discrepancy is also observed in the benchmark study between the different benchmark codes. This can also be observed from the results presented in Figure 5.17, where the differences in the delayed neutron source results across the various benchmark codes are more clearly visible compared to other observables presented in this simulation step.

Finally, analogous to steps 1.1 and 1.2 of the Tiberga benchmark case, Table 5.12 presents the simulated effective multiplication factor, reactivity, and reactivity change relative to step 0.2, alongside the benchmark values. Just as in the previous steps, we observe particularly good alignment with the benchmark results for the reactivity change, as this observable accounts for the differential handling of the neutronics calculations (as explained in section 5.6). For the reactivity itself, the results align most closely with those of the TUD-S_6 benchmark code, as this is the only code that also considers neutron transport calculations, whereas the other benchmark codes rely on the neutron diffusion approximation.

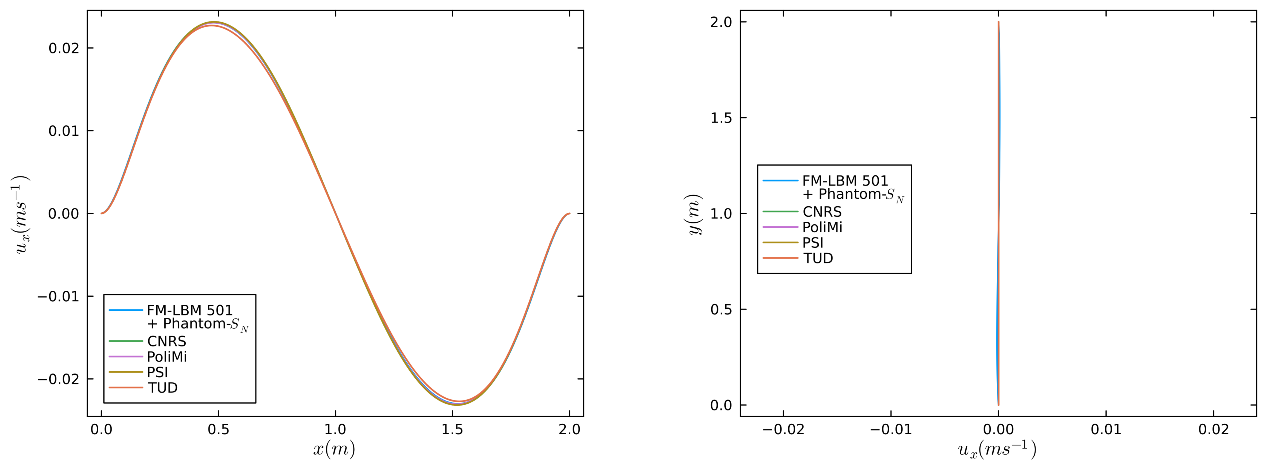

Figure 5.14: Simulated horizontal velocity component along the horizontal (left) and vertical (right) centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

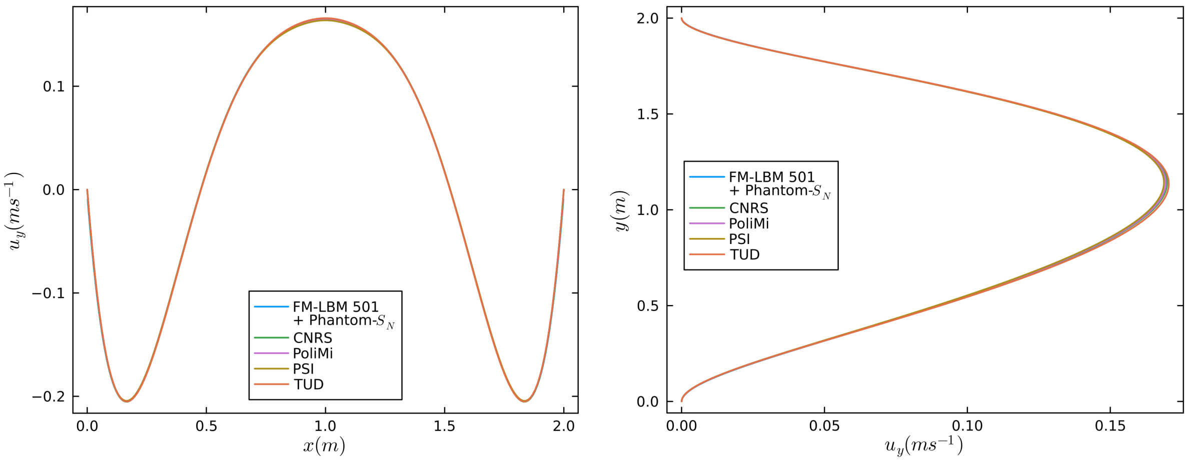

Figure 5.15: Simulated vertical velocity component along the horizontal (left) and vertical (right) centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

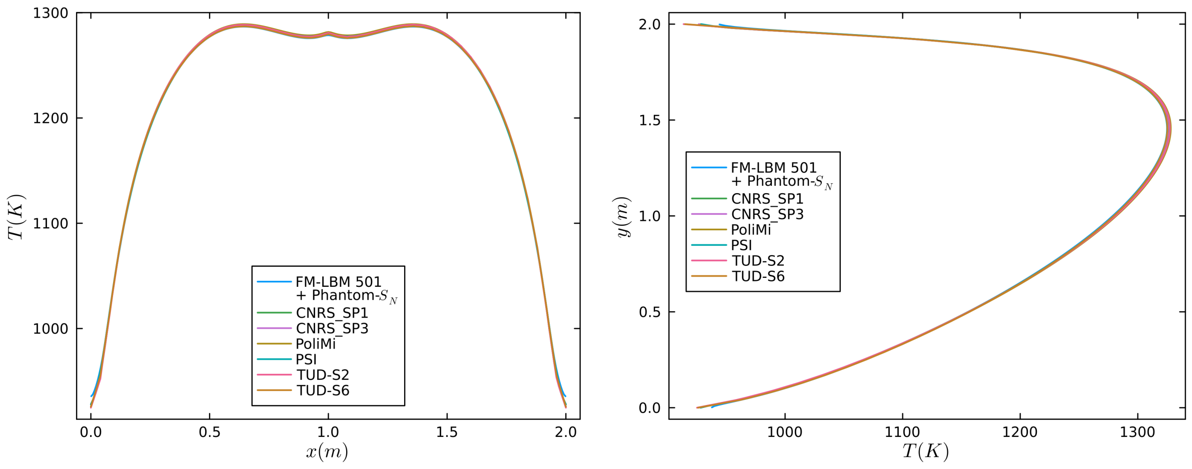

Figure 5.16: Simulated temperature field along the horizontal (left) and vertical (right) centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

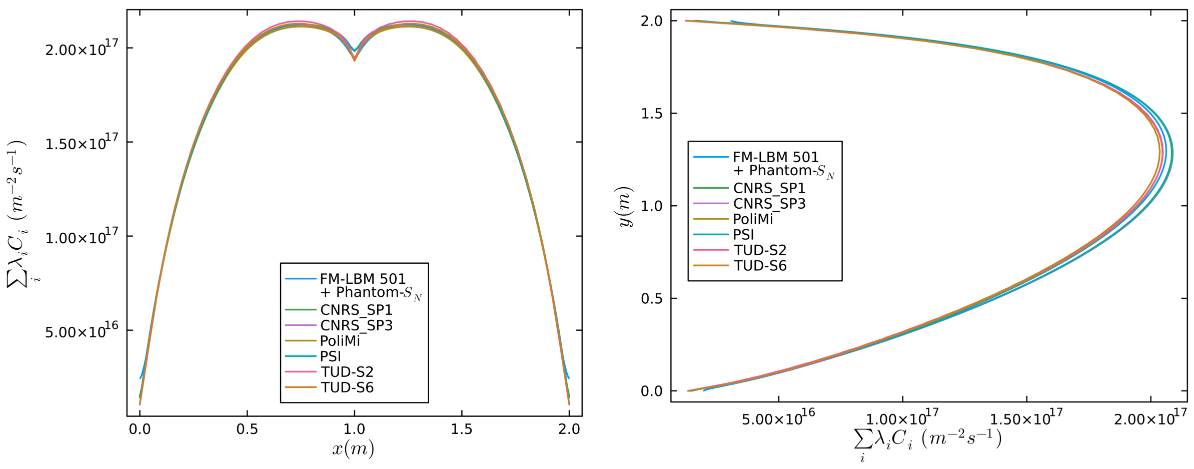

Figure 5.17: Simulated delayed neutron source along the horizontal (left) and vertical (right) centerlines in step 1.3 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500, compared to the benchmark results.

Table 5.12: Effective multiplication factor, reactivity (in per cent mille), and reactivity change relative to step 0.2, as calculated in this study for step 1.3 of the Tiberga benchmark case, shown alongside benchmark values for comparison.

Code	k_{eff}	ρ (pcm)	$\rho_{1.3} - \rho_{0.2}$ (pcm)
FM-LBM 501 + Phantom-S_N	0.9940039	-603.2	-1181.4
CNRS-SP_1	0.9919730	-809.2	-1220.5
CNRS-SP_3	0.9914045	-867.0	-1220.7
PoliMi	0.9920064	-805.8	-1227.0
PSI	0.9919858	-807.9	-1219.6
TUD-S_2	0.9927933	-725.9	-1208.5
TUD-S_6	0.9939735	-606.3	-1184.4

5.9. Step 1.4: Full coupling

Step 1.4 is the final step of the Tiberga benchmark study that focusses on steady-state simulations. In this step, the same setup as in the previous step is used, where all fields are simulated with all coupling processes applied. However, this time the simulation is performed for different values of U_{lid} (> 0.0) and P_{ref}. Rather than comparing the various observables fields along the centerlines AA' and BB' for all combinations of U_{lid} and P_{ref}, the benchmark study only reports the reactivity changes compared to the simulation results from step 0.2 for the different benchmark codes. In total, 30 different combinations of U_{lid} and P_{ref} are considered in the benchmark study.

In this research, a selection of 4 combinations of U_{lid} and P_{ref} are chosen from the 30 reported in the benchmark study, corresponding to $U_{lid} \in \{0.3 \text{ ms}^{-1}, 0.5 \text{ ms}^{-1}\}$, and $P_{ref} \in \{0.6 \text{ GW}, 1.0 \text{ GW}\}$. The simulated reactivity and reactivity changes for these combinations are presented in Table 5.13, alongside the values reported in the benchmark study. Additionally, the observables along the vertical and horizontal centerlines for the simulation with $U_{lid} = 0.5 \text{ ms}^{-1}$ and $P_{ref} = 1.0$ GW are provided in Appendix B for completeness.

Table 5.13: Reactivity (in per cent mille) and reactivity change relative to step 0.2 for different values of U_{lid} and P_{ref}, as calculated in step 1.4 of the Tiberga benchmark case, which considers the fully coupled multiphysics system. The results are presented alongside the values reported in the benchmark study for comparison.

Code		P_{ref} (GW)			
	U_{lid} (ms^{-1})	0.6	1.0	0.6	1.0
		ρ (pcm)		$\rho_{1.4} - \rho_{0.2}$ (pcm)	
FM-LBM 501 + Phantom-S_N		-134.6	-595.0	-712.7	-1173.1
CNRS-SP_1		-323.9	-800.8	-735.2	-1212.1
CNRS-SP_3		-381.6	-858.7	-735.3	-1212.4
PoliMi		-312.8	-797.8	-734.0	-1219.0
PSI	0.3	-323.9	-795.1	-735.6	-1206.8
TUD-S_2		-245.1	-717.6	-727.7	-1200.2
TUD-S_6		-135.1	-598.3	-713.2	-1176.4
FM-LBM 501 + Phantom-S_N		-130.8	-586.7	-708.9	-1164.8
CNRS-SP_1		-321.6	-793.5	-732.9	-1204.8
CNRS-SP_3		-379.3	-851.5	-733.0	-1205.2
PoliMi		-315.8	-792.8	-737.0	-1214.0
PSI	0.5	-321.4	-788.1	-733.1	-1199.8
TUD-S_2		-242.6	-710.4	-725.2	-1193.0
TUD-S_6		-132.7	-591.6	-710.8	-1169.7

Table 5.13 demonstrates strong alignment between our simulation results and the benchmark data. It shows that the quality of our simulation results is consistent across the different values of U_{lid} and P_{ref}. Just as in previous steps, we observe that our simulated reactivity aligns most closely with the TUD-S_6 results, as this is the only benchmark code that accounts for neutron transport effects. Furthermore, the reactivity change relative to step 0.2, as calculated in this research, shows closer alignment with all benchmark studies, not just the TUD-S_6 results. As explained in section 5.6, this is because the reactivity change controls for the differential handling of the neutronics simulation across the various benchmark codes.

Step 1.4 of the Tiberga benchmark case completes the validation of the full multiphysics simulation tool under steady-state conditions. Overall, our code demonstrates strong alignment with the simulation results from the benchmark studies. In all benchmark steps, the observables compared along the horizontal and vertical centerlines show average discrepancies of less than one percent compared to the benchmark studies, except for the differential fission rate density shown in Figure 5.13. As explained, this discrepancy is due to the mesh-based approach used in Phantom-S_N for handling temperature feedback effects on the nuclear cross-sections. Finally, the simulated reactivities in each benchmark step consistently aligned closely with the TUD-S_6 results, given that this simulation tool also solves the complete NTE without relying on the diffusion approximaion. Furthermore, the reactivity changes with respect to the independent calculation of the neutron flux in step 0.2, showed great consistency across all benchmark results. As explained in section 5.6, this agreement in reactivity changes between all benchmark results, including those solving the neutron diffusion equations, are due to this metric effectively accounting for the differential handling of the neutronics calculations between the codes. Based on these findings, we can conclude that the multiphysics simulation tool developed in this research yields satisfactory results under steady-state conditions.

5.10. Convergence Speed

All steady-state simulations from steps 1.1 to 1.4 used the power method to solve the complete system. Within this framework, the Phantom-S_N algorithm is used to solve the neutron flux, while the FM-LBM algorithm is responsible for solving the remaining fields. This setup involves a nested loop structure, with both loops requiring convergence criteria. The power method, serving as the outer loop, converges when the relative changes in the solution vector $\gamma^{(l)}$ and the effective multiplication factor $k_{\textit{eff}}^{(l)}$ satisfies the convergence criteria specified in Equation 5.2. The FM-LBM algorithm operates as the inner loop, advancing the velocity, temperature, and precursor fields over several LBM time steps until these fields reach convergence, which is based on their relative changes, as defined in Equation 5.3.

To optimize computational efficiency, an appropriate combination of thresholds for these convergence criteria must be selected. The power method's convergence criteria ultimately determine the accuracy of the simulation results and must therefore be set to ensure the desired accuracy. However, the convergence criterion of the FM-LBM algorithm also plays a critical role, as this criterion directly influence the number of iterations required for the power method to achieve convergence. Setting a strict convergence threshold for the FM-LBM algorithm ensures that the velocity, temperature, and precursor fields reach a steady state solution within each iteration of the power method. This minimizes the number of power method iterations required to reach full convergence of the system. Conversely, relaxing the FM-LBM convergence criterion introduces small errors due to partially converged fields, requiring additional power method iterations to compensate. This is because the power method relies on the previous iteration's partially converged solution vector. In such cases, the FM-LBM algorithm is invoked more times, allowing it to advance the fields toward their true steady-state solutions.

The main point to be made here, is that within these steady-state simulations, the FM-LBM algorithm proved to be the most time consuming to reach steady-state. Particularly during the first few iterations of the power method, the FM-LBM algorithm required significantly more time to converge to a threshold of 10^{-6} than a threshold of 10^{-4}. To address this, a trade-off can be made by relaxing the FM-LBM convergence criteria. A relaxed threshold results in fewer LBM time steps per power method iteration, thereby reducing the computation time of the FM-LBM algorithm for each power iteration. However, this comes at the expense of introducing additional power method iterations to achieve overall steady-state convergence. The challenge lies in selecting an FM-LBM convergence criterion that balances the computational time of the FM-LBM algorithm per power iteration with the number of additional power iterations required, ultimately minimizing the total simulation time for the entire multiphysics tool.

In our simulations, a threshold of 10^{-4} for the FM-LBM algorithm was found to be the most effective choice. This threshold was determined through trial and error, where different configurations were tested, and the one providing the most acceptable simulation time was selected. It is important to note, however, that this threshold may not always represent the optimal value. Multiple other simulation parameters also affect this choice, such as the LBM parameters that influences the convergence time of the FM-LBM algorithm. With that said, the LBM parameters should also be chosen most favorably, such that, for a given grid size, the FM-LBM algorithm also reaches steady-state solutions in the shortest time possible, while maintaining both numerical stability and accuracy.

6 Validation of Transient Multiphysics Coupled Simulation

Having confirmed the successful coupling between the FM-LBM algorithm for thermal hydraulics and the Phantom-S_N algorithm for neutronics, as well as the capability of our multiphysics simulation tool to produce accurate steady-state solutions, the next step is to validate its performance under transient conditions. This validation step builds on the results from the previous chapter, as we continue benchmarking our simulation results against those presented in the Tiberga benchmark study [96].

Section 6.1 outlines the benchmark setup from the Tiberga study in the case of transient simulations. Section 6.2 presents our simulation results and compares them with the benchmark data. Section 6.3 analyzes the computational performance of the overall multiphysics system and its components.

6.1. Benchmark Setup

In the last step of the Tiberga benchmark case, the transient behavior of the simulation tools are tested. This step essentially builds upon the steady-state simulation results of step 1.4, which considers the full coupled system with $U_{lid} = 0.5$ ms^{-1} and $P_{ref} = 1.0$ GW, whose results can be found in Appendix B. This means the complete setup described in Section 5.1 for the steady state simulations stays the same. The problem is then made transient by imposing a perturbation in the frequency domain, specifically by oscillating the heat transfer coefficient through time. This means the new time dependent heat transfer coefficient is given by

$$\gamma(t) = \gamma_0 \left[1 + 0.1 \sin\left(2\pi f_{pert} t\right)\right] , \tag{6.1}$$

where γ_0 denotes the reference heat transfer coefficient equal to the heat transfer coefficient used in the steady state simulations, and f_{pert} denotes the perturbation frequency. The perturbation in the frequency domain basically means that the transient simulation is initialized with the steady state solution calculated in step 1.4 of the benchmark study, and from that point immediately starts with the oscillation of the heat transfer coefficient through time.

In response to the time dependent variation of the heat transfer coefficient we expect an oscillating motion in the power output of the simulation due to the negative density feedback coefficient. To measure whether the system response is consistent across the different simulation tools, the benchmark study reports the normalized power gain and phase-shift between the heat transfer coefficient and the power output, where the normalized power gain is defined as

$$\text{Gain} = \frac{(P_{max} - P_{avg})/P_{avg}}{(\gamma_{max} - \gamma_{avg})/\gamma_{avg}} . \tag{6.2}$$

Here, the denominator will correspond to the amplitude of $\gamma(t)$, P_{avg} will equal the initial reference power of 1 GW, and P_{max} is the maximum power output measured during the transient simulation. Furthermore, the phase-shift is calculated by comparing the times at which different maxima and minima occur between heat transfer coefficient and the power output. Specifically, the phase-shift is calculated using

$$\text{Phase-shift} = 2\pi \cdot \frac{f}{2N} \left\{ \sum_{i=1}^{N} \left[t^i_{\gamma_{\max}} - t^i_{P_{\max}}\right] + \sum_{i=1}^{N} \left[t^i_{\gamma_{\min}} - t^i_{P_{\min}}\right] \right\} . \tag{6.3}$$

Here, a total of N maxima and minima are compared, which excludes the first two maxima and minima. This exclusion is intentional to ensure the system has reached a quasi-equilibrium state, where the oscillatory motion remains consistent across different oscillations. It is important to note that this omission of the first two maxima is also applied when determining P_{max} for the calculation of the power gain.

In the benchmark, the phase-shift and the power gain are calculated for a total of 7 different perturbation frequencies corresponding to $f_{pert} \in \{0.0125,\ 0.025,\ 0.05,\ 0.1,\ 0.2,\ 0.4,\ 0.8\}$. Following the setup used in the TUD-S_6 code from the benchmark study, the timesteps for the Phantom-S_N algorithm are set to $1/200$ of the perturbation period associated with each frequency. Additionally, the number of FM-LBM timesteps is determined by using the time unit conversion factor to ensure that the total physical time simulated by the FM-LBM algorithm matches the timestep of the Phantom-S_N algorithm.

A remark must be made regarding the determination of the FM-LBM timesteps. Since the transient simulations begin from the steady-state results of step 1.4, they are inherently constrained by a specific time conversion factor determined by the LBM simulation parameters used in step 1.4. This constraint has two key implications. First, as the perturbation frequency increases, the Phantom-S_N timesteps decrease, which in turn reduces the number of FM-LBM timesteps per Phantom-S_N timestep. One way to address this issue is to perform multiple steady-state simulations with LBM parameters chosen such that, in transient simulations, the number of FM-LBM timesteps per Phantom-S_N timestep remains constant across all perturbation frequencies. However, this approach was not adopted in this work for two reasons. Firstly, the transient simulations already demonstrated satisfactory results, making it unnecessary to generate steady-state results with different time conversion factors that would lead to more FM-LBM timesteps per Phantom-S_N. Secondly, steady-state simulations with the required parameters would take an unfeasibly long time to converge.

The second implication is that for higher perturbation frequencies, the number of FM-LBM timesteps corresponding to one Phantom-S_N timestep not always resulted an integer number. Since the FM-LBM algorithm cannot simulate a fractional number of timesteps, this posed a challenge. This issue was resolved by conducting additional steady-state simulations with slight variations in the LBM viscosity parameter. These adjustments where chosen in such a way, that the time conversion factor ensured the number of FM-LBM timesteps per Phantom-S_N timestep in transient simulations was an integer number. For completeness, Table 6.1 provides an overview of the number of FM-LBM timesteps per Phantom-S_N timestep for the different perturbation frequencies.

Table 6.1: Overview of the number of FM-LBM timesteps per Phantom-S_N timestep for various perturbation frequencies. When the time conversion factor from the steady-state simulations led to a non-integer number of FM-LBM timesteps, additional steady-state simulations were conducted with slightly adjusted LBM viscosity values. These adjustments produced a time conversion factor that ensured an integer number of FM-LBM timesteps in the transient simulations.

Perturbation frequency (s^{-1})	Phantom-S_N timestep (s)	Number of FM-LBM timesteps (lt)
0.0125	0.4	1002
0.025	0.2	501
0.05	0.1	250
0.1	0.05	125
0.2	0.025	63
0.4	0.0125	31
0.8	0.00625	16

6.2. Benchmark Results

Before comparing our simulation results to the benchmark data, Figure 6.1 first gives a snapshot of the oscillating motion of the heat transfer coefficient and the power output in the transient simulation for two perturbation frequencies 0.0125 and 0.2 by displaying the normalized deviations of $\gamma(t)$ and $P(t)$, defined as

$$\tilde{\gamma}(t) = \frac{\gamma(t) - \gamma_{\text{avg}}}{\gamma_{\text{avg}}}, \qquad \tilde{P}(t) = \frac{P(t) - P_{\text{avg}}}{P_{\text{avg}}}. \tag{6.4}$$

Here, γ_{avg} is equal to γ_0 and P_{avg} is equal to the reference power of 1 GW. These figures clearly illustrate how the power output oscillates as a result of the perturbation in the heat transfer coefficient. Furthermore, we see how the different frequencies of the perturbation affect this power response, where the phase-shift becomes larger and the gain becomes smaller for higher perturbation frequencies, which is a result of the response time of the system being to large to capture the rapid variations in the salt cooling. Snapshots of the transient simulations from the other perturbation frequencies can be found in Appendix C.

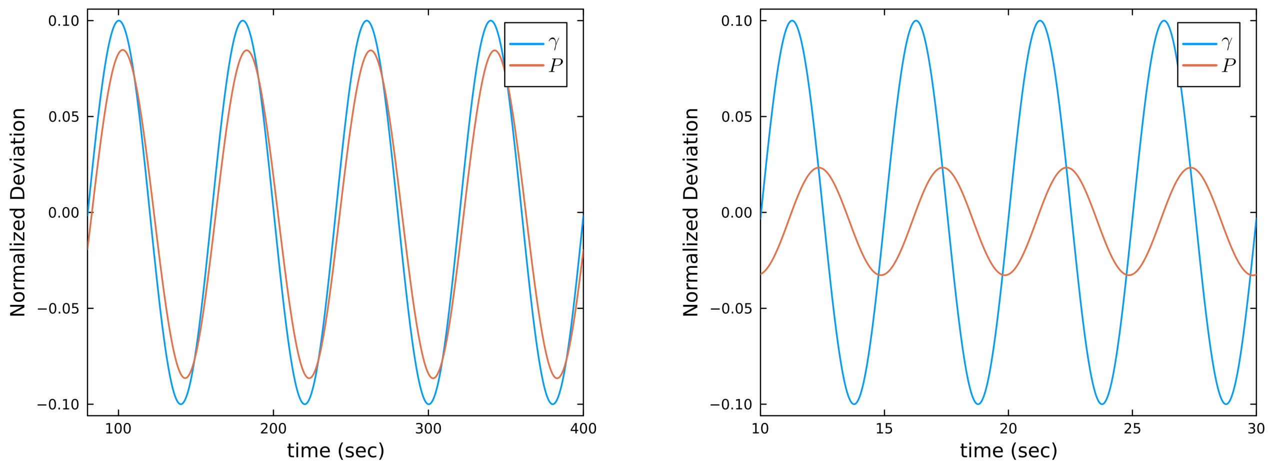

Figure 6.1: Snapshots of the normalized deviations of the heat transfer coefficient and the resulting power output in the transient simulations for two different frequencies $f_{pert} = 0.0125$ (left) and $f_{pert} = 0.2$ (right). These figure illustrate the system response of the output power to the perturbation on the heat transfer coefficient in the frequency domain.

To compare the response of the system to the perturbation in the heat transfer coefficient to the benchmark codes, Figure 6.2 shows the phase shift and the normalized power gain corresponding to the different perturbation frequencies in Bode plots. The left figure shows that there is not a lot of deviation in the calculated power gain between the different benchmark codes, and that the power gain calculated using our simulation tool shows good alignment with these values. The right figure, however, illustrates that the corresponding values for the phase-shift is somewhat scattered between the different benchmark codes. In the benchmark study the average discrepancies between the benchmark codes for the phase-shifts is calculated as 2.2 % which they considered acceptable. Our results follow the same general shape of the phase-shift with respect to the perturbation frequency and shows a similar order of discrepancy with the other benchmark results.

From the Bode plots in Figure 6.2 it can be concluded that the system response to a perturbation in the frequency domain correctly matches those found in the benchmark study. As a result, we can conclude that the multiphysics simulation tool developed in this research is also able to accurately simulate transient behaviour next to steady-state solutions, thereby completing the Tiberga benchmark case.

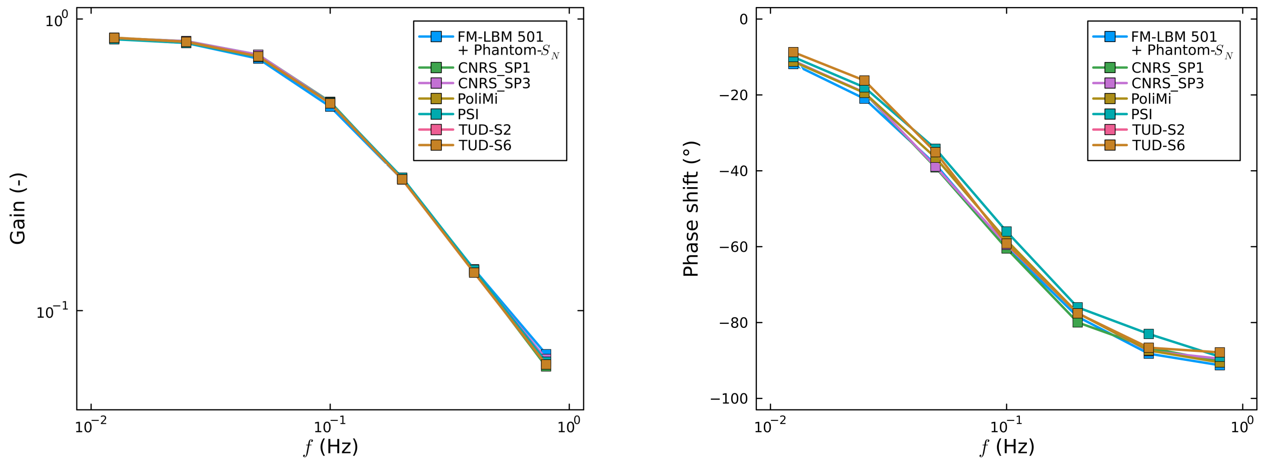

Figure 6.2: System response to a perturbation in the frequency domain on the heat transfer coefficients illustrated with the use of Bode plots. The plots show the power gain (left) and the phase-shift (rigth) between the oscillating variations of the heat transfer coefficient and the resulting power output.

6.3. Performance Analysis

Now that the multiphysics simulation tool has been successfully completed and demonstrated to provide satisfactory results under both steady-state and transient conditions, we move to evaluating its performance.

This evaluation focuses exclusively on the FM-LBM algorithm, which encompasses the thermal hydraulics and precursor transport. The performance of the Phantom-S_N code is not assessed in this research, as our contributions to it were limited to adding code to facilitate its coupling with the FM-LBM algorithm. The core functionality of Phantom-S_N, which involves solving the NTE using DG-FEM, was neither designed nor implemented by us. As a result, decisions related to its implementation and potential performance optimizations is not of interest to this research. Remarks on the performance of Phantom-S_N and possible improvements can be found in [92]. Additionally, the performance impact of the coupling between the algorithms is also not included in this assessment. It should be noted, however, that the process of exchanging variables between the thermal hydraulics and neutronics codes via plain file I/O, combined with restarting the Julia code during each iteration, is highly time-consuming and significantly affects overall performance. These inefficiencies arise from implementing the thermal hydraulics code and the neutronics code in different programming languages. The Julia language was selected for the FM-LBM code to leverage its extensive GPU functionalities, especially regarding memory management, while Phantom-S_N was already implemented in Fortran-90. These limitations could be addressed by developing a unified codebase that integrates both Phantom-S_N and FM-LBM in a single programming language. However, rewriting the Phantom-S_N code into Julia code would be an extremely time-intensive task and falls outside the scope of this project due to the substantial complexity of the existing code.

Analogous to section 4.3 the performance of the FM-LBM algorithm is addressed by evaluating the number of cell evaluations per second in terms of million lattice updates per second (MLUPS), as defined in Equation 4.4. Figure 6.3 illustrates the amount of MLUPS as a function of domain size, comparing the performance of single versus double precision floating-point numbers. Note that the x-axis in Figure 6.3 denotes the number of lattice points along a single axis, while in reality the total domain consists of N^2 lattice nodes. The figure illustrates a maximal performance of around 57 MLUPS for double precision floating-point numbers and 70 MLUPS for single precision floating-point numbers for the full FM-LBM algorithm including precursor transport.

Two observations can be made regarding Figure 6.3. First, the number of cell evaluations per second for double precision floating-point numbers does not appear to follow a smooth function with domain size. This behavior especially stands out when compared to the single precision floating-point results or the functions presented in Figure 4.4, where smooth, stagnating functions are observed for both single and double precision floating-point numbers. When comparing to the other functions, the double precision floating-point performance in the current simulation tool seem to exhibit a noticeable drop around $N = 450$ and $N = 500$. These inefficiencies can be attributed to the behavior of the warp scheduler. In the current complex setup, with multiple distribution functions, a variety of kernel functions, and different levels of parallelization across these kernel functions, it is impossible to say what the warp scheduling process actually looks like. As described in Section 2.4.5, the warp scheduler allocates full blocks at once. It is plausible that domain configurations of 450^2 and 500^2 result in suboptimal block allocation, leading to inefficiencies in the warp scheduling process and, consequently, a reduction in the number of lattice evaluations per second.

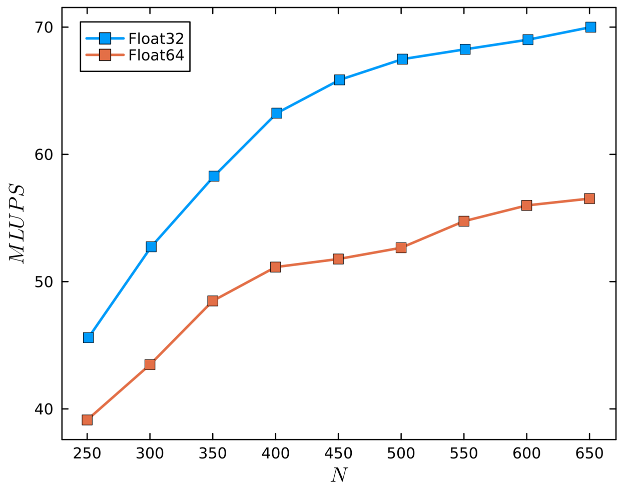

Figure 6.3: Simulation speed of the FM-LBM algorithm including precursor transport expressed in MLUPS, comparing performance between single and double precision floating-point numbers. Note that, in this context, a lattice update refers to the complete execution of all kernel functions for both the velocity, temperature, and precursor fields at a single lattice point during one LBM time iteration. Furthermore, the simulation speeds are measured for different square domain sizes, where N represents the number of lattice nodes along a single axis (e.g., $N = 200$ corresponds to a total of 200^2 lattice points).

The second notable observation concerns the performance of the current simulation results compared to the results presented in Chapter 4 for the FM-LBM algorithm applied to thermal fluid simulation without precursor transport, as shown in Figure 4.4. For both single and double precision floating-point calculations, the number of MLUPS decreases by approximately a factor of 7 compared to these results. This performance drop exceeds expectations, considering that the computational load only increases by a factor of 5 when introducing 8 precursor distribution functions in addition to the distribution functions for velocity and temperature, as now the total number of distribution functions increases from 2 to 10. Given that the number of reads and writes to both global and shared memory scales proportionally with the number of distribution functions, the theoretical upper limit, as defined in Equation 4.5, should decrease by a factor of 5. This suggests that the observed performance degradation, which is greater than 5, cannot be solely attributed to oversaturated memory bandwidth. Instead, it indicates the presence of additional performance limitations. Potential bottlenecks can include increased CPU overhead due to the need to launch a greater number of kernel functions or excessive demand for shared and register memory. The latter could lead to memory overspill, where excess memory allocations are redirected to the slower global memory rather than the faster, intended memory tiers. However, accurately identifying these additional performance constraints, beyond the finite memory bandwidth of the GPU, requires further investigation into this specific implementation of the FM-LBM algorithm and its associated performance issues.

Nevertheless, the maximum performance of 57 MLUPS for double precision floating-point numbers is still deemed satisfactory, particularly when compared to serial implementations of NTH tools using LBM. For instance, in a similar study, [80] employed a serial implementation of the LBM algorithm to simulate both thermal hydraulics with precursor transport and neutronics. In the final step of the Tiberga benchmark study, they reported a simulation time of 23 hours for a transient simulation involving 100,000 time steps on a 50×50 lattice grid using double precision floating-point numbers, corresponding to a performance of approximately 0.03 MLUPS. Even if we conservatively double this performance to account for the additional computational load of simulating neutronics fields, the GPU-accelerated LBM approach still achieves a performance gain of nearly 1000 times. This significant improvement underscores the efficiency and strength of GPU-accelerated LBM in NTH tools.

7 Advancing Towards Actual Molten Salt Fast Reactor Core Simulations

At this stage, we have developed a multiphysics simulation tool that couples thermal hydraulics simulations with neutronics simulations, and we have demonstrated that this simulation tool produces satisfactory results under both steady-state and transient conditions. In some respects, the simulation tool represents significant advancements over other solutions presented in the literature, such as the inclusion of neutron transport effects, thereby eliminating the need for the diffusion approximation in neutronics calculations. However, in other aspects, the simulation tool meets only minimal requirements for simulating the multiphysics within an MSFR reactor core. For example, we have thus far only considered laminar flow conditions and reduced problem spaces by simulating in a 2D square geometry. This chapter provides a forward-looking discussion on the necessary extensions and improvements required for the multiphysics tool to achieve accurate and comprehensive modeling of a realistic MSFR reactor core, moving beyond the simplified conditions and reduced problem spaces thus far employed.

Section 7.1 discusses the extension of the simulation tool to three-dimensional problem spaces. Furthermore, Section 7.2 provides a short background in turbulent modeling and explains how this can be incorporated into the LBM framework. Lastly, Section 7.3 explores the implementation of the MSFR core geometry and its implications for the boundary condition schemes of the LBM algorithm.

7.1. 3D Modeling

Thus far, the NTH tool has been used exclusively for simulations in two-dimensional problem spaces. However, to perform calculations in realistic MSFR geometries, it is necessary to extend these simulations to three-dimensional spaces. Fortunately, Phantom-S_N is designed to handle both 2D and 3D problem geometries, enabling three-dimensional neutronics simulations simply by implementing a three-dimensional mesh and running Phantom-S_N in "3D mode". Moreover, the FM-LBM algorithm developed in this research is also fully implemented in 3D. This means, the D3Q19 simulation scheme, as shown in 3.1, was used for all 2D calculations. However, these simulations were modeled using a single layer of lattice nodes along the additional axis. A periodic boundary condition along this axis was applied to effectively reduce the domain to two dimensions. This approach is feasible because the 2D and 3D implementations of the LBM algorithm are fundamentally the same. The only difference is that the kernel functions iterate over additional directions in 3D schemes, whereas the underlying collision and propagation computations remain unchanged. Finally, the coupling mechanism between Phantom-S_N and FM-LBM is designed to support both 2D and 3D simulations. During 2D simulations, dummy values are assigned to the third coordinate, which are replaced by actual data in 3D simulations. Furthermore, Galerkin projection and B-Spline interpolation are both implemented to handle both 2D and 3D cases. As a result, while the benchmark problems presented in this work showcase only two-dimensional results, the framework is fully prepared for three-dimensional applications.

To illustrate the capability of the NTH tool to perform simulations in three-dimensional space, we conducted steady-state simulations of a molten salt in a cubic box. In these simulations we use the same property characteristics of the fuel salt, and nuclear data as given in the Tiberga benchmark problem. Furthermore, we examine two scenarios: First, a purely buoyancy-driven flow, and second, a buoyancy-driven flow with an induced shear momentum source. In both cases, we apply no-slip boundary conditions for the velocity field, adiabatic boundary conditions for the temperature field, vacuum boundary conditions for the neutron flux, and Neumann boundary conditions for the precursor densities on all six walls. In the second scenario we introduce a fixed velocity U_{wall} directed along $\hat{y}$ at the plane $x = L$, which generates a shear momentum source in addition to the buoyancy force. Finally, just as in the the Tiberga benchmark case, we facilitate salt cooling by introducing a volumetric heat sink, and we model temperature feedback effects on the neutronics only through density feedback, meaning we ignore doppler feedback. As a consequence, the same simulation parameters as presented in Chapter 5 will be used.

7.1.1. Pure Buoyancy-Driven Flow

First we discuss the pure buoyancy-driven flow in which we do not include a momentum source at the side of the cube. Table 7.1 provides an overview of the simulation parameters used with their values presented in both physical and LBM units. Note that a few changes have been made on the values of a few of these parameters compared to the two dimensional cases presented in Chapter 5. First of all, the value of the volumetric heat transfer coefficient, γ, is reduced to $1.0 \cdot 10^5$, this is because with the added space resulting from simulation in 3D, the original value of $1.0 \cdot 10^6$ removed so much heat, that there was barely any deviation in the temperature field left. Furthermore, the number of lattice nodes along a each dimension is reduced to 101, resulting in 101^3 lattice nodes in total. This is mainly done to reduce the computational effort of the three-dimensional simulation. As a result, the almost all parameters also change their value in LBM units, given that the LBM unit conversion factors are interrelated, as described in Section 3.1.6.

Table 7.1: Simulation parameter used for the simulation of a molten salt in a cubic box in steady state conditions. Here we assume a pure buoyancy-driven flow, with the same property characteristics of the fuel salt, and nuclear data as given in the Tiberga benchmark problem. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values.

Parameter	Physical value	Physical unit	LBM value	LBM unit
L	2.0	m	101	ls
ρ	$2.0 \cdot 10^3$	kg m^{-3}	1.0	lm ls^{-3}
g	9.81	m s^{-2}	$1.94 \cdot 10^{-3}$	ls lt^{-2}
ν	$2.5 \cdot 10^{-2}$	m^{2}s^{-1}	$1.26 \cdot 10^{-1}$	ls^2 lt^{-1}
α	$2.5 \cdot 10^{-5}$	m^2 s^{-1}	$1.26 \cdot 10^{-4}$	ls^2 lt^{-1}
D	$1.67 \cdot 10^{-5}$	m^2 s^{-1}	$8.42 \cdot 10^{-5}$	ls^2 lt^{-1}
β_{th}	$2.0 \cdot 10^{-4}$	K^{-1}	$1.80 \cdot 10^{-1}$	lT^{-1}
γ	$1.0 \cdot 10^5$	W m^{-3}K^{-1}	$1.76 \cdot 10^{-2}$	lm ls^{-2} lt^{-3} lT^{-1}
C_p	$6.15 \cdot 10^6$	J m^{-3}K^{-1}	$5.48 \cdot 10^2$	lm ls^{-2} lt^{-2} lT^{-1}
T_{ref}	900	K	1.0	lT
P_{ref}	$1.0 \cdot 10^9$	W	$2.53 \cdot 10^4$	lm ls^2 lt^{-3}
Pr	1000	-	1000	-
Sc	1500	-	1500	-

Figure 7.1 presents the simulation results for the temperature field and the delayed neutron source using heat maps across five equally spaced horizontal cross-sections of the cubic box. Additionally, Figure 7.2 illustrates the resulting velocity field using a three-dimensional stream plot. The results demonstrate that the highest heat and precursor generation occurs at the center of the domain. This is expected, given that the neutron density is highest in this region. This heat production induces an upward motion of the fuel salt at the center of the domain due to buoyancy forces, as can be seen from the velocity field. By comparing the heat maps at $L = 0.5$ m and $L = 1.5$ m, we observe how this upward motion drives convection, transporting heat from the middle plane upwards. Although less pronounced, a similar convective effect is also evident in the delayed neutron source field, given that the precursors are also carried along with the fluid motion.

The simulation results give an effective multiplication factor of $k_{eff} = 0.9610244$, resulting in a reactivity of $\rho = -4055.6$. This reactivity which is lower compared to the situation in which we simulate the neutron flux independent of all other fields, where we observed a reactivity of $\rho_0 = -3967.1$. Finally, we observe that all simulated fields are fully symmetric with respect to the $x = y$ and $x = -y$ planes. This symmetry arises as there is no physical processes or force introduced that preferentially affects the molten salt in either of x or y directions over the other.

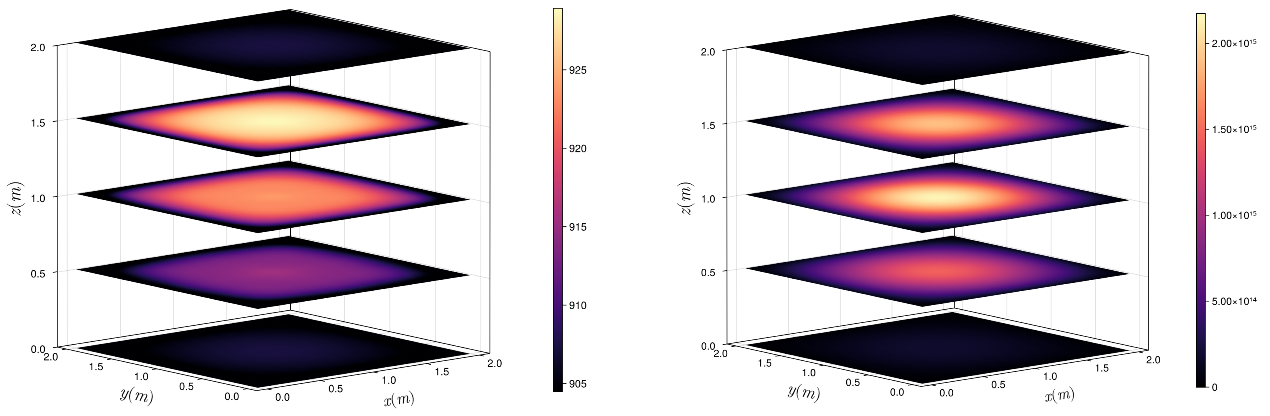

Figure 7.1: Temperature field (left) and delayed neutron source (right) from three dimensional steady-state simulation of a pure buoyancy-driven molten salt. This figure illustrates the scalar variables using heat maps across five equally spaced horizontal cross-sections of the cubic domain. The colorbars indicate the magnitude of the scalar fields in K (left) and m^{-3}s^{-1} (right).

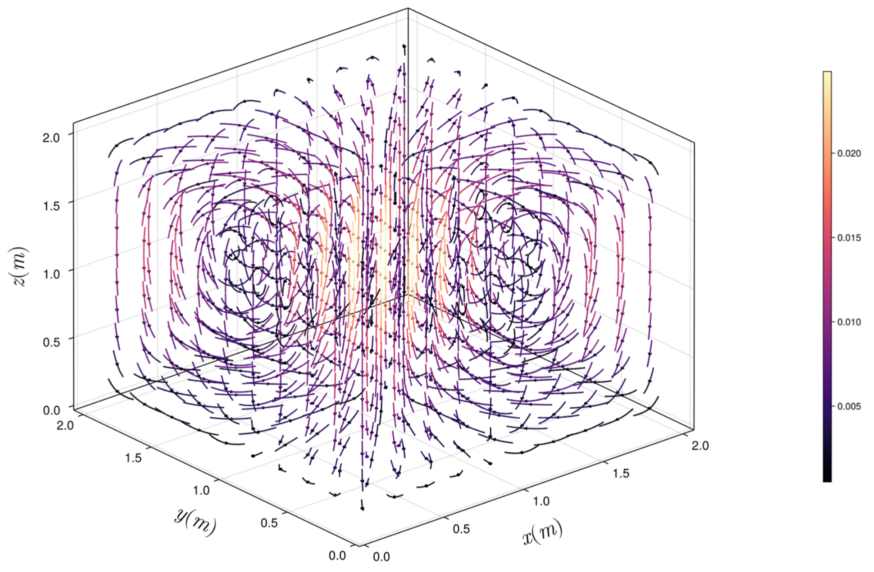

Figure 7.2: Velocity field from the three dimensional steady-state simulation of a pure buoyancy-driven molten salt. This figure illustrates the direction of the velocity field using a three-dimensional streamplot. Furthermore, magnitude of the velocity field is illustrated using the colors and the corresponding colorbar indicating the velocity in ms^{-1}.

Finally, next to the heat maps presented in Figure 7.1 and the three-dimensional streamplot presented in Figure 7.2, we also included one-dimensional cross-section plots of the different variables in Appendix D. These figures show more details of the simulated fields, and can be used as benchmark data, whereas the figures presented in this chapter are only useful for illustration purposes.

7.1.2. Buoyancy-Driven Flow With Shear Momentum Source

In the second scenario, a momentum source is introduced into the problem domain by imposing a fixed velocity, U_{wall}, directed in the positive y direction at the plane $x = L$. This shear momentum source breaks the symmetry observed in the pure buoyancy-driven flow case with respect to the $x = \pm y$ planes. The simulation parameters for this scenario, presented in both physical and LBM units, are summarized in Table 7.2. As in the previous case, the volumetric heat transfer coefficient, γ, is reduced compared to the Tiberga benchmark case to stimulate greater temperature variations. Additionally, the velocity U_{wall} is significantly lowered from its value in the Tiberga benchmark case. This reduction ensures a balance between the competing effects of buoyancy forces, which mostly act in the center of the cubic domain, and the shear momentum introduced at the domain boundary, leading to complex three-dimensional flow patterns.

Table 7.2: Simulation parameter used for the simulation of a molten salt in a cubic box in steady state conditions. Here we assume a buoyancy-driven flow with a shear momentum source, with the same property characteristics of the fuel salt, and nuclear data as given in the Tiberga benchmark problem. The parameters are given in both physical and LBM units, where the conversion factors described in Section 3.1.6 are used to transform the physical values to the LBM values.

Parameter	Physical value	Physical unit	LBM value	LBM unit
L	2.0	m	101	ls
U_{wall}	$2.5 \cdot 10^{-2}$	ms^{-1}	$2.5 \cdot 10^{-3}$	ls lt^{-1}
ρ	$2.0 \cdot 10^{3}$	kg m^{-3}	1.0	lm ls^{-3}
g	9.81	m s^{-2}	$1.94 \cdot 10^{-3}$	ls lt^{-2}
ν	$2.5 \cdot 10^{-2}$	m^{2}s^{-1}	$1.26 \cdot 10^{-1}$	ls^{2} lt^{-1}
α	$2.5 \cdot 10^{-5}$	m^{2} s^{-1}	$1.26 \cdot 10^{-4}$	ls^{2} lt^{-1}
D	$1.67 \cdot 10^{-5}$	m^{2} s^{-1}	$8.42 \cdot 10^{-5}$	ls^{2} lt^{-1}
β_{th}	$2.0 \cdot 10^{-4}$	K^{-1}	$1.80 \cdot 10^{-1}$	lT^{-1}
γ	$1.0 \cdot 10^{5}$	W m^{-3}K^{-1}	$1.76 \cdot 10^{-2}$	lm ls^{-2} lt^{-3} lT^{-1}
C_p	$6.15 \cdot 10^{6}$	J m^{-3}K^{-1}	$5.48 \cdot 10^{2}$	lm ls^{-2} lt^{-2} lT^{-1}
T_{ref}	900	K	1.0	lT
P_{ref}	$1.0 \cdot 10^{9}$	W	$2.53 \cdot 10^{4}$	lm ls^{2} lt^{-3}
Re	2	-	2	-
Pr	1000	-	1000	-
Sc	1500	-	1500	-

Figure 7.3 presents the simulation results for the temperature field and delayed neutron source, visualized as heat maps across five equally spaced horizontal cross-sections of the cubic domain. Additionally, Figure 7.4 depicts the simulated velocity field using a three-dimensional streamplot. The temperature and delayed neutron source fields show minimal changes compared to the pure buoyancy-driven case. Heat convection and precursor transport are still visible in the center of the domain, driven by the upward motion of the fuel salt in this region due to buoyancy forces. However, some asymmetry arises in the heat maps within the xy-plane, particularly near the moving wall at $x = L$. The primary difference with the pure buoyancy-driven case, however, lies in the velocity field. The streamplot clearly demonstrates how the molten salt flow is influenced not only by buoyancy effects but also by the fixed velocity U_{wall} imposed at the $x = L$ plane. This external momentum source disrupts the spatial symmetry, introducing more complex three-dimensional flow patterns. As mentioned before, the velocity of the wall is chosen in such a way that the momentum source it creates competes with that of the buoyancy force, leading to these asymmetric flow structures. This asymmetry in the velocity field propagates to the other simulated variables due to interaction between the fields, leading to asymmetric results in all simulated variables. This three-dimensional asymmetry highlights the capability of the NTH simulation tool to simulate three-dimensional problem spaces.

The simulation including the shear momentum source yields an effective multiplication factor of $k_{eff} = 0.9610203$, corresponding to a reactivity of $\rho = -4056.1$. This value closely aligns with the reactivity obtained in the pure buoyancy-driven case.

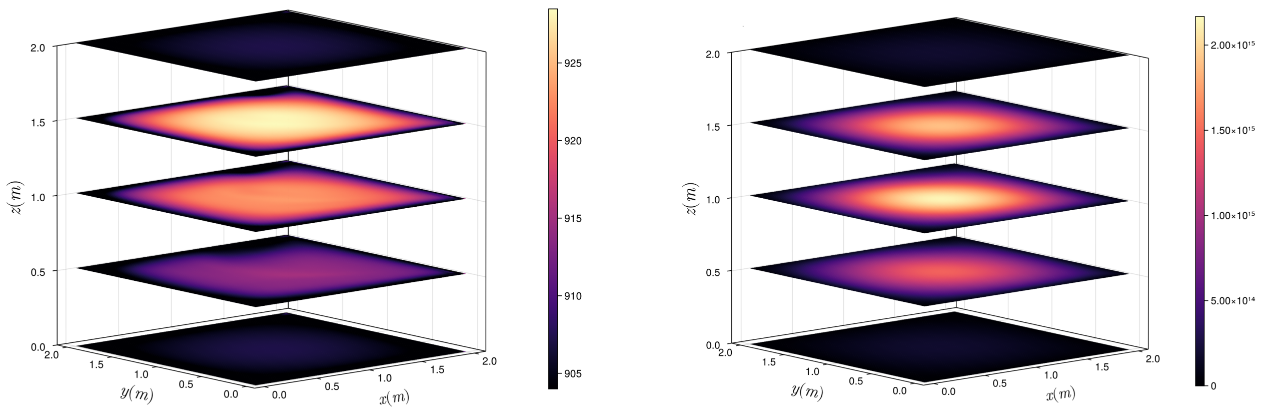

Figure 7.3: Temperature field (left) and delayed neutron source (right) from three dimensional steady-state simulation of a buoyancy-driven molten salt with shear momentum source. This figure illustrates the scalar variables using heat maps across five equally spaced horizontal cross-sections of the cubic domain. The colorbars indicate the magnitude of the scalar fields in K (left) and m^{-3}s^{-1} (right).

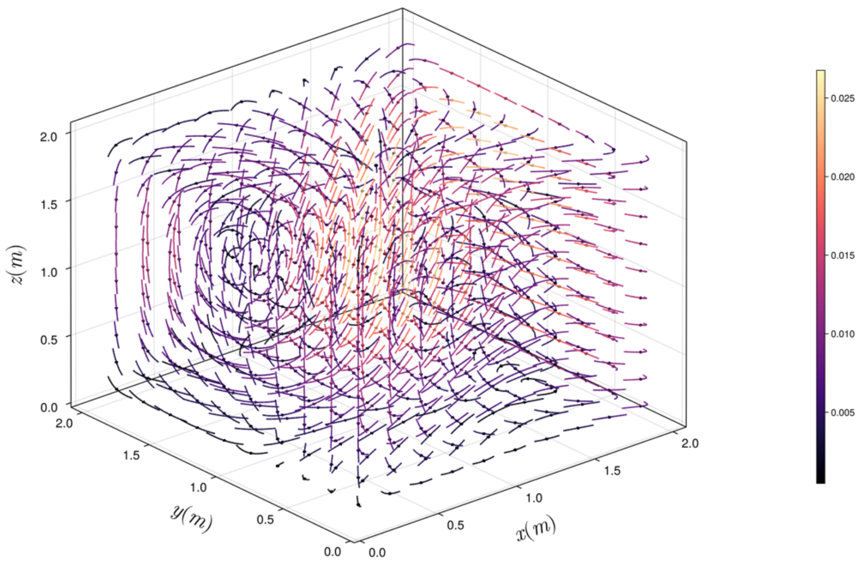

Figure 7.4: Velocity field from the three dimensional steady-state simulation of a buoyancy-driven molten salt with shear momentum source. This figure illustrates the direction of the velocity field using a three-dimensional streamplot. The magnitude of the velocity field is illustrated using the colors and the corresponding colorbar indicating the velocity in ms^{-1}.

Finally, just as in the pure buoyancy-driven case, next to the heat maps presented in Figure 7.3 and the three-dimensional streamplot presented in Figure 7.4, we also included one-dimensional cross-section plots of the different variables in Appendix D for benchmark purposes.

7.2. Turbulence Modeling Using LBM

The second major improvement that needs to be made to the NTH simulation tool in order to move towards more realistic MSFR core simulations, is the inclusion of turbulence modeling to the FM-LBM thermal hydraulics code. At this point, the simulation tool is only able to perform NTH simulations in laminar conditions, while in reality turbulent effects play a role in MSFR reactor core physics.

In the turbulent regime, the non-linear convective term in the NSE, represented by the second term in Equation 2.3, becomes sufficiently significant that even the slightest perturbations in the initial conditions or the fluid's properties lead to entirely different outcomes in the flow field, resulting in a seemingly random and chaotic behavior of the simulated fluid [14]. The turbulent regime is closely associated with high Reynolds numbers, as illustrated by the non-dimensional analysis of the NSE in Equation 2.8, where the convective term dominates the diffusive term at high Reynolds numbers. In practice, perturbations in initial conditions and fluid properties are unavoidable, meaning new modeling approaches are required to capture this non-linear behavior inherent to high Reynolds number flows. One such approach is Large Eddy Simulation (LES), which is particularly relevant to this research due to its relatively straightforward implementation within the LBM framework.

7.2.1. Energy Cascade

Turbulent flows consists of eddies of different sizes, which can be regarded as a vortex shaped motion. These eddies can be characterized by their length scale l or their wavenumber k related to each other through

$$k = \frac{2\pi}{l}\,. \tag{7.1}$$

The biggest eddies are directly driven by the external forces acting on the fluid and have a length scale, l_0, of comparable size to the characteristic length of the problem space. Due to their unstable nature, these eddies initialize a so-called energy cascade, where bigger eddies collapse into smaller eddies, thereby transferring their energy. This cascade continuous until the smallest eddies, with a length scale η, dissipate there energy into heat due to viscous forces. This energy cascade is summarized in Figure 7.5, which illustrates the energy distribution of the turbulent eddies as a function of the wavenumber. Energy is transferred from low-wavenumber (large-scale) eddies to high-wavenumber (small-scale) eddies. The largest eddies, which contain the most energy, are driven by external forces and thus exhibit a directional preference during their collapse. This behavior places them within the anisotropic range. As the energy cascade progresses and the eddies become smaller, they eventually lose this directional preference due to the dominance of viscous forces. At this stage, they enter the isotropic range, where their behavior becomes directionally independent. The isotropic range is further divided into the inertial subrange and the dissipation range. The latter contains the smallest eddies, where energy is ultimately converted into heat [14].

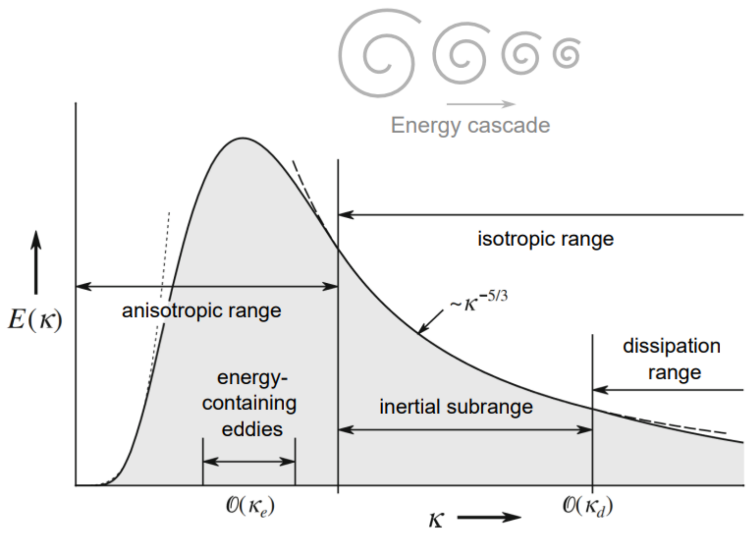

Figure 7.5: Illustration of the energy cascade in turbulent flows, showing the energy distribution of turbulent eddies as a function of the wavenumber. The figure highlights that the largest eddies, driven by external forces acting on the fluid, contain the most energy. These larger eddies transfer their energy by breaking down into progressively smaller eddies. Eventually, the eddies reach a scale where viscous forces dominate, leading to the dissipation of their energy as heat [14].

Following the Kolmogorov's first similarity hypothesis, the smallest scale eddies and the large scale eddies can be related to each other using the Reynold number, where their ratio follows the scaling law [14]

$$\frac{\eta}{l_0} \sim Re^{-3/4}\,. \tag{7.2}$$

When solving the NSE in the turbulent regime using Direct Numerical Simulation (DNS), the spatial discretization must be fine enough to resolve the smallest eddies. However, as predicted by Kolmogorov's hypothesis, the size of these smallest eddies decreases with increasing Reynolds numbers, necessitating progressively finer grid spacing for higher Reynolds number flows. This requirement makes DNS computationally infeasible for many practical applications. As a consequence, alternative modeling approaches are needed that remain computationally feasible while still accurately capturing the physics of high Reynolds number flows.

7.2.2. Large Eddy Simulation

In LES, the turbulent eddy scales are separated into large-scale eddies and sub-grid eddies. In this technique, all eddies up to a certain cut-off length Δ are resolved using the complete equations of motion while a different sub-grid model is used to calculate a combined contribution of the remaining sub-grid eddies [39]. Conceptually, this is equivalent to applying a high-pass spatial filter to the scalar fields. If we define $\widetilde{\phi}$ as the resolved part of the complete scalar field ϕ, a convolution kernel G, associated with the cut-off length Δ, can be defined such that

$$\widetilde{\phi}(\mathbf{x}) = \int_{-\infty}^{\infty} G(\boldsymbol{\xi})\phi(\mathbf{x}-\boldsymbol{\xi})d\boldsymbol{\xi}\,. \tag{7.3}$$

By requiring that this filter should be consistent, linear, and commute with differentiation, specific characteristics for the filter can be derived. These properties enable the filter to be applied directly to the governing equations [37], leading to the Filtered Navier-Stokes Equations (FNSE), defined as

$$\rho\frac{\partial \widetilde{u}_i}{\partial t} + \rho\frac{\partial \widetilde{u}_i\widetilde{u}_j}{\partial x_j} = \rho f_i + \frac{\partial}{\partial x_j}\left[\widetilde{p}\delta_{ij} + \mu\frac{\partial \widetilde{u}_i}{\partial x_j} - \rho\tau^R_{ij}\right]\,. \tag{7.4}$$

This equation illustrates that the FNSE for the resolved fields is similar to that of the normal NSE, with the addition of a correction term, $\rho\tau^R_{ij}$, known as the sub-grid stress tensor. This term captures the combined effect of the sub-grid scales that are omitted by the high-pass spatial filter [14]. As a consequence, LES is associated with modeling the flow field of the turbulent eddies up to a certain cut-off length using the standard NSE, subsequently a so-called sub-grid scale (SGS) model is applied to simulate the correction term $\rho\tau^R_{ij}$.

One commonly applied class of SGS models, known as Eddy-Viscosity Models, simulate the sub-grid stress tensor through the eddy-viscosity, ν_t, related to the sub-grid stress tensor via the relationship [14]

$$-\rho\tau^R_{ij} = \rho\nu_t\left(\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{\partial \widetilde{u}_j}{\partial x_i}\right) - \frac{1}{3}\rho\tau^R_{kk}\delta_{ij}\,. \tag{7.5}$$

These Eddy-Viscosity Models provide approximate solutions for the eddy viscosity based on the resolved fields of the FNSE. They are particularly advantageous for LBM simulations, as the calculated eddy viscosity can be directly incorporated into the LBM framework.

7.2.3. LES-LBM

LES is a specifically attractive turbulence model approach when using LBM for fluid simulation. First of all, the cut-off length Δ is naturally implied by the spacing of the lattice grid. This means the grid acts as an implicit filter where only eddies with a characteristic length larger than the grid spacing are resolved using LBM. Secondly, the LBM algorithm allows for the direct incorporation of eddy viscosity into the framework by defining an effective viscosity, ν_e, defined as

$$\nu_e = \nu_0 + \nu_t \,, \tag{7.6}$$

where ν_0 denotes the molecular viscosity, normally used in LBM simulations. This effective viscosity is then used in the collision step of the LBM algorithm [121].

The challenge, however, lies in determining the eddy viscosity from the resolved fields using Eddy-Viscosity Models. The Smagorinsky model [85], one of the earliest and most widely applied models, simulated the eddy viscosity as

$$\nu_t = C_S^2 \Delta^2 \left| \widetilde{S}_{ij} \right| , \tag{7.7}$$

with

$$\widetilde{S}_{ij} = \frac{1}{2} \left(\frac{\partial \tilde{u}_j}{\partial x_i} + \frac{\partial \tilde{u}_i}{\partial x_j} \right) . \tag{7.8}$$

Here, C_S denotes the Smagorinsky constant which has values generally ranging from 0.05 to 0.16 [115]. The tilde notation indicates that the eddy viscosity is computed using the resolved velocity fields from the FNSE. Due to its simplicity and ease of implementation, the Smagorinsky model is a popular choice in LES research. For instance, Wang et al. [110] use the Smagorinsky model in their LBM-NTH simulation tool, where they implement both thermal hydraulics and neutronics into the LBM framework. However, its simplicity comes at a cost, when turbulent fields are calculated using the Smagorinsky model, they generally show an overly dissipative nature near walls [107]. To address this limitation, more advanced Eddy-Viscosity Models have been developed, such as the Vreman model [106] and the Wall-Adapting Local Eddy (WALE) model [73]. Similar to the Smagorinsky model, these models calculate eddy viscosity from the resolved velocity fields but use more sophisticated formulations to improve accuracy. As a result, they provide a better representation of turbulent behavior near walls. These advanced models have also found successful implementations in LES-LBM studies. For example, Zhou et al. [121] utilize the Vreman model in combination with LES-FM-LBM to simulate turbulent channel flows, while Zhang et al. [120] apply the WALE model in LES-LBM to investigate turbulence in electrohydrodynamics simulations.

7.3. MSFR Core Geometry Implementation

The third and final significant improvement required for the simulation tool to advance toward actual MSFR simulations is the incorporation of the MSFR core geometry into the NTH tool. Although the current simulation tool can perform 3D simulations, it is limited to Euclidean geometries. This limitation stems from the boundary condition schemes implemented in the FM-LBM algorithm, which are restricted to problem boundaries that align precisely with the Euclidean lattice grid. However, real-world geometries, such as those of an MSFR reactor core, introduce complex boundary shapes that do not necessarily conform to this grid. To address this challenge, advanced boundary condition techniques must be integrated into the FM-LBM algorithm in order to handle these types of complex geometries.

Before examining potential LBM boundary schemes for complex boundary shapes, we first define the geometry of the MSFR reactor core to be simulated. As described in the introduction section of this report, the MSFR core features a toroidal shape with 16 segments of heat exchangers and pumps symmetrically arranged around the core. This symmetry enables the use of angular periodicity, allowing simulations to focus on a single segment of the MSFR core while applying periodic boundary conditions along the azimuthal boundaries to effectively replicate the behavior of the entire domain. Tiberga [92] demonstrated that this can be accomplished by simulating a ”pizza-slice” of the full domain, encompassing one group of heat exchangers and pumps. This approach captures 1/16th of the entire problem domain and is illustrated in Figure 7.6.

In this section, we address the implementation of the MSFR core geometry, as depicted in Figure 7.6, within the NTH simulation tool. It is important to emphasize that the focus is not on the Phantom-S_N algorithm, as it already incorporates the MSFR core geometry. This is achieved through the application of an adaptive

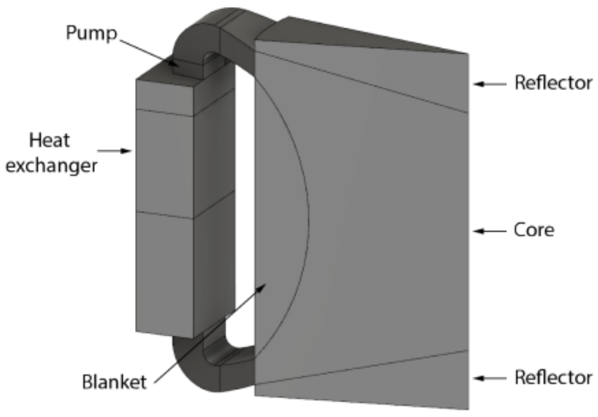

Figure 7.6: Problem geometry of an MSFR core segment for numerical simulation. This geometry represents a single segment of the full reactor core, containing one group of heat exchangers and pumps. It leverages the angular periodicity of the MSFR to simulate the entire reactor core by applying periodic boundary conditions along the azimuthal boundaries. The figure also highlights additional components relevant to the reactor core, such as the fertile blanket and neutron reflectors, as described in Figure 1.1 [92].

mesh in the DG-FEM algorithm, along with reflective boundary conditions on the sides of the problem domain and vacuum boundary conditions elsewhere [92]. Instead, the discussion focuses on the implementation of the reactor segment geometry within the LBM framework, particularly exploring boundary schemes to handle complex boundary shapes. First, we will examine the implementation of periodic boundary conditions on the sides of the reactor segment. Subsequently, we will address bounce-back methods for enforcing no-slip boundary conditions on complex-shaped boundaries. The latter boundary schemes can be applied to other parts of the reactor segment, such as the pump, heat exchanger, reflector, and fertile blanket. Lastly, this section will not delve into additional implications of these components, such as the momentum source introduced by the pump or heat removal by the heat exchangers.

7.3.1. Periodic Boundary Conditions

Symmetry allows us to simulate only 1/16th of the entire problem domain by introducing periodic boundaries at the sides of the segment. These periodic boundaries ensure that molten salt exiting one side of the segment reenters the domain at the opposite side. However, implementing this symmetry within the LBM algorithm in the current situation presents challenges due to the angle $\frac{1}{16} \cdot 2\pi$ associated with the simulated segment. This angle results in boundaries that are positioned differently with respect to the Euclidean lattice grid across different ghost domains connected to the simulated domain. The primary issue arises because particle distributions leaving the domain on one side must be rotated by this angle before reentering on the opposite side to maintain angular symmetry. This rotation, however, causes the reentering particle distributions to correspond to lattice sides and directions that do not align with the lattice grid. This problem is illustrated in Figure 7.7 using a simplified domain projected on the xy-plane. The figure shows how the lattice grid orientation differs between the simulated domain and a ghost domain. Furthermore, it shows how a horizontally entering particle distribution on one side corresponds to an exiting particle distribution on the other side that is rotated by $\frac{1}{16} \cdot 2\pi$. The exiting particle density, however, no longer corresponds to a valid lattice side in the Euclidean lattice grid, due to the different orientation of the lattice grid in the ghost domain.

There are multiple ways to tackle this problem. One option is to simulate a quarter of the problem domain, meaning we simulate four out of the sixteen segments. In this setup, particle distributions leaving the domain are rotated 90 degrees before reentering on the opposite side. As a result, the particle distributions align with the lattice grid again, due to the Euclidean grid's symmetry under a 90-degree rotation. The obvious drawback of this approach is that it does not fully exploit the symmetry of the problem domain. By simulating four segments instead of one, we effectively quadruple the computational effort, which is highly undesirable in large three-dimensional simulations.

A second option is to transform the LBM into curvilinear space, as demonstrated by [117]. In this approach, a cylindrical coordinate system is applied to the lattice grid. This allows the number of lattice nodes along the azimuthal axis to be chosen such that the physical boundaries corresponding to the sides of the segment align perfectly with the lattice grid. Consequently, particle distributions can move seamlessly between neighboring lattice nodes across the periodic boundaries. While this approach appears to be an ideal solution for implementing periodic boundary conditions along the sides of the domain, it introduces significant challenges for other boundaries, such as the channel containing the pump and heat exchangers, which require

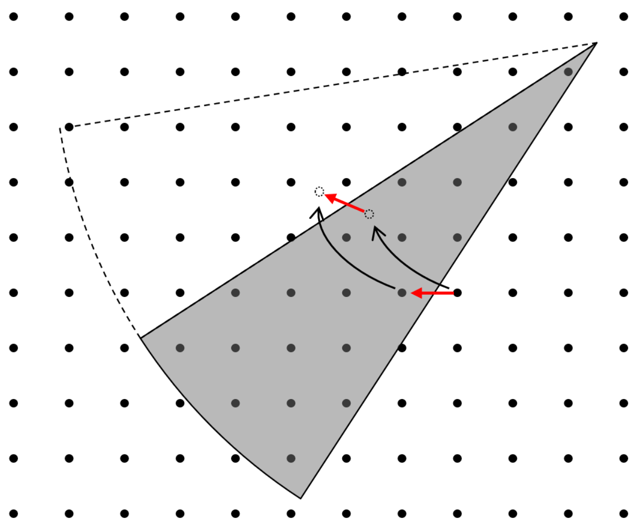

Figure 7.7: Visualization of the challenge in implementing periodic boundary conditions along the sides of a segment of the MSFR core problem domain. The figure illustrates how the orientation of the square lattice grid differs between the simulated domain and a ghost domain. It also depicts how a horizontally entering particle distribution on one side corresponds to an exiting particle distribution on the other side that is rotated by $\frac{1}{16} \cdot 2\pi$. The exiting particle density, however, no longer corresponds to a valid lattice side in the Euclidean lattice grid, due to the different orientation of the lattice grid in the ghost domain.

no-slip conditions. These boundaries will not align with the cylindrical lattice grid, making it impossible to apply straightforward bounce-back methods. Instead, specialized boundary schemes must be developed to handle these complex boundary geometries. However, as will be discussed in the next section, even in simple Euclidean lattice grids, addressing such boundaries poses challenges and requires advanced numerical techniques. Extending these methods to cylindrical coordinates will further complicate the implementation of bounce-back boundary conditions. Moreover, transforming the LBM algorithm into curvilinear space compromises one of its key advantages: its simplicity. In curvilinear space, additional corrections are required to address issues such as the apparent anisotropy of diffusion introduced by the coordinate transformation. Furthermore, managing memory storage for lattice information becomes more complex. For example, maintaining the relationships between neighboring nodes in cylindrical coordinates complicates the use of square arrays in memory. These considerations make the curvilinear approach less practical for the current problem.

A final solution that avoids both the expansion of the simulated domain and the transformation of the lattice grid involves the use of interpolation techniques to calculate the exiting particle distributions on one side that correspond to the incoming particle distributions on the other side. In Figure 7.7, this approach entails using interpolation to determine the exiting particle distribution function moving at an angle, which is then equated to the horizontally incoming particle distribution function on the opposite side. Various methods can be employed to perform this interpolation. One option is to directly interpolate the particle distribution functions, as demonstrated by [119], who applied second-order upwind interpolation to the particle distribution functions. Alternatively, macroscopic quantities can be interpolated instead of the particle distributions themselves. This can be achieved using conventional techniques, such as B-Spline interpolation, as discussed in Section 3.4.5. The interpolated macroscopic quantities can then be rotated to preserve angular symmetry and subsequently decomposed back into particle distribution functions. In either case, interpolation need only be applied to a finite number of points, corresponding to the amount of particle distributions entering the domain on the other side. Consequently, the computational effort remains minimal, while the simplicity of the LBM algorithm is also preserved by avoiding transformations of the lattice grid.

7.3.2. No-Slip Boundary Conditions

In addition to the periodic boundary conditions applied to the sides of the domain, no-slip boundary conditions must be imposed on the remaining boundaries. These boundaries are located along the reflectors, the fertile blanket, and the channel containing the heat exchanger and the pump. Similar to the sides of the segment, these boundaries will not align with the square lattice grid. Consequently, specialized numerical techniques are required to implement no-slip conditions on these complex-shaped boundaries. In this section, we discuss three possible numerical schemes based on the bounce-back method for applying no-slip boundary conditions to such boundaries. These techniques can be directly applied to the remaining boundaries of the problem domain.

Staircase Approximation

One of the most popular methods for implementing complex boundaries within the LBM framework is through the usage of the staircase approximation. This approach involves overlaying the problem geometry onto a square lattice grid, where lattice nodes inside the problem domain are designated as inner nodes (or fluid nodes), and those outside the domain are classified as outer nodes (or solid nodes). The numerical boundary is then defined as the halfway point between consecutive inner and outer lattice nodes, resulting in the physical boundary being approximated by a staircase-like shape. Figure 7.8 illustrates the staircase approximation for a circular geometry. Note that the number of lattice points in this example is deliberately kept very low for illustration purposes, in practice such a resolution would be insufficient for accurate simulations.

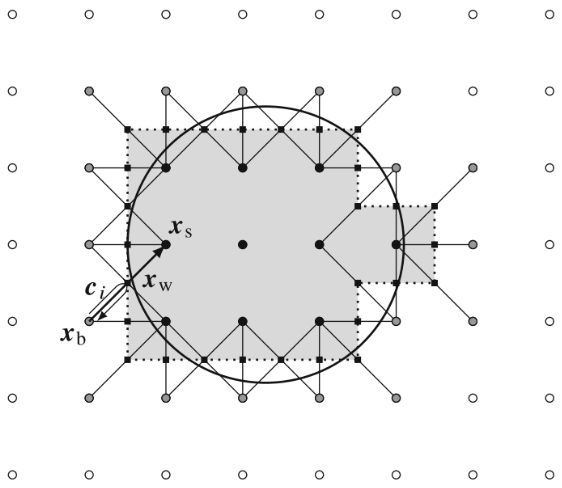
$\boldsymbol{x}_\mathrm{s}$
$\boldsymbol{c}_i$
$\boldsymbol{x}_\mathrm{w}$
$\boldsymbol{x}_\mathrm{b}$

Figure 7.8: Illustration of the staircase approximation for a circular domain. The staircase approximation defines the lattice node falling inside the circle as inner (fluid) nodes, while the other lattice nodes as defined as outer (solid) nodes. The numerical domain is defined by halfway points between consecutive inner and outer nodes, leading to the staircase-shaped numerical domain, illustrated by the gray area in the figure [55].

The no-slip boundary condition is implemented by combining the staircase approximation with the standard bounce-back method described in Equation 3.23. In this method, particle distribution functions leaving the approximated domain are reflected back into the problem domain by reversing their propagation direction at the boundary. This approach is commonly referred to as the Simple Bounce-Back (SBB) method, which for most applications is shown to be first-order accurate with respect to the lattice grid spacing. Its primary advantages include ease of implementation and exact mass conservation, regardless of the shape of the boundary geometry. The latter property is particularly significant, as it is often violated by most higher-order accurate boundary condition methods [40]. The main drawback of the SBB method is that it approximates the physical boundary with a staircase shape, which prevents the no-slip boundary condition from being precisely applied at the physical boundary. Consequently, this approximation introduces artificial slip at the physical boundary, leading to inaccuracies in the results [55].

In essence, the SBB method is identical to the standard bounce-back scheme, except that it is applied to a rectangular boundary that approximates the physical boundary. However, more advanced boundary schemes have been proposed in the literature that modify not only the boundary approximation but also the numerical computations of the bounce-back method itself. These schemes provide higher-order accuracy compared to the SBB method and should be considered if the SBB method does not yield satisfactory results. Two such advanced bounce-back methods are discussed next. However, the LBM literature also proposes additional approaches, including ghost methods and immersed-boundary methods. For a comprehensive overview of these additional techniques, we refer to the book of Krüger et al. [55].

Interpolated Bounce-Back

Given that the numerical boundary condition is defined at the halfway point between consecutive inner and outer lattice nodes, the numerical boundary rarely coincides precisely with the physical boundary. Most of the time, the physical boundary is positioned at some other point between these two lattice nodes. This discrepancy is the main limitation of the SBB scheme, which approximates the physical boundary with a staircase-like shape. The concept behind Interpolated Bounce-Back (IBB) methods is to address this error by incorporating additional information about the position of the physical boundary within the lattice grid [55].

One such IBB method, introduced by Bouzidi et al. [12], defines the bounce-back scheme as

$$f_{\bar{i}}\left(\mathbf{x}_{\mathrm{b}}, t+\Delta t\right)=\left\{\begin{array}{ll} 2q f_i^*\left(\mathbf{x}_{\mathrm{b}}, t\right)+(1-2q) f_i^*\left(\mathbf{x}_{\mathrm{f}}, t\right) & q \leq \frac{1}{2} \\ \frac{1}{2q} f_i^*\left(\mathbf{x}_{\mathrm{b}}, t\right)+\frac{2q-1}{2q} f_{\bar{i}}^*\left(\mathbf{x}_{\mathrm{b}}, t\right) & q \geq \frac{1}{2} \end{array}\right. . \tag{7.9}$$

Here, $\bar{i}$ represents the direction opposite to i, f_i^* and f_i denote the pre- and post-streaming distribution functions, $\mathbf{x}_{\mathrm{b}}$ is the boundary lattice node within the fluid domain, indicating that the physical boundary lies between $\mathbf{x}_{\mathrm{b}}$ and the consecutive solid node $\mathbf{x}_{\mathrm{s}}$ outside the domain. Additionally, $\mathbf{x}_{\mathrm{f}}$ refers to the nearest fluid node beyond $\mathbf{x}_{\mathrm{b}}$ within the domain. The parameter q represents the normalized distance of the physical wall and is defined as $q = d/(\mathbf{c}_i \Delta t)$, where d is the distance between $\mathbf{x}_{\mathrm{b}}$ and the physical boundary.

The IBB scheme operates under the principle that particle distribution functions travel a distance of $|\mathbf{c}_i|\Delta t$ in each time step and are reflected at the problem boundaries. However, when the physical boundary is not exactly halfway between two consecutive lattice nodes, particle distributions may end up at an intermediate position within the lattice grid. The IBB method addresses this by interpolating the pre-streaming particle distributions within the grid, ensuring that after streaming and reflection at the physical wall, the particle distributions end up exactly on the boundary lattice node. This is illustrated in Figure 7.9, where an interpolated particle distribution function is located between $\mathbf{x}_{\mathrm{f}}$ and $\mathbf{x}_{\mathrm{b}}$, ultimately arriving at $\mathbf{x}_{\mathrm{b}}$ after streaming. It is worth noting that Equation 7.9 simplifies to the SBB scheme when the physical boundary is positioned precisely halfway between two consecutive lattice nodes. This result is intuitive, as in such a scenario, the particle distributions naturally return to the boundary node after traveling a distance of $|\mathbf{c}_i|\Delta t$.

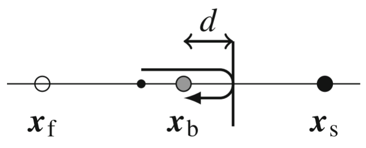
d
$\boldsymbol{x}_{\mathrm{f}}$
$\boldsymbol{x}_{\mathrm{b}}$
$\boldsymbol{x}_{\mathrm{s}}$

Figure 7.9: Visual representation of the IBB boundary scheme. Here, an interpolated post-streaming particle distribution function is found between $\mathrm{x}f$ and $\mathbf{x}_{\mathrm{b}}$ such that, the particle distribution arrives exactly at $\mathbf{x}_{\mathrm{b}}$ after streaming [55].

The IBB scheme introduced by Bouzidi et al. achieves second-order accuracy, significantly reducing the model error compared to the first-order accuracy of the SBB method. However, similar to SBB, the IBB scheme still introduces artificial slip near the boundary. Additionally, as previously mentioned, the IBB boundary method is not mass-conserving due to the interpolation inherent to the IBB scheme. Therefore, one should assess whether the improvements in modeling accuracy outweigh potential errors resulting from the loss of mass-conservation [12].

Partially Saturated Bounce-Back

Another alternative to the SBB method is the Partially Saturated Bounce-Back (PSBB) boundary method. In this scheme, lattice nodes represent computational cells that can exhibit a mixture of fluid and solid properties. This concept is illustrated in Figure 7.10, where a nodal fluid/solid fraction, ϵ, determines the extent to which a cell is fluid or solid. Specifically, $\epsilon = 0$ corresponds to a pure solid, while $\epsilon = 0$ indicates a pure fluid. As a consequence, partially saturated cells emerge at the boundary, with the solid/fluid fraction taking intermediate values $0 < \epsilon < 1$, depending on the proportion of the lattice cell volume that falls within the problem domain defined by the physical boundary. To accommodate partially saturated cells, the LBM equation, as given in Equation 3.1, is modified into

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) - f_i(\mathbf{x}, t) = \Delta t\left[\,(1-B(\epsilon))\Omega_i(f) + B(\epsilon)\Omega_i^s(f)\right] . \tag{7.10}$$

Here, $B(\epsilon)$ is a constant related to the solid/fluid fraction of the lattice cell, and Ω_i^s is the collision operator for solid nodes, defined as

$$\Omega_i^s(f) = \left(f_{\bar{i}}(\mathbf{x}, t) - f_{\bar{i}}^{eq}(\rho, \mathbf{u})\right) - \left(f_i(\mathbf{x}, t) - f_i^{eq}(\rho, \mathbf{u})\right) . \tag{7.11}$$

The PSBB scheme operates as follows: for pure fluid nodes, the LBM scheme performs the standard collision step. Conversely, for partially saturated cells, the collision operator is applied only to the fluid fraction, while a bounce-back scheme is used for the solid fraction. Specifically, the bounce-back scheme implemented within the solid collision operator, Ω_i^s, and acts only on the non-equilibrium component of the distribution functions.

The PSBB boundary method offers several advantages over the previously discussed boundary schemes. First, it provides enhanced accuracy compared to SBB, resulting in significantly smoother fluid motion, as

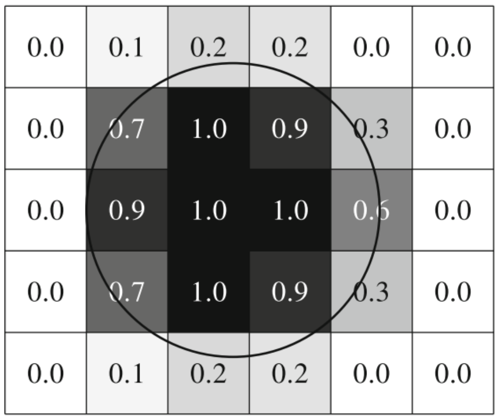

Figure 7.10: Visual representation of the PSBB boundary method. Boundary nodes are considered partially saturated cells whose fluid/solid fraction determines the extent to which a cell corresponding to a node is solid or fluid. By extending the LBM scheme, PSBB applies normal collision to the fluid fraction of the boundary cell, while a bounce-back scheme is performed on the non-equilibrium part of the distribution functions for the solid fraction [55].

demonstrated by [88]. The exact order of accuracy, however, is strongly influenced by the precision in determining the fluid/solid fractions. Additionally, in contrast to the IBB boundary scheme, the PSBB method is exactly mass-conserving, given that the PSBB scheme is essentially a more sophisticated staircase approximation. However, despite these advantages, the PSBB scheme does not fully eliminate the introduction of artificial slip near boundaries. Since it still relies on the staircase approximation, it cannot fully capture the true geometry of the boundary.

The PSBB boundary method can be employed when neither the SBB nor the IBB boundary conditions yield satisfactory results. However, two critical challenges must be addressed in this context: the determination of the constant $B(\epsilon)$ and the accurate computation of ϵ. For the standard BGK implementation of the LBM, the constant $B(\epsilon)$ is defined in the literature as

$$B(\epsilon, \tau) = \frac{\epsilon(\tau - 1/2)}{(1-\epsilon) - (\tau - 1/2)}, \tag{7.12}$$

where τ denotes the relaxation time of the BGK operator. In the case of an FM-LBM implementation, this parameter must instead correspond to the relaxation time embedded within the filter matrix, necessitating the redefinition of $B(\epsilon)$. Additionally, a robust method is required to determine the solid/fluid fractions for the boundary lattice cells, given that the accuracy of the PSBB scheme heavily depends on these fractions. Notably, these fractions only need to be calculated once, at the start of the simulation, given that we are dealing with a static problem domain. As a result, sophisticated methods, such as, sub-cell or cut-cell approaches as proposed in the literature can be used to achieve this [21, 45].

8 Conclusion and Recommendations

This research focused on the development of a novel NTH simulation tool to model the multiphysics behavior of an MSFR reactor core. The designed tool integrates a GPU-accelerated FM-LBM algorithm, employed for thermal fluid and precursor transport simulations, with the existing transport code Phantom-S_N for neutronics calculations. This coupled system has produced a fully functional multiphysics simulation framework tailored for MSFR analyses. This chapter presents the conclusions drawn from this work, structured into six sections. Sections 8.1 to 8.4 address each of the research goals outlined in Section 1.3. Following this, Section 8.5 provides an overall conclusion regarding the integration of GPU-accelerated FM-LBM techniques within MSFR core simulation tools. Lastly, Section 8.6 offers recommendations for future research, providing guidance for further advancements in the development of the NTH simulation tool.

8.1. FM-LBM Model Development

The first part of this research focused on developing a GPU-accelerated FM-LBM algorithm to simulate thermal hydraulics and precursor transport. This algorithm employs a double distribution function approach, where separate distribution functions are defined for the velocity and temperature fields. This concept was extended further by incorporating additional distribution functions for the different precursor families. The algorithm operates through an alternation of propagation and collision steps on a lattice grid for each distribution function. Interactions between the different fields occur during the collision steps, either through convective terms in the filter matrix multiplications or via source terms. This approach simulates the evolution of the thermal fluid, which, under stable conditions, converges to the steady-state solution after a sufficient number of simulation steps.

The FM-LBM algorithm's symmetric calculations during propagation and collision steps on each lattice node make it an ideal candidate for parallel computation using GPUs. Consequently, the algorithm was implemented using the Julia-CUDA framework, enabling efficient utilization of NVIDIA GPUs for simulation tasks. In this setup, simulation steps are executed through kernel functions, which define operations at individual lattice sites and are performed concurrently across GPU cores. The CPU manages the execution by launching kernel functions on the GPU and ensuring proper synchronization of the entire lattice domain between successive kernel launches. For the implementation of the algorithm, the Julia-CUDA framework was specifically chosen due to its high-level syntax, superior computational performance compared to other high-level languages such as Python, and the flexibility it offers for GPU programming. This flexibility facilitated the optimization of GPU performance through low-level memory management, leveraging the hierarchical memory architecture of NVIDIA GPUs. For example, the localized but faster shared memory was employed to store the solution vector during the collision kernel, significantly reducing memory access latency. Additionally, large multidimensional arrays stored in global memory were transformed into one-dimensional arrays with carefully structured data ordering. This transformation improved memory coalescence during matrix multiplications within the collision kernel, further enhancing computational efficiency. As a result, a highly optimized GPU-accelerated FM-LBM algorithm was developed, with its computational performance benchmarked against other studies focusing on GPU-optimized LBM algorithms, as will be further elaborated on in Section 8.4.

8.2. Coupling Mechanism

The second part of this research focused on integrating the newly developed FM-LBM simulation tool with the existing Phantom-S_N algorithm for neutronics simulation, creating a fully integrated framework for MSFR reactor core simulations that captures both thermal hydraulics, precursor transport, and neutronics. However, significant disparities between the two algorithms in terms of computational frameworks, mathematical approaches, and physical modeling posed numerous challenges to their integration.

The first challenge was combining the two algorithms into a single codebase, given that the Phantom-S_N algorithm is written in Fortran-90, while the FM-LBM algorithm is written in Julia. This challenge was addressed by leveraging the built-in functionality of Fortran-90 to perform system calls via the command-line interface. Phantom-S_N was designated as the master operator, making periodic system calls to the Julia executable. During these calls, the FM-LBM algorithm executes, and Fortran automatically waits for its completion before continuing. Moreover, data exchange between the two codes was facilitated through simple I/O operations, where relevant data, such as variable values at lattice nodes, was communicated via plain text files.

A further challenge arose from the differing spatial discretization methods used by the two algorithms. Phantom-S_N employs a DG-FEM approach, where the spatial domain is subdivided into mesh elements, and variables within each element are expressed as linear combinations of basis functions and their coefficients. In contrast, FM-LBM describes physical fields on its lattice nodes. Consequently, transferring information between the two algorithms required translating lattice node variables into coefficients of the DG-FEM basis functions. This was achieved using Galerkin projection. Physical variables were first interpolated on the LBM lattice grid using B-spline interpolation to extract physical quantities at the quadrature points of the DG-FEM mesh elements. Subsequently, by approximating integrals over mesh elements with weighted summations of the integrand at quadrature points, the physical fields were transformed into the coefficients of the basis functions using Galerkin projection.

Finally, the sequence in which the two algorithms perform calculations and exchange information had to be carefully determined based on simulation conditions. For steady-state simulations, the k_{eff} eigenvalue problem was solved using the power method. This iterative algorithm dictated when the NTE was solved using Phantom-S_N and when precursor transport was calculated using FM-LBM. It also fixes how information must be transferred between the two algorithm, such as using the delayed neutron source from the precursor calculations to transform the fission source term of the NTE on the right hand side. Inherent to this method is that both algorithms operate in steady-state, requiring the FM-LBM algorithm to run until convergence. Conversely, transient simulations were more straightforward to implement, with both algorithms performing calculations for a single time step in sequence. This setup resulted in a simple alternation between the two codes, where the output of one code served as the input for the other via source terms.

8.3. Comparison to Benchmark

This research benchmarked the NTH simulation tool against existing literature in three distinct stages. First, the FM-LBM algorithm for thermal hydraulics was validated separately by simulating a side-heated square cavity. This initial benchmarking ensured that the FM-LBM algorithm produced satisfactory results before advancing to its coupling with the Phantom-S_N algorithm. In the second stage, the full multiphysics simulation tool under steady-state conditions was validated using the Tiberga benchmark case. This study examines the fully coupled NTH solution within a lid-driven square cavity and compares simulation results among four different multiphysics simulation tools developed by various universities and institutions. In their study, the coupling process is executed incrementally, enabling the identification and resolution of potential errors within specific components of the multiphysics simulation tool. Finally, the same Tiberga benchmark was utilized to validate the transient multiphysics simulations, analyzing system responses in power output due to perturbations in frequency domain on the heat sink term. In all three stages of validation, the algorithm demonstrated satisfactory alignment with results from the literature.

The side-heated square cavity case is a well-established benchmark problem, with many studies reporting maximum horizontal and vertical velocity values along the centerlines of the square domain. Our simulation results were compared to a few of these studies, revealing strong agreement with average relative discrepancies under 1%, confirming the FM-LBM algorithm's capability for accurate thermal fluid simulations.

The Tiberga benchmark study, however, poses greater sensitivity to modeling approaches. That is why, rather than reporting single maximum values, this benchmark study compares entire variable fields, including velocity, temperature, fission rate density, and delayed neutron source, along the horizontal and vertical centerlines of the square domain for steady-state simulations. To facilitate quantitative comparisons, average discrepancies were computed by comparing physical field values from the benchmark codes and our simulation at 200 evaluation points along the centerlines. For all but one step of the benchmark problem, the discrepancies across different fields exhibited average relative errors below 1%. The exception was the differential fission rate density reported in step 1.2 of the benchmark, however, this is consistent with findings from the benchmark study, which also reports larger discrepancies among the different benchmark codes for this field.

Two notable observations emerged from the steady-state results of the Tiberga benchmark. First, our simulation results aligned most closely with the TUD-S_6 results, as this was the only benchmark code that also solves the complete NTE without relying on the neutron diffusion approximation. This alignment was particularly evident in the reactivities reported with each benchmark step. Here, we observed significant deviations between our results and those of the benchmark codes, except for TUD-S_6 results. The benchmark study addressed this by also reporting differential reactivity, which measures reactivity differences between simulations with and without additional features like precursor transport or power coupling. By focusing on these differences, this metric essentially controls for the different handling of the neutronics simulations. Consequently, agreement was found with all benchmark codes for the reported differential reactivities.

The second point concerns the use of very high Prandtl and Schmidt numbers in the benchmark study, which effectively ignore diffusion effects. These high values creates challenges for LBM simulations, as maintaining stability requires a very fine lattice grid. This greatly increases the computational cost, making LBM simulations infeasible. However, since diffusion effects are already minor for small temperature and precursor density gradients, we found that the benchmark results could be approximated using lower Prandtl and Schmidt numbers. With a Prandtl number of 1000 and a Schmidt number of 1500, our simulation tool already produced satisfactory results without requiring excessive computational resources. The only noteworthy discrepancies compared to the benchmark appeared near the domain boundaries, which can be attributed to the larger temperature and precursor density gradients in these areas. Additionally, we show that further increasing these values does not offer significant improvements in the simulation results.

Finally, in the transient case of the Tiberga benchmark study, the power output of the simulation tool in response to heat sink perturbations in the frequency domain was analyzed. The study reported normalized gains and phase shifts in power output across a range of perturbation frequencies in the heat sink term. Our simulation results demonstrated satisfactory agreement with the benchmark data, confirming that the fully coupled NTH tool provides accurate simulations for both steady-state and transient scenarios.

8.4. Computational Performance

This research also evaluated the computational performance of the simulation tool, with a particular focus on the FM-LBM algorithm. This focus is justified for several reasons. First, this research did not directly contribute to the development of the Phantom-S_N codebase. Instead, we only implemented code to facilitate its coupling with the FM-LBM framework, without making any efforts to optimize the performance of the DG-FEM method used in Phantom-S_N. Second, the coupling mechanism itself significantly impacts computational performance. This is primarily due to the differences in programming languages, which results in the overhead associated with the reading and writing to plain text files, and the startup time of the Julia code. These performance bottlenecks could potentially be addressed in future research by re-implementing Phantom-S_N in Julia to leverage its GPU capabilities. However, this was beyond the scope of the current work. Finally, we specifically investigated the GPU implementation of FM-LBM to achieve superior performance compared to serial implementations. Moreover, Julia-CUDA was specifically chosen to exploit GPU functionalities, as it offers better performance compared to other scientific programming languages, such as Python, that also support CUDA. This makes it particularly relevant to compare the performance of our FM-LBM implementation with existing literature.

In the side-heated cavity benchmark case, the FM-LBM algorithm achieved a peak performance of 390 MLUPS using double-precision floating-point numbers. Compared to studies focused on optimizing GPU-accelerated LBM implementations (excluding multi-GPU setups), this performance is within a reasonable range. These studies report peak simulation speeds of up to 1200 MLUPS but use more simplified problem settings. For instance, they typically simulate only the velocity field, whereas the side-heated cavity benchmark in this research includes both velocity and temperature fields, effectively doubling the computational workload. Additionally, these studies employ the BGK collision operator, which is computationally less demanding than the filter matrix collision operator used in this work. The filter matrix collision operator involves double matrix multiplications, making it the most computationally intensive task in our simulations. This explains why the collision kernel's computation time is approximately three times that of the propagation and boundary condition kernels combined.

In the Tiberga benchmark case, we further increased the computational load by including eight additional distribution functions for the different precursor families, resulting in a fivefold increase in computational requirements. Under these conditions, the FM-LBM algorithm achieved a performance of 57 MLUPS with double-precision floating-point numbers. This represents a performance drop by a factor of approximately seven compared to the side-heated cavity benchmark case instead of five. While performance losses up to a factor five can be attributed to the finite memory bandwidth of the GPU, the additional drop in performance stem from other performance bottlenecks. Potential issues are increased CPU overhead and memory overspill, where excess demands on shared and register memory redirect allocations to slower global memory. Despite this drop, the observed performance of 57 MLUPS for the full FM-LBM implementation is still considered satisfactory, particularly when compared to serial implementations of NTH tools using LBM methods. When corrected for the fact that our tool does not incorporate neutronics in LBM, we observe a simulation speedup of approximately 1000 times in terms of MLUPS compared to the serial implementation described by [80], which applied LBM to both thermal hydraulics and neutronics in a two-dimensional NTH tool.

8.5. GPU-Accelerated FM-LBM in MSFR Core Simulation Tools

As a final conclusion, it is yet to be seen whether the integration of a GPU-accelerated FM-LBM algorithm for thermal hydraulics simulation into a multi-physics simulation tool for MSFR core simulations is the right choice. The main motivation behind using lattice Boltzmann techniques over other numerical techniques for thermal hydraulics simulation is the possibility to parallelize its computations, which in turn can lead to computational gains. Up till this point, however, we have only simulated relatively simple, and small problem geometries. For these problem spaces, the FM-LBM algorithm reaches accurate results in an acceptable number of time steps. This means, GPU-implementations of the FM-LBM algorithm lead to relatively fast thermal hydraulics results within the complete simulation tool. For larger problem domains required in realistic MSFR core simulations, however, this advantage may diminish. Particularly in steady-state simulations, the use of large lattice grids leads to significant computational demands, with the algorithm requiring substantial time to converge to steady-state solutions. Even with GPU acceleration, such simulations may become impractical within acceptable timeframes. A potential solution could involve implementing a multi-GPU version of the FM-LBM algorithm. While this approach could establish the FM-LBM as a viable candidate for thermal-hydraulics simulations in MSFR core analysis, it would result in considerable costs for acquiring the necessary hardware.

Additionally, one of the major limitations of the FM-LBM framework is its implementation in complex geometries. As discussed in Section 7.3, incorporating complex boundary shapes into the Euclidean LBM grid is a challenging task. It requires precise documentation of physical boundary locations relative to lattice nodes, resulting in a framework that lacks flexibility. In contrast, mesh-based algorithms for thermal-hydraulics simulations, such as those described by Tiberga et al. [94], can easily use conformal meshes that align with the complex geometry shape of the reactor core.

In conclusion, further research is needed to evaluate the real advantage of LBM methods over other simulation techniques in terms of computation time in realistic MSFR geometries, and whether this is worth the investment over other simulation techniques such as FEM implementations that offer better compatibility with complex geometries.

8.6. Recommendations for Future Research

As discussed in Chapter 7, the simulation tool developed in this research meets only the minimal requirements for simulating the multiphysics phenomena within an MSFR reactor core. Therefore, we have already proposed several extensions necessary for advancing towards more realistic modeling of an actual MSFR. These include performing simulations in three-dimensional space, incorporating turbulence, and adapting the simulation to an MSFR core geometry by modeling a segment of a cylinder. Beyond these significant advancements, several smaller adjustments and extensions could further improve the simulation tool for MSFR reactor core simulations. These improvements are listed below.

- In this research, not all physical processes within an MSFR reactor core have been considered. For example, we have only included density feedback effects on the nuclear cross-sections, while ignoring Doppler shifts, which also influence the temperature dependence of nuclear cross-sections. Additionally, decay heat generated by fission products has not been accounted for. Incorporating these effects would enhance the realism of the multiphysics simulation tool, making it better suited for studying MSFR core behavior.
- When incorporating the heat exchanger and pump into the simulation geometry, further research should consider modeling phase transitions of the fuel salt within the LBM framework. Given the high melting point of the fuel salt, the molten salt mixture is prone to solidification during salt cooling. This behavior

is particularly relevant within the heat exchanger, where the fuel salt is actively cooled to extract energy from the system. Addressing this phenomenon would provide a more accurate representation of the MSFR reactor core.

- When implementing turbulence effects using LES-LBM techniques, future research should consider adding grid refinement strategies to the LBM algorithm. Turbulent eddies have smaller length scales near walls, so refining the grid in these regions would reduce the cutoff length, capturing more complex turbulence near boundaries. Additionally, grid refinement can improve accuracy in areas with steep temperature or precursor density gradients, which currently exhibit the highest relative errors in simulations.
- Finally, further efforts should focus on enhancing the computational performance of the algorithms to make simulations of three-dimensional geometries with fine grid spacing feasible. As previously suggested, a major computational gain can be achieved by integrating both the FM-LBM algorithm and Phantom-S_N into a single programming language, reducing the overhead associated with coupling. Beyond integration, optimizing the algorithms themselves should also be considered. For Phantom-S_N, this could involve implementing more parallelized code, such as concurrent computation of sweeps over angular ordinates or independent parts within a sweep. For the FM-LBM algorithm, a multi-GPU implementation could be explored, as it has been shown in the literature to significantly accelerate calculations. This improvement would be particularly beneficial for steady-state simulations in the power method, where the FM-LBM algorithm must converge in every iteration. For large problem domains, this requires simulating a significant number of LBM timesteps.

References

[1] (DHPC), Delft High Performance Computing Centre. *DelftBlue Supercomputer (Phase 2)*. `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2`. 2024.

[2] Agreement, Paris. "Paris agreement". In: *report of the conference of the parties to the United Nations framework convention on climate change (21st session, 2015: Paris). Retrived December*. Vol. 4. 2017. HeinOnline. 2015, p. 2.

[3] Aidun, C and Clausen, JR. "Lattice-Boltzmann method for complex flows". In: *Annual review of fluid mechanics* 42.1 (2010), pp. 439–472.

[4] Allibert, M et al. "Molten salt fast reactors". In: *Handbook of Generation IV Nuclear Reactors*. Elsevier, 2016, pp. 157–188.

[5] Alsmirat, M et al. "Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU-GPU implementations". In: *Multimedia Tools and Applications* 76 (2017), pp. 3537–3555.

[6] An, S, Yu, H, and Yao, J. "GPU-accelerated volumetric lattice Boltzmann method for porous media flow". In: *Journal of Petroleum Science and Engineering* 156 (2017), pp. 546–552.

[7] Aufiero, M et al. "Development of an OpenFOAM model for the Molten Salt Fast Reactor transient analysis". In: *Chemical Engineering Science* 111 (2014), pp. 390–401.

[8] Bhatnagar, P, Gross, E, and Krook, M. "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems". In: *Physical review* 94.3 (1954), p. 511.

[9] Blanco, JA, Rubiolo, P, and Dumonteil, E. "Neutronic modeling strategies for a liquid fuel transient calculation". In: *EPJ Web of Conferences*. Vol. 247. EDP Sciences. 2021, p. 06013.

[10] Börm, S and Mehl, C. *Numerical methods for eigenvalue problems*. Walter de Gruyter, 2012.

[11] Borstlap, L. "Developing a Filter-Matrix Lattice Boltzmann Boundary Method for Conjugate Heat Transfer and Phase Change". In: (2024).

[12] Bouzidi, M, Firdaouss, M, and Lallemand, P. "Momentum transfer of a Boltzmann-lattice fluid with boundaries". In: *Physics of fluids* 13.11 (2001), pp. 3452–3459.

[13] Cammi, A et al. "A multi-physics modelling approach to the dynamics of Molten Salt Reactors". In: *Annals of Nuclear Energy* 38.6 (2011), pp. 1356–1372.

[14] Cant, S. "SB Pope, Turbulent Flows, Cambridge University Press, Cambridge, UK, 2000, 771 pp." In: *Combustion and Flame* 125.4 (2001), pp. 1361–1362.

[15] Cervi, E et al. "Development of an SP3 neutron transport solver for the analysis of the Molten Salt Fast Reactor". In: *Nuclear Engineering and Design* 346 (2019), pp. 209–219.

[16] Chai, Z and Shi, B. "Multiple-relaxation-time lattice Boltzmann method for the Navier-Stokes and nonlinear convection-diffusion equations: Modeling, analysis, and elements". In: *Physical Review E* 102.2 (2020), p. 023306.

[17] Chang, Q and Yang, T. "A lattice Boltzmann method for image denoising". In: *IEEE Transactions on Image Processing* 18.12 (2009), pp. 2797–2802.

[18] Chapman, S and Cowling, TG. *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press, 1990.

[19] Chatterjee, D. "An enthalpy-based thermal lattice Boltzmann model for non-isothermal systems". In: *Europhysics Letters* 86.1 (2009), p. 14004.

[20] Chaudri, KS et al. "Coupled neutronics/thermal hydraulics evaluation for thorium based fuels in thermal spectrum SCWR". In: *Progress in Nuclear Energy* 68 (2013), pp. 55–64.

[21] Chen, L, Yu, Y, and Hou, G. "Sharp-interface immersed boundary lattice Boltzmann method with reduced spurious-pressure oscillations for moving boundaries". In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 87.5 (2013), p. 053306.

[22] Chen, S and Doolen, GD. "Lattice Boltzmann method for fluid flows". In: *Annual review of fluid mechanics* 30.1 (1998), pp. 329–364.

[23] De Boor, C. “A practical guide to splines”. In: *Springer-Verlag google schola* 2 (1978), pp. 4135–4195.

[24] Delbosc, N et al. “Optimized implementation of the Lattice Boltzmann Method on a graphics processing unit towards real-time fluid simulation”. In: *Computers & Mathematics with Applications* 67.2 (2014), pp. 462–475.

[25] Dematté, L and Prandi, D. “GPU computing for systems biology”. In: *Briefings in bioinformatics* 11.3 (2010), pp. 323–333.

[26] Dietz, T, Shwom, R, and Whitley, C. “Climate change and society”. In: *Annual Review of Sociology* 46.1 (2020), pp. 135–158.

[27] Documentation, CUDA Toolkit. In: (). URL: `https://docs.nvidia.com/cuda/`.

[28] Documentation, Fortran90. In: (). URL: `https://www.fortran90.org/`.

[29] Dolan, TJ. *Molten salt reactors and thorium energy*. Woodhead Publishing, 2017.

[30] Duderstadt, J and Hamilton, L. *Nuclear reactor analysis*. Wiley, 1976.

[31] Eberly, D. “B-Spline Interpolation on Lattices”. In: (2020).

[32] Eklund, A et al. “Medical image processing on the GPU–Past, present and future”. In: *Medical image analysis* 17.8 (2013), pp. 1073–1094.

[33] Ellis, M et al. “Preliminary coupling of the Monte Carlo code OpenMC and the multiphysics object-oriented simulation environment for analyzing doppler feedback in Monte Carlo simulations”. In: *Nuclear Science and Engineering* 185.1 (2017), pp. 184–193.

[34] Farrell, PE and Maddison, JR. “Conservative interpolation between volume meshes by local Galerkin projection”. In: *Computer Methods in Applied Mechanics and Engineering* 200.1-4 (2011), pp. 89–100.

[35] Ferziger, J, Perić, M, and Street, R. *Computational methods for fluid dynamics*. springer, 2019.

[36] Fiorina, C et al. “GeN-Foam: a novel OpenFOAM® based multi-physics solver for 2D/3D transient analysis of nuclear reactors”. In: *Nuclear Engineering and Design* 294 (2015), pp. 24–37.

[37] Garnier, E, Sagaut, P, and Deville, M. “Large eddy simulation of shock/boundary-layer interaction”. In: *AIAA journal* 40.10 (2002), pp. 1935–1944.

[38] Gerardin, D et al. “Design evolutions of the molten salt fast reactor”. In: (2017).

[39] Ghosal, S. “An analysis of numerical errors in large-eddy simulations of turbulence”. In: *Journal of Computational Physics* 125.1 (1996), pp. 187–206.

[40] Ginzbourg, I and d’Humières, D. “Local second-order boundary methods for lattice Boltzmann models”. In: *Journal of statistical physics* 84 (1996), pp. 927–971.

[41] Ginzburg, I, Verhaeghe, F, and d’Humieres, D. “Two-relaxation-time lattice Boltzmann scheme: About parametrization, velocity, pressure and mixed boundary conditions”. In: *Communications in computational physics* 3.2 (2008), pp. 427–478.

[42] Habermann, C and Kindermann, F. “Multidimensional spline interpolation: Theory and applications”. In: *Computational Economics* 30 (2007), pp. 153–169.

[43] Habich, J et al. “Performance analysis and optimization strategies for a D3Q19 lattice Boltzmann kernel on nVIDIA GPUs using CUDA”. In: *Advances in Engineering Software* 42.5 (2011), pp. 266–272.

[44] Habich, J et al. “Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results”. In: *Computers & Fluids* 80 (2013), pp. 276–282.

[45] Han, Y and Cundall, P. “Resolution sensitivity of momentum-exchange and immersed boundary methods for solid–fluid interaction in the lattice Boltzmann method”. In: *International Journal for Numerical Methods in Fluids* 67.3 (2011), pp. 314–327.

[46] Hargraves, R and Moir, R. “Liquid Fluoride Thorium Reactors: An old idea in nuclear power gets reexamined”. In: *American Scientist* 98.4 (2010), pp. 304–313.

[47] He, X, Chen, S, and Doolen, G. “A novel thermal model for the lattice Boltzmann method in incompressible limit”. In: *Journal of computational physics* 146.1 (1998), pp. 282–300.

[48] Hu, T et al. “Finite volume method based neutronics solvers for steady and transient-state analysis of nuclear reactors”. In: *Energy Procedia* 127 (2017), pp. 275–283.

[49] Huang, S, Xiao, S, and Feng, W. “On the energy efficiency of graphics processing units for scientific computing”. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2009, pp. 1–8.

[50] Jawerth, B, Lin, P, and Sinzinger, E. “Lattice Boltzmann models for anisotropic diffusion of images”. In: *Journal of Mathematical Imaging and Vision* 11 (1999), pp. 231–237.

[51] Kelly, JE. “Generation IV International Forum: A decade of progress through international cooperation”. In: *Progress in Nuclear Energy* 77 (2014), pp. 240–246.

[52] Kittisopikul, HA. “JuliaMath/Interpolations.jl: v0.14.7”. In: (2022).

[53] Koning, A et al. *The jeff-3.1 nuclear data library-jeff report 21*. Tech. rep. Organisation for Economic Co-operation and Development, 2006.

[54] Kophazi, J and Lathouwers, D. “Three-dimensional transport calculation of multiple alpha modes in subcritical systems”. In: *Annals of Nuclear Energy* 50 (2012), pp. 167–174.

[55] Krüger, T et al. “The lattice Boltzmann method”. In: *Springer International Publishing* 10.978-3 (2017), pp. 4–15.

[56] Language, Julia Documentation · The Julia. In: (). URL: `https://docs.julialang.org/en/v1/`.

[57] Lathouwers, D. “Goal-oriented spatial adaptivity for the SN equations on unstructured triangular meshes”. In: *Annals of Nuclear Energy* 38.6 (2011), pp. 1373–1381.

[58] LeBlanc, D. “Molten salt reactors: A new beginning for an old idea”. In: *Nuclear Engineering and design* 240.6 (2010), pp. 1644–1656.

[59] Lee, H et al. “IPCC, 2023: Climate Change 2023: Synthesis Report, Summary for Policymakers. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Core Writing Team, H. Lee and J. Romero (eds.)]. IPCC, Geneva, Switzerland.” In: (2023).

[60] Leppänen, J et al. “The Serpent Monte Carlo code: Status, development and applications in 2013”. In: *Annals of Nuclear Energy* 82 (2015), pp. 142–150.

[61] Lewis, E and Miller, W. “Computational methods of neutron transport”. In: (1984).

[62] Li, W, Wei, X, and Kaufman, A. “Implementing lattice Boltzmann computation on graphics hardware”. In: *The Visual Computer* 19 (2003), pp. 444–456.

[63] Madiajagan, M and Raj, S. “Parallel computing, graphics processing unit (GPU) and new hardware for deep learning in computational intelligence research”. In: *Deep learning and parallel computing environment for bioengineering systems*. Elsevier, 2019, pp. 1–15.

[64] Mandl, F. *Statistical physics*. Vol. 14. John Wiley & Sons, 1991.

[65] Mase, GT, Smelser, R, and Mase, GE. *Continuum mechanics for engineers*. CRC press, 2009.

[66] McCracken, ME and Abraham, J. “Multiple-relaxation-time lattice-Boltzmann model for multiphase flow”. In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 71.3 (2005), p. 036701.

[67] Membarth, R et al. “Generating device-specific GPU code for local operators in medical imaging”. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE. 2012, pp. 569–581.

[68] Modak, RS and Jain, VK. “Sub-space iteration scheme for the evaluation of λ-modes of finite-differenced multi-group neutron diffusion equations”. In: *Annals of Nuclear Energy* 23.3 (1996), pp. 229–237.

[69] Mohamad, AA and Succi, S. “A note on equilibrium boundary conditions in lattice Boltzmann fluid dynamic simulations”. In: *The European Physical Journal Special Topics* 171.1 (2009), pp. 213–221.

[70] Mylonakis, A, Varvayanni, M, and Catsaros, N. “A Newton-based Jacobian-free approach for neutronic-Monte Carlo/thermal-hydraulic static coupled analysis”. In: *Annals of Nuclear Energy* 110 (2017), pp. 709–725.

[71] Nellis, G and Klein, S. *Introduction to engineering heat transfer*. Cambridge University Press, 2020.

[72] Neumann, P et al. “A coupled approach for fluid dynamic problems using the PDE framework peano”. In: *Communications in Computational Physics* 12.1 (2012), pp. 65–84.

[73] Nicoud, F and Ducros, F. “Subgrid-scale stress modelling based on the square of the velocity gradient tensor”. In: *Flow, turbulence and Combustion* 62.3 (1999), pp. 183–200.

[74] Nieuwstadt, F, Boersma, B, and Westerweel, J. *Turbulence: Introduction to Theory and Applications of Turbulent Flows*. Springer International Publishing, 2016.

[75] Nieuwstadt, FT, Westerweel, J, and Boersma, B. *Introduction to theory and applications of turbulent flows*. Springer, 2016.

[76] NVIDIA. “NVIDIA A100 coress with 80 GB specs”. In: (2021). URL: `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf`.

[77] Pandey, M et al. “The transformational role of GPU computing and deep learning in drug discovery”. In: *Nature Machine Intelligence* 4.3 (2022), pp. 211–221.

[78] Papadrakakis, M, Stavroulakis, G, and Karatarakis, A. “A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures”. In: *Computer Methods in Applied Mechanics and Engineering* 200.13-16 (2011), pp. 1490–1508.

[79] Perkó, Z et al. “Adjoint-based sensitivity analysis of coupled criticality problems”. In: *Nuclear science and engineering* 173.2 (2013), pp. 118–138.

[80] Polderman, C. “Developing a multi-physics tool for the neutronics-thermal-hydraulics in a molten salt reactor using the lattice Boltzmann method”. In: (2024).

[81] Protocol, Kyoto. “Kyoto protocol”. In: *UNFCCC Website.* (1997), pp. 230–240. URL: `http://unfccc.int/kyoto%5C_protocol/items/2830.php`.

[82] Reddy, JN. *An introduction to continuum mechanics*. Cambridge university press, 2013.

[83] Rouch, H et al. “Preliminary thermal–hydraulic core design of the Molten Salt Fast Reactor (MSFR)”. In: *Annals of Nuclear Energy* 64 (2014), pp. 449–456.

[84] Serp, J et al. “The molten salt reactor (MSR) in generation IV: overview and perspectives”. In: *Progress in Nuclear Energy* 77 (2014), pp. 308–319.

[85] Smagorinsky, J. “General circulation experiments with the primitive equations: I. The basic experiment”. In: *Monthly weather review* 91.3 (1963), pp. 99–164.

[86] Somers, JA. “Direct simulation of fluid flow with cellular automata and the lattice-Boltzmann equation”. In: *Applied Scientific Research* 51 (1993), pp. 127–133.

[87] Steinkraus, D, Buck, I, and Simard, P. “Using GPUs for machine learning algorithms”. In: *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*. IEEE. 2005, pp. 1115–1120.

[88] Strack, OE and Cook, BK. “Three-dimensional immersed boundary conditions for moving solids in the lattice-Boltzmann method”. In: *International Journal for Numerical Methods in Fluids* 55.2 (2007), pp. 103–125.

[89] Succi, S. “Lattice Boltzmann method for quantum field theory”. In: *Journal of Physics A: Mathematical and Theoretical* 40.26 (2007), F559.

[90] Succi, S and Benzi, R. “Lattice Boltzmann equation for quantum mechanics”. In: *Physica D: Nonlinear Phenomena* 69.3 (1993), pp. 327–332. ISSN: 0167-2789. DOI: `https://doi.org/10.1016/0167-2789(93)90096-J`. URL: `https://www.sciencedirect.com/science/article/pii/016727899390096J`.

[91] Sudhakar, T and Das, A. “Evolution of multiphase lattice Boltzmann method: A review”. In: *Journal of The Institution of Engineers (India): Series C* 101.4 (2020), pp. 711–719.

[92] Tiberga, M. “Development of a high-fidelity multi-physics simulation tool for liquid-fuel fast nuclear reactors”. PhD thesis. Delft University of Technology Delft, The Netherlands, 2020.

[93] Tiberga, M, Lathouwers, D, and Kloosterman, JL. “A discontinuous Galerkin FEM multiphysics solver for the Molten Salt Fast Reactor”. In: *International Conference on Mathematics and Computational Methods applied to Nuclear Science and Engineering (M&C 2019), Portland, OR, USA*. 2019.

[94] Tiberga, M, Lathouwers, D, and Kloosterman, JL. “A multi-physics solver for liquid-fueled fast systems based on the discontinuous Galerkin FEM discretization”. In: *Progress in Nuclear Energy* 127 (2020), p. 103427.

[95] Tiberga, M et al. “Preliminary investigation on the melting behavior of a freeze-valve for the Molten Salt Fast Reactor”. In: *Annals of Nuclear Energy* 132 (2019), pp. 544–554.

[96] Tiberga, M et al. “Results from a multi-physics numerical benchmark for codes dedicated to molten salt fast reactors”. In: *Annals of Nuclear Energy* 142 (2020), p. 107428.

[97] Tolke, J. “Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA”. In: *computing and Visualization in Science* 13.1 (2010), p. 29.

[98] Tran, NP, Lee, M, and Choi, DH. “Memory-efficient parallelization of 3D lattice Boltzmann flow solver on a GPU”. In: *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. IEEE. 2015, pp. 315–324.

[99] Tran, NP, Lee, M, and Hong, S. "Performance optimization of 3D lattice Boltzmann flow solver on a GPU". In: *Scientific Programming* 2017.1 (2017), p. 1205892.

[100] Tritton, DJ. *Physical fluid dynamics*. Springer Science & Business Media, 2012.

[101] Unknown. In: (). URL: `https://en.wikipedia.org/wiki/Bicubic_interpolation`.

[102] Vahl Davis, G de. "Natural convection of air in a square cavity: a bench mark numerical solution". In: *International Journal for numerical methods in fluids* 3.3 (1983), pp. 249–264.

[103] Valero-Lara, P. "Reducing memory requirements for large size LBM simulations on GPUs". In: *Concurrency and Computation: Practice and Experience* 29.24 (2017), e4221.

[104] Vazquez, M et al. "Coupled neutronics thermal-hydraulics analysis using Monte Carlo and sub-channel codes". In: *Nuclear Engineering and Design* 250 (2012), pp. 403–411.

[105] Verdú, G et al. "3D λ-modes of the neutron-diffusion equation". In: *Annals of Nuclear Energy* 21.7 (1994), pp. 405–421.

[106] Vreman, AW. "An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications". In: *Physics of fluids* 16.10 (2004), pp. 3670–3681.

[107] Vreman, B, Geurts, B, and Kuerten, H. "COMPARISION of numerical schemes in large-eddy simulation of the temporal mixing layer". In: *International journal for numerical methods in fluids* 22.4 (1996), pp. 297–311.

[108] Waltar, A, Todd, D, and Tsvetkov, P. *Fast spectrum reactors*. Springer, 2011.

[109] Wang, S et al. "A passive decay heat removal system for emergency draining tanks of molten salt reactors". In: *Nuclear Engineering and Design* 341 (2019), pp. 423–431.

[110] Wang, Y and Ma, Y. "lbmNTH: A unified lattice Boltzmann framework for coupled neutronics-thermal-hydraulics analysis". In: *Annals of Nuclear Energy* 166 (2022), p. 108750.

[111] Wang, Y, Ma, Y, and Xie, M. "GPU accelerated lattice Boltzmann method in neutron kinetics problems". In: *Annals of Nuclear Energy* 129 (2019), pp. 350–365.

[112] Wang, Y, Ma, Y, and Xie, M. "High-order lattice Boltzmann method for multi-group neutron diffusion solution". In: *Progress in Nuclear Energy* 110 (2019), pp. 341–353.

[113] Weart, S. *The discovery of global warming: Revised and expanded edition*. Harvard University Press, 2008.

[114] website, Generation IV International Forum (GIF). In: (). URL: `https://www.gen-4.org/gif/jcms/c9334/origins`.

[115] Weickert, M et al. "Investigation of the LES WALE turbulence model within the lattice Boltzmann framework". In: *Computers & Mathematics with Applications* 59.7 (2010), pp. 2200–2214.

[116] Xu, A and Li, BT. "Particle-resolved thermal lattice Boltzmann simulation using OpenACC on multi-GPUs". In: *International Journal of Heat and Mass Transfer* 218 (2024), p. 124758.

[117] Yoshida, H and Nagaoka, M. "Lattice Boltzmann method for the convection–diffusion equation in curvilinear coordinate systems". In: *Journal of Computational Physics* 257 (2014), pp. 884–900.

[118] Younes, W and Loveland, W. *An introduction to nuclear fission*. Springer, 2021.

[119] Zhang, N and Liu, D. "LBM modeling of three-dimensional mixed convection in CZ crystal growth on curvilinear coordinates system". In: *Results in Physics* 61 (2024), p. 107754.

[120] Zhang, Y et al. "Application of large eddy simulation models to electroconvection turbulence study with lattice Boltzmann method". In: *Physical Review Fluids* 9.8 (2024), p. 083703.

[121] Zhuo, C and Zhong, C. "LES-based filter-matrix lattice Boltzmann model for simulating fully developed turbulent channel flow". In: *International Journal of Computational Fluid Dynamics* 30.7-10 (2016), pp. 543–553.

[122] Zhuo, C and Zhong, C. "LES-based filter-matrix lattice Boltzmann model for simulating turbulent natural convection in a square cavity". In: *International Journal of Heat and Fluid Flow* 42 (2013), pp. 10–22.

[123] Zhuo, C, Zhong, C, and Cao, J. "Filter-matrix lattice Boltzmann model for incompressible thermal flows". In: *Physical Review E* 85.4 (2012), p. 046703.

[124] Zou, Q and He, X. "On pressure and velocity boundary conditions for the lattice Boltzmann BGK model". In: *Physics of fluids* 9.6 (1997), pp. 1591–1598.

A

Neutronics Data From The Tiberga Benchmark Case

Table A.1: Average total number of neutrons emitted per fission event per group, prompt and delayed fission spectra, average energy emitted per fission event per group, and inverse neutron group velocities.

Group, g	$\nu_{tot,g}$	$\chi_{p,g}$	$\chi_{d,g}$	E_{fiss} (J)	$1/\mathrm{v}_g$(sec cm^{-1})
1	2.85517	$3.53812 \cdot 10^{-1}$	$4.30325 \cdot 10^{-3}$	$3.240722 \cdot 10^{-11}$	$4.00367 \cdot 10^{-10}$
2	2.54532	$5.23642 \cdot 10^{-1}$	$3.87734 \cdot 10^{-1}$	$3.240722 \cdot 10^{-11}$	$7.39846 \cdot 10^{-10}$
3	2.43328	$1.21033 \cdot 10^{-1}$	$5.81848 \cdot 10^{-1}$	$3.240722 \cdot 10^{-11}$	$2.61748 \cdot 10^{-9}$
4	2.43127	$1.35457 \cdot 10^{-3}$	$2.27947 \cdot 10^{-2}$	$3.240722 \cdot 10^{-11}$	$6.69270 \cdot 10^{-9}$
5	2.43330	$1.51226 \cdot 10^{-4}$	$2.89130 \cdot 10^{-3}$	$3.240722 \cdot 10^{-11}$	$1.55845 \cdot 10^{-8}$
6	2.43330	$7.37236 \cdot 10^{-6}$	$4.28935 \cdot 10^{-4}$	$3.240722 \cdot 10^{-11}$	$4.24462 \cdot 10^{-8}$

Table A.2: Total and fission cross-sections per energy group.

Group, g	$\Sigma_{t,g}$(cm^{-1})	$\Sigma_{f,g}$(cm^{-1})
1	$1.65512 \cdot 10^{-1}$	$1.11309 \cdot 10^{-3}$
2	$2.17253 \cdot 10^{-1}$	$1.08682 \cdot 10^{-3}$
3	$3.18009 \cdot 10^{-1}$	$1.52219 \cdot 10^{-3}$
4	$2.42093 \cdot 10^{-1}$	$2.58190 \cdot 10^{-3}$
5	$2.50351 \cdot 10^{-1}$	$5.36326 \cdot 10^{-3}$
6	$2.72159 \cdot 10^{-1}$	$1.44917 \cdot 10^{-2}$

Table A.3: P_0 scattering cross sections.

	$\Sigma_{s,0,g'\rightarrow g}$(cm^{-1})					
	Group g					
Group, g'	1	2	3	4	5	6
1	$1.08476 \cdot 10^{-1}$	$5.23316 \cdot 10^{-2}$	$4.01805 \cdot 10^{-3}$	$1.09869 \cdot 10^{-4}$	$2.53290 \cdot 10^{-5}$	$3.78334 \cdot 10^{-6}$
2	0	$1.83666 \cdot 10^{-1}$	$3.19138 \cdot 10^{-2}$	$2.34218 \cdot 10^{-5}$	$2.25259 \cdot 10^{-6}$	$2.00405 \cdot 10^{-7}$
3	0	0	$2.98293 \cdot 10^{-1}$	$1.63470 \cdot 10^{-2}$	$1.70575 \cdot 10^{-5}$	$1.24625 \cdot 10^{-6}$
4	0	0	0	$2.17472 \cdot 10^{-1}$	$1.90243 \cdot 10^{-2}$	$1.36858 \cdot 10^{-8}$
5	0	0	0	0	$2.27173 \cdot 10^{-1}$	$1.05885 \cdot 10^{-2}$
6	0	0	0	0	0	$2.37826 \cdot 10^{-1}$

Table A.4: P_1 scattering cross sections.

	$\Sigma_{s,1,g'\rightarrow g}$(cm^{-1})					
	Group g					
Group, g'	1	2	3	4	5	6
1	$5.02479 \cdot 10^{-2}$	$5.31822 \cdot 10^{-3}$	$4.77967 \cdot 10^{-4}$	$3.40533 \cdot 10^{-5}$	$8.93761 \cdot 10^{-6}$	$1.34789 \cdot 10^{-6}$
2	0	$3.78784 \cdot 10^{-2}$	$6.77085 \cdot 10^{-3}$	$1.92591 \cdot 10^{-6}$	$6.99121 \cdot 10^{-8}$	$1.59216 \cdot 10^{-8}$
3	0	0	$2.42287 \cdot 10^{-2}$	$5.13491 \cdot 10^{-3}$	$8.27643 \cdot 10^{-7}$	$3.25237 \cdot 10^{-8}$
4	0	0	0	$1.75032 \cdot 10^{-2}$	$5.64363 \cdot 10^{-3}$	$5.05232 \cdot 10^{-10}$
5	0	0	0	0	$1.49908 \cdot 10^{-2}$	$3.22691 \cdot 10^{-3}$
6	0	0	0	0	0	$1.17041 \cdot 10^{-2}$

Table A.5: P_2 scattering cross sections.

	$\Sigma_{s,2,g'\rightarrow g}$(cm^{-1})					
	Group g					
Group, g'	1	2	3	4	5	6
1	$2.92625 \cdot 10^{-2}$	$1.42924 \cdot 10^{-3}$	$8.34387 \cdot 10^{-5}$	$1.81137 \cdot 10^{-5}$	$4.60830 \cdot 10^{-6}$	$7.72548 \cdot 10^{-7}$
2	0	$1.34479 \cdot 10^{-2}$	$1.08011 \cdot 10^{-4}$	$1.22275 \cdot 10^{-7}$	$1.36245 \cdot 10^{-8}$	$5.37827 \cdot 10^{-9}$
3	0	0	$3.26562 \cdot 10^{-3}$	$2.07083 \cdot 10^{-4}$	$2.24708 \cdot 10^{-7}$	$5.79136 \cdot 10^{-8}$
4	0	0	0	$7.74424 \cdot 10^{-4}$	$4.07402 \cdot 10^{-4}$	$2.06625 \cdot 10^{-9}$
5	0	0	0	0	$5.04809 \cdot 10^{-4}$	$1.82238 \cdot 10^{-4}$
6	0	0	0	0	0	$3.21414 \cdot 10^{-4}$

Table A.6: P_3 scattering cross sections.

	$\Sigma_{s,3,g'\rightarrow g}$(cm^{-1})					
	Group g					
Group, g'	1	2	3	4	5	6
1	$1.43002 \cdot 10^{-2}$	$1.47138 \cdot 10^{-3}$	$1.41136 \cdot 10^{-5}$	$4.86683 \cdot 10^{-6}$	$1.18151 \cdot 10^{-6}$	$1.98042 \cdot 10^{-7}$
2	0	$2.82616 \cdot 10^{-3}$	$1.27263 \cdot 10^{-4}$	$2.35861 \cdot 10^{-8}$	$2.87008 \cdot 10^{-10}$	$7.05060 \cdot 10^{-9}$
3	0	0	$2.81143 \cdot 10^{-4}$	$1.40428 \cdot 10^{-4}$	$4.64989 \cdot 10^{-8}$	$6.27160 \cdot 10^{-8}$
4	0	0	0	$1.23373 \cdot 10^{-5}$	$8.90726 \cdot 10^{-6}$	$3.97151 \cdot 10^{-10}$
5	0	0	0	0	$3.24220 \cdot 10^{-6}$	$6.27731 \cdot 10^{-7}$
6	0	0	0	0	0	$8.75552 \cdot 10^{-6}$

B
Simulated Observables for Step 1.4 of The Tiberga Benchmark

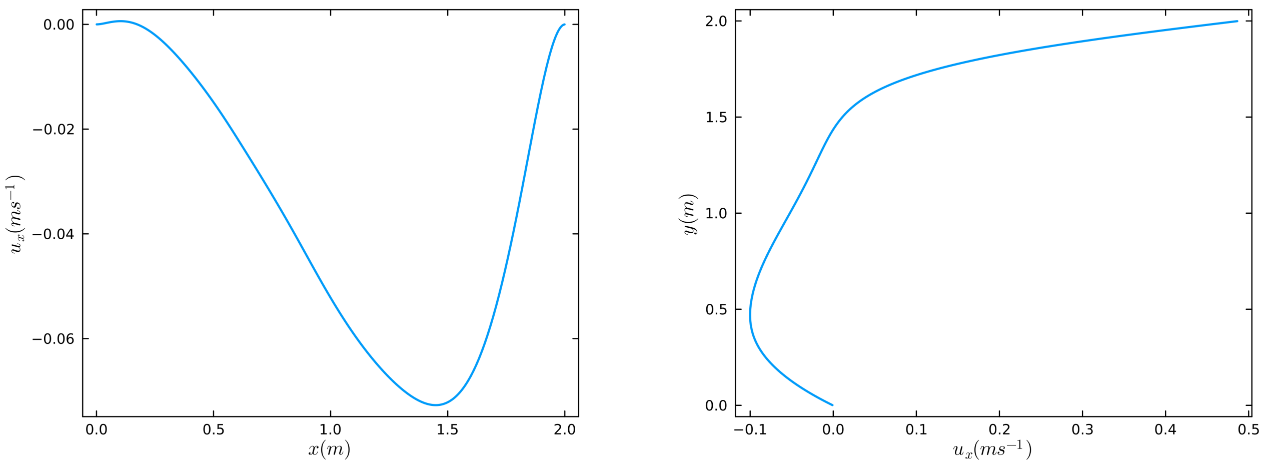

Figure B.1: Simulated horizontal velocity component along the horizontal (left) and vertical (right) centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500.

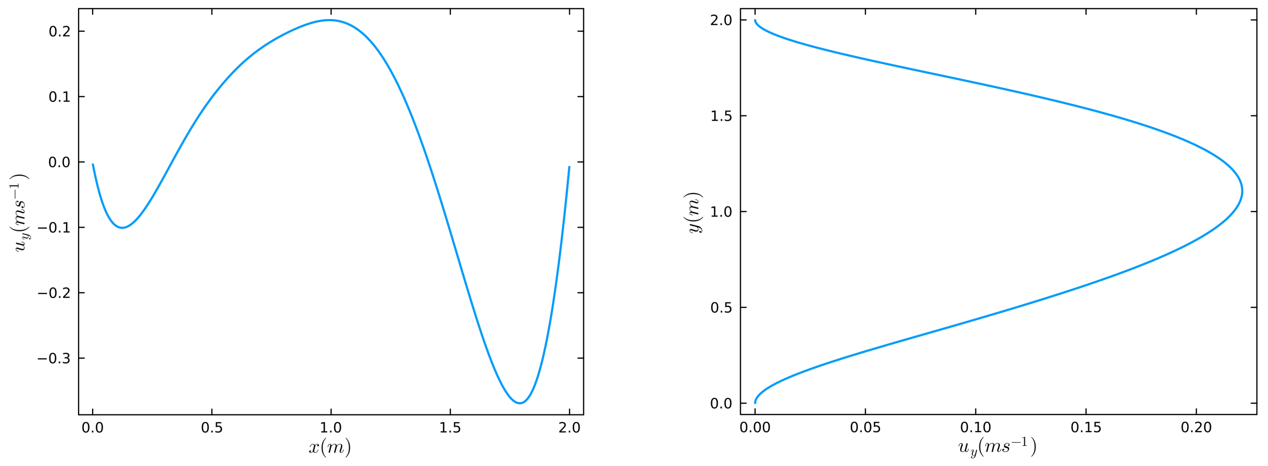

Figure B.2: Simulated vertical velocity component along the horizontal (left) and vertical (right) centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500.

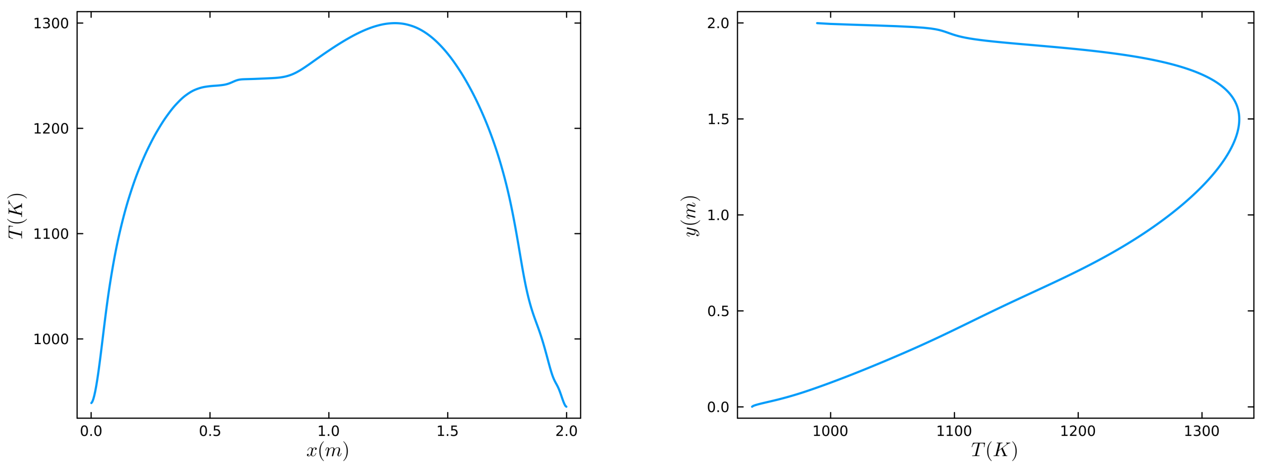

Figure B.3: Simulated temperature field along the horizontal (left) and vertical (right) centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500.

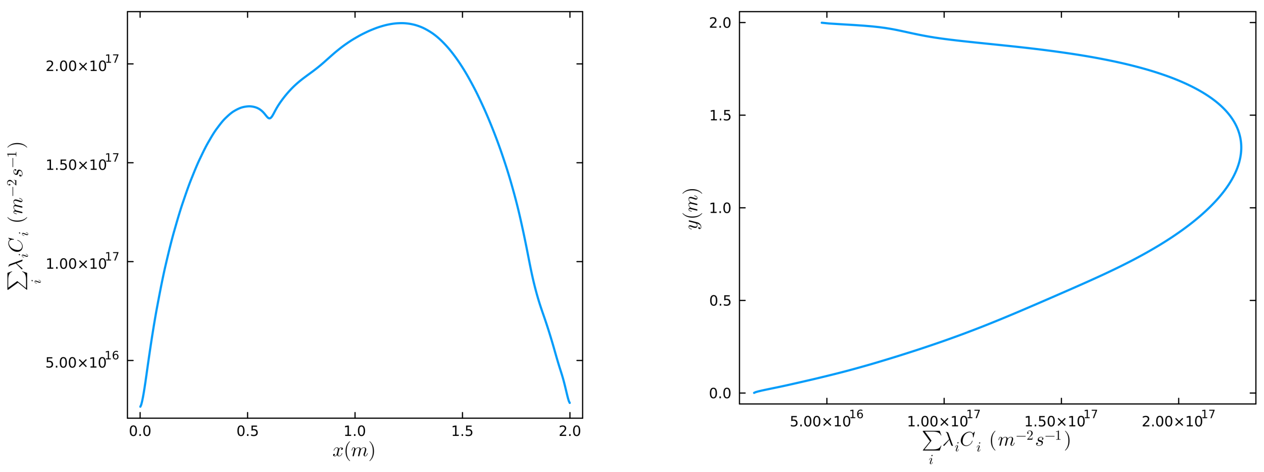

Figure B.4: Simulated delayed neutron source along the horizontal (left) and vertical (right) centerlines in step 1.4 of the Tiberga benchmark case for a simulation grid of 501×501 with a Prandtl number of 1000 and Schmidt number 1500.

C
Snapshots of Transient Simulation Results

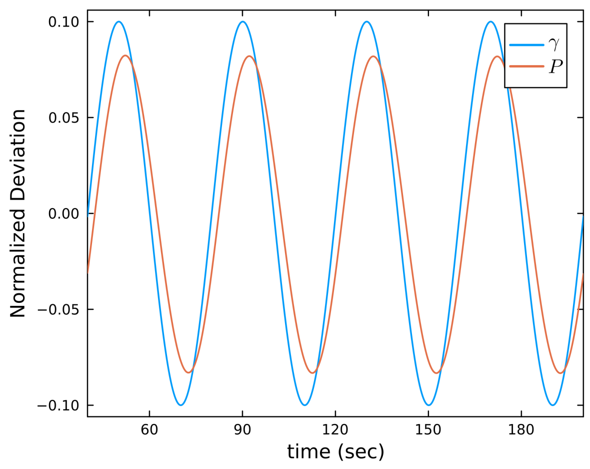

Figure C.1: Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.025$.

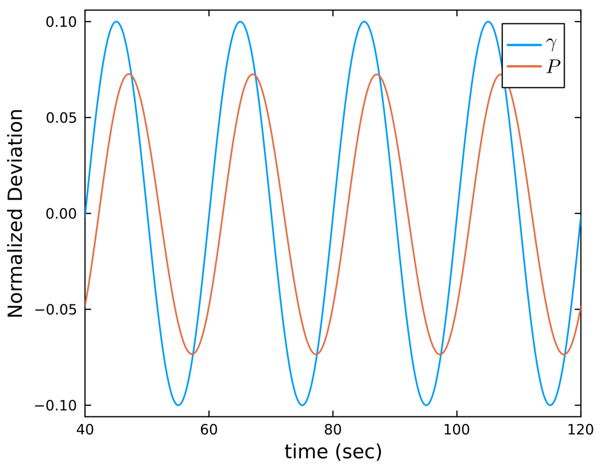

Figure C.2: Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.05$.

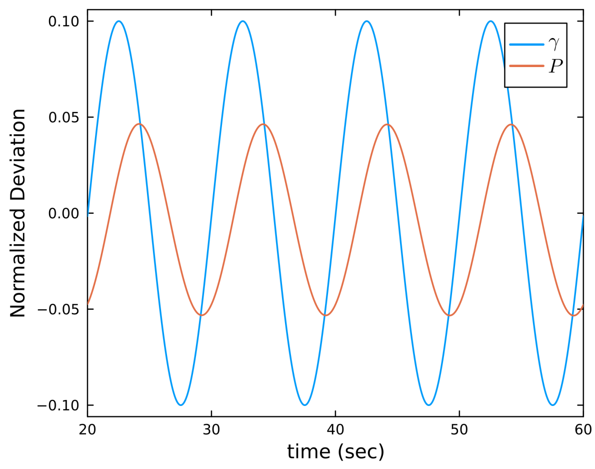

Figure C.3: Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.1$.

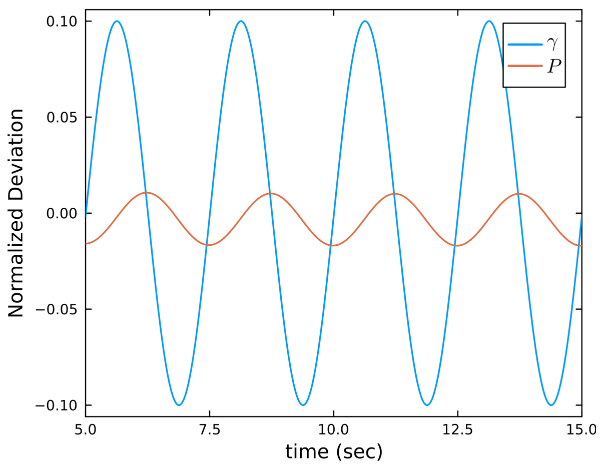

Figure C.4: Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.4$.

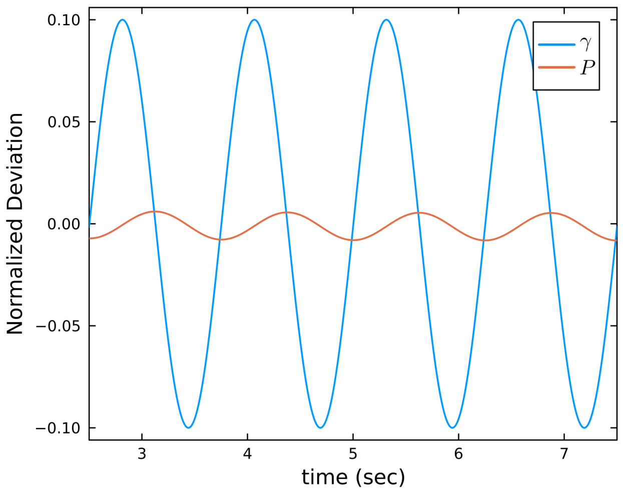

Figure C.5: Power response to the perturbation of the heat transfer coefficient in the frequency domain with $f_{pert} = 0.8$.

D
1D Cross-Section Plots of 3D Simulation Results

In this appendix we illustrate one-dimensional plots of the simulated variables from the 3D simulations presented in Section 7.1. The figures illustrate the three velocity components, u_x, u_y, u_z, the temperature field, T, and the delayed neutron source $\sum_i \lambda_i C_i$ along the x-, y- and z- cubic centerlines for both the pure-buoyancy-driven flow from Section 7.1.1 and the buoyancy-driven flow with shear momentum source from Section 7.1.2. The x-centerline means that the other coordinates are fixed at a value $y, z = L/2$.

D.1. Pure Buoyancy-Driven Flow

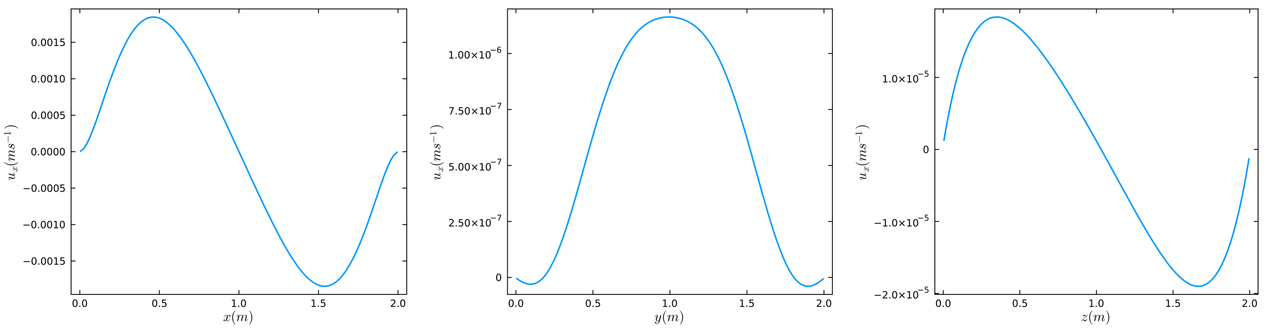

Figure D.1: u_x along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow

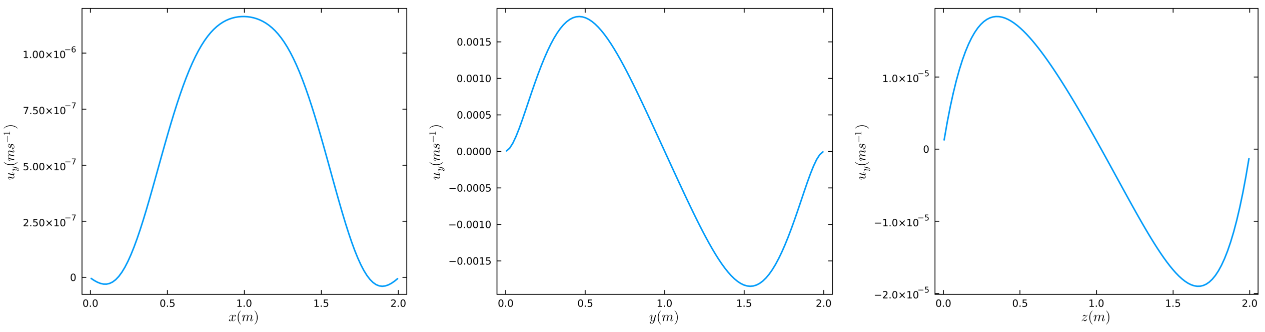

Figure D.2: u_y along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow

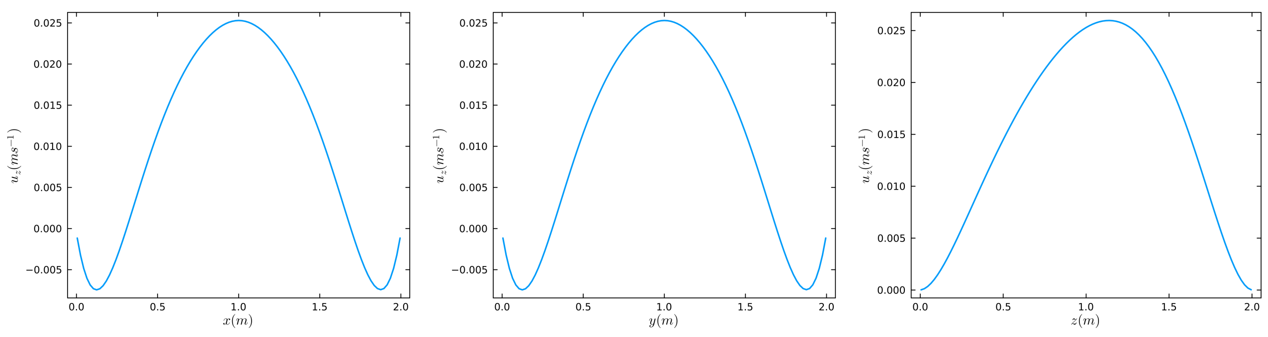

Figure D.3: u_z along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow

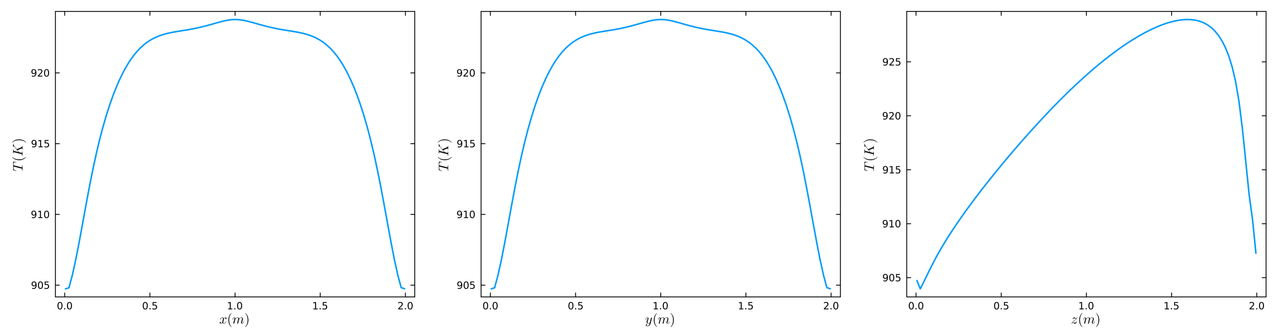

Figure D.4: Temperature field, T, along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow

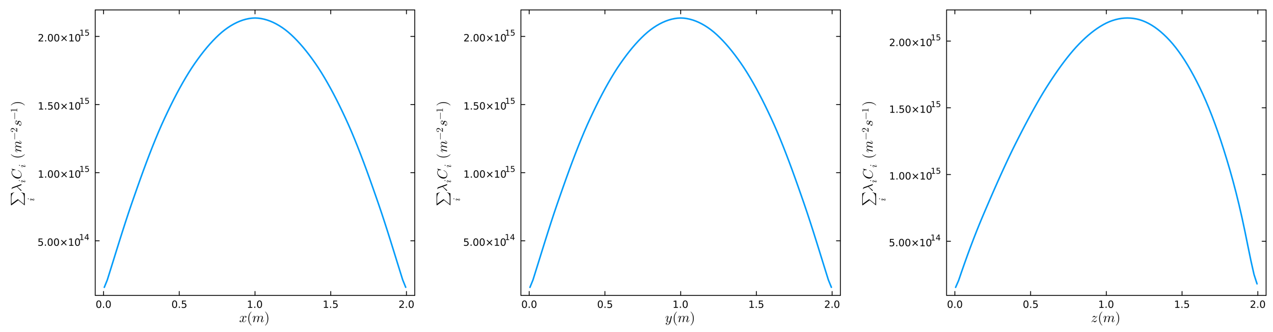

Figure D.5: Delayed neutron source, $\sum_i \lambda_i C_i$, along x-, y- and z- cubic centerlines for the pure buoyancy-driven flow

D.2. Buoyancy-Driven Flow With Shear Momentum Source

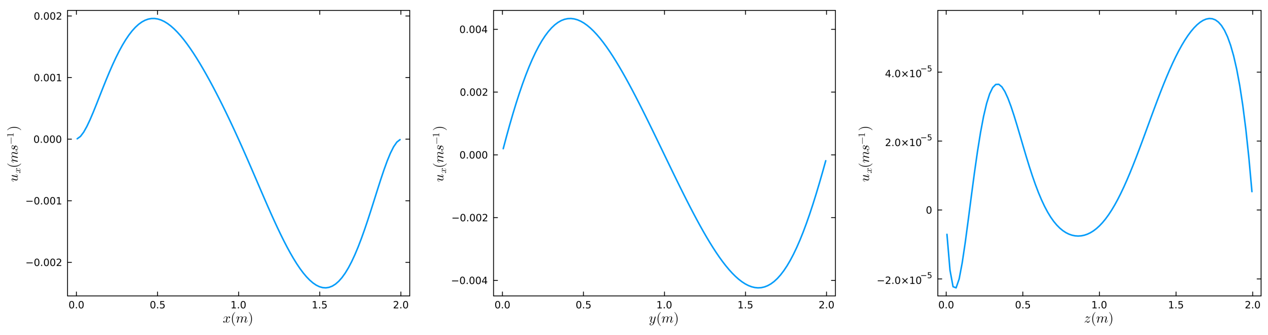

Figure D.6: u_x along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source

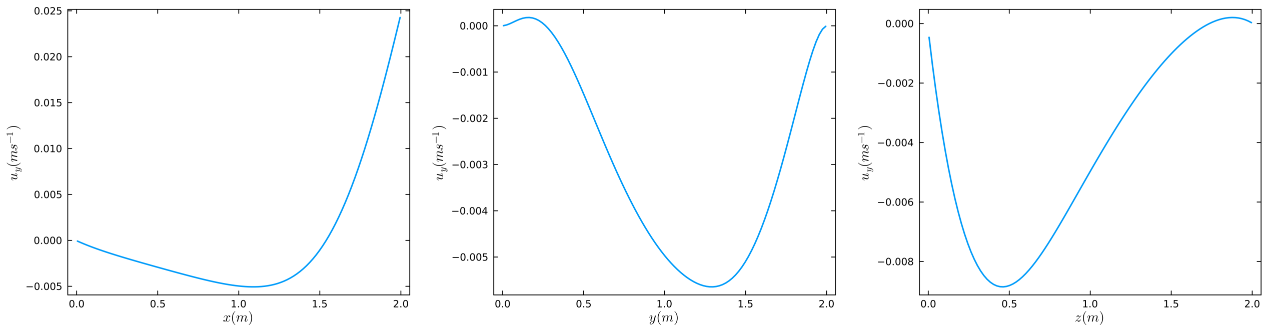

Figure D.7: u_y along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source

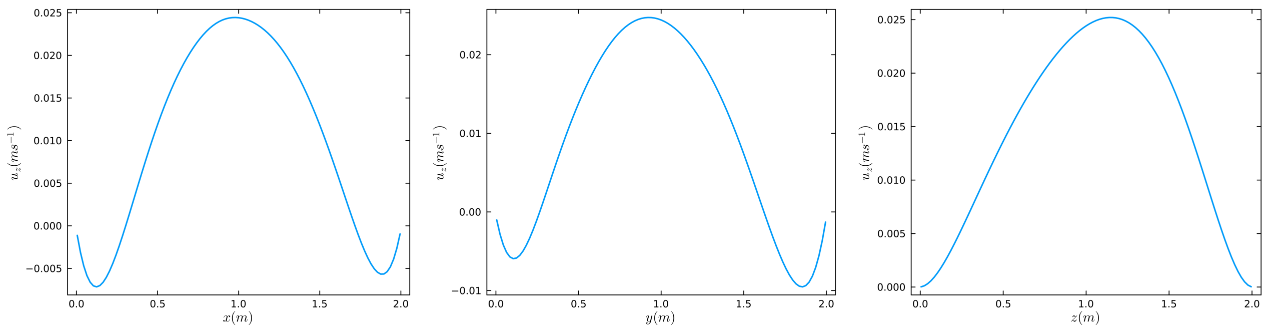

Figure D.8: u_z along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source

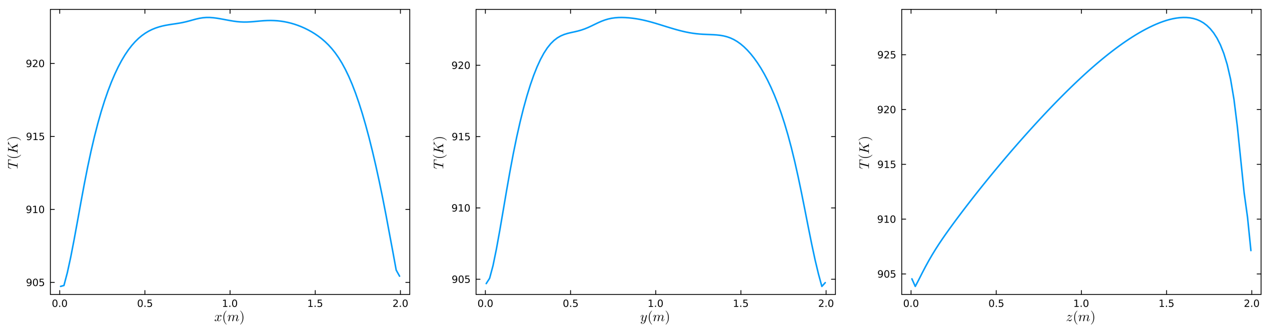

Figure D.9: Temperature field, T, along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source

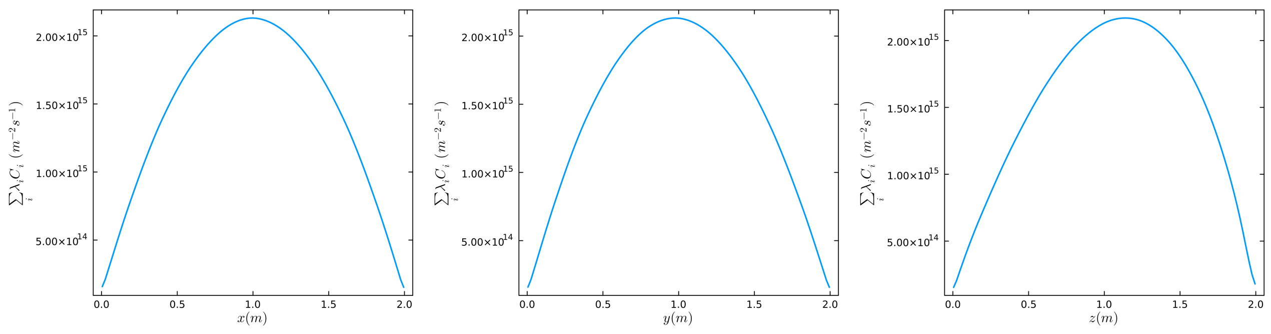

Figure D.10: Delayed neutron source, $\sum_i \lambda_i C_i$, along x-, y- and z- cubic centerlines for the buoyancy-driven flow with shear momentum source