POLITECNICO DI TORINO

PETROLEUM AND MINING ENGINEERING

Master Thesis in Petroleum Engineering

NON-ISOTHERMAL FLUID FLOW RESERVOIR SIMULATION USING DUMUX SOFTWARE

Tutors:

Prof. Dario Viberti

Dr. Eloisa Salina Borello

Candidate:

Mahmoud Aboelseoud

July 2021

ACKNOWLEDGEMENTS

I was actually lucky to have the chance to come and study at Politecnico di Torino for my master's degree. It is definitely a privilege to hold a degree from such a prestigious university. I would like to thank my company (Belayim Petroleum Company (Petrobel)) in Egypt for nominating me for the ENI scholarship which allowed me this great opportunity.

This work truly would have never been possible without the incredible effort and the highly appreciated guidance by my cooperative and supportive supervisors: Professor Dario Viberti and Dr. Eloisa Salina Borello. Their contribution and advice along the way have been critical for the completion of this thesis.

I am also really grateful to the members of the Dumux mailing list who have been of great assistance by responding to my questions and inquiries.

Finally, I feel the necessity for acknowledging my family for providing me with the proper study environment despite the psychological stresses of the COVID-19 pandemic. Last, but not least, I thank my father who is not here anymore for everything he has done for me. I dedicate this work to his soul.

ABSTRACT

Geothermal energy has received global attention for being a clean, sustainable and cheap energy source. Numerical modeling is currently an integral part of all the stages of geothermal processes such as exploration and development. In this 3D simulation study, a low-temperature geothermal doublet has been considered. In such system, hot water is produced from one well (producer) and cooled water whose heat content has been depleted by a surface heat exchanger is re-injected underground through another well (injector).

This work tests the practicality of employing the Finite-Volume based Dumux simulator for modeling the development of the cooled-water thermal front over the years. Dumux is a relatively new multipurpose open-source simulator that is based on the C++ programming language. The Dumux simulation outcome has been validated by a comparison against that of the Finite-Difference based commercial reservoir simulator ECLIPSE. The computational performances of both simulators have also been analyzed.

Two major scenarios were adopted: a convection-dominated scenario in which only the geothermal aquifer layer was modeled and a convection plus conduction scenario in which the caprock and bedrock layers were also modeled. The inclusion of the caprock and bedrock layers into the model geometry accounts for the added thermal conduction effects across those layers. The outlining of two non-isothermal scenarios was meant to evaluate whether the added computational cost of modeling the caprock and bedrock layers was justified.

The consistency of the Dumux solution for a varying maximum allowed time step size was investigated. Furthermore, the sensitivity of the simulation variables to different injection and production rate schemes was examined. The studied simulation variables are pressure and temperature as primary variables in addition to water viscosity as a secondary variable. Both temporal and spatial variations of variables were inspected according to the modeled scenario.

LIST OF CONTENTS

1 INTRODUCTION

Non-isothermal fluid flow characterizes geothermal aquifers which are used for energy extraction through heat exchange from the produced geothermal fluid. Geothermal energy originates from the Earth's interior where heat flows towards the surface and is retained by sub-surface aquifers (Limberger et al., 2018). Geothermal energy is a sustainable source of energy that is free of carbon and has a reduced environmental impact (Beaude et al., 2019). Another main advantage of geothermal energy is its low cost which grants geothermal systems the capacity to eventually be the world's cheapest source of clean thermal fuel (Goldstein et al., 2013). Geothermal heat was first successfully exploited to generate electricity used for lighting purposes in Tuscany, Italy (Limberger et al., 2018). Geothermal systems may be classified into high-temperature (higher than 150 °C), intermediate-temperature (from 90 °C to 150 °C) and low-temperature (from 25 °C to less than 90 °C) resources (Dai & Chen, 2008). Low-temperature geothermal systems are mostly used for direct applications such as district heating and fish farming (Ouali et al., 2015). In this numerical study, a low-temperature 3D geothermal doublet has been considered. A geothermal doublet is a geothermal system consisting of two wells (producer and injector) where hot water is produced from one well (producer) and cooled water is re-injected underground through the other well (injector) after transferring its energy via a surface heat exchanger (Mahbaz et al., 2021). A 3D schematic from Ganguly et al., (2017) for the geothermal doublet system can be seen in Figure 1.1 The injected water has a double benefit of maintaining the reservoir pressure and depleting the heat of the reservoir rock as the cooled front propagates through the reservoir and thus not only the heat content of the geothermal fluid is extracted but also that of the reservoir rock (Elemér, 2014).

Numerical simulation has become crucial for all the stages of geothermal processes such as the exploration stage for evaluating the geothermal potential and the development phase for optimizing the use of the geothermal resource (Beaude et al., 2019). Dumux is a relatively new Finite-Volume and Finite-Element based simulator that is intended for the simulation of multi-phase or multi-component or multi-physics or even multi-domain flow and transport processes in porous media. Dumux is a free and open-source software developed by the University of Stuttgart since January 2007 for research purposes (Koch, Gläser, et al., 2020). It is based on the widely used C++ programming language which provides flexibility to implement different kinds of Equations of State, constitutive equations, boundary conditions, etc. Furthermore, this flexibility also allows the Dumux users to implement new features such as modified physical or chemical behaviors. Source code adjustments can be shared within the Dumux community. It should be highlighted here that Dumux is a simulator not focalized on oil and gas reservoir problems like the industrial simulators like ECLIPSE (by Schlumberger) but instead it can be regarded as a multipurpose simulator that can be used for numerical modeling within various fields of application. For instance, Dumux can be used for modeling natural phenomena such as soil water evaporation due to solar radiation (Heck et al., 2020) or environmental problems such as disposal of

radioactive waste (Ahusborde et al., 2015) or biomedical engineering problems such as brain tissue perfusion (Koch, Flemisch, et al., 2020) and, of course, modeling oil and gas reservoir engineering problems such as simulation of oil production (Koch, Gläser, et al., 2020). Dumux has also been applied to geothermal applications like modeling the temperature distribution resulting from the injection-extraction operation for a confined layered geothermal reservoir (Ganguly et al., 2017a) and analysis of the impact of heat dissipated from a geothermal aquifer on temperature distribution (Ganguly et al., 2017b).

This work investigates the feasibility of using the Finite-Volume based research simulator Dumux to simulate the thermal front development of a low-temperature geothermal doublet over the years. To this end, comparison with the results and computational performance of ECLIPSE are also provided. The ECLIPSE software, commercialized by Schlumberger company, is one of the most widely used commercial reservoir simulators within the Oil & Gas industry. ECLIPSE is a Finite-Difference based simulator. It has a package intended for black oil simulation (ECLIPSE 100) and another package intended for compositional simulation (ECLIPSE 300). The package employed in this work is the ECLIPSE 100 and will always be referred to throughout the whole work as "ECLIPSE". Since Dumux is a multipurpose open-source code, many features available in ECLIPSE have to be manually coded in the C++ language to make the modeled scenarios in Dumux and ECLIPSE consistent with each other. This was the main challenge of the work.

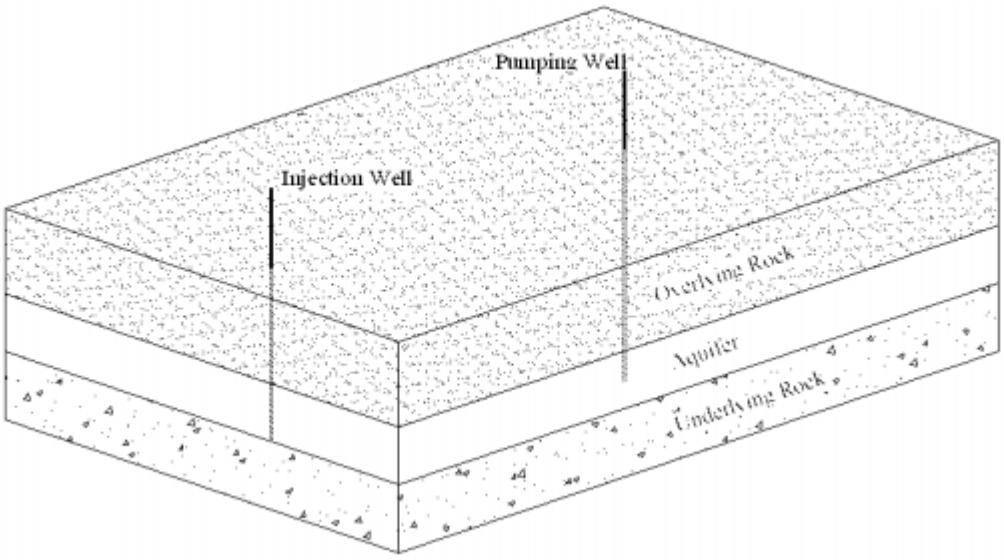


Figure 1.1: 3D schematic of a geothermal doublet system (Ganguly et al., 2017)

Two main scenarios were considered: in scenario A, only the geothermal aquifer layer was modeled and thus heat transfer occurred mainly by convection; in scenario B, caprock and bedrock layers were also modeled to account for the additional effect of heat transport by solid conduction through those layers. The definition of two non-isothermal scenarios was mainly aimed at understanding whether accounting for the

effect of conduction is worth the additional computational cost due to the larger number of cells used in the corresponding numerical model. Another reason is to inspect whether the comparison between Dumux and ECLIPSE may show some pronounced differences when modeling the convection-dominated scenario with respect to convection plus conduction through caprock and bedrock. For both scenarios, we investigated the variation of pressure with time at both the injection and production wells and the spatial variation of pressure, temperature and water viscosity along certain cross-sections in the computational domain. As for scenario B, we additionally investigated the temporal variation of temperature in the boundary cell above the injection well to check the temperature behavior due to pure conduction.

This thesis is comprised of a total of seven chapters. Chapter 2 is a problem formulation chapter that addresses the mathematical equations which depict the physics of the problem in addition to the imposed initial and boundary conditions necessary to render the problem well-posed. Chapter 3 details the characteristics of the computational domain such as the domain dimensions, well locations and the applied meshing in addition to the fluid and solid properties. Chapter 4 gives a general overview of the Dumux simulator with regard to its development stages, the applications it has been applied to, benchmark studies and comparisons against other simulators it has been involved in, fundamental files that form any Dumux problem as well as the available models, discretization schemes and distinguishing features. Chapter 5 illustrates how the main aspects of the problem are implemented in the Dumux code such as the selection of the model and the discretization schemes while elaborating the corresponding ECLIPSE implementation. The simulation results are presented and discussed in chapter 6 which also provides the comparison between the results and the involved computational cost of Dumux against those of ECLIPSE. Finally, the main conclusions of the work are drawn in chapter 7.

2 PROBLEM FORMULATION

Our problem consists of two main physical phenomena: single-phase liquid water flow in a porous medium within the Darcy domain and heat transport through the porous medium by convection and conduction. To make our problem well-posed, initial and boundary conditions had to be assigned to each of our two scenarios: scenario A and scenario B. For each of the two scenarios, the same initial conditions were assumed for the geothermal aquifer layer but different boundary conditions were used due to differences in the model geometry of the two scenarios.

2.1 Single-Phase Liquid Water Flow Subproblem

The first governing equation for this subproblem is a mass balance/conservation equation that can be expressed as follows:

$$-\nabla \cdot (\rho_w u) = \frac{\partial(\rho_w \phi)}{\partial t} - q \tag{2.1}$$

where ρ_w is the water density at reservoir conditions in Kg/m_{rc}^3, u is the Darcy velocity in m/s, ϕ is the porosity (unitless) and q is the source or sink term in $Kg/(m_{rc}^3.s)$. Equation (2.1) is comprised of 3 main terms as follows:

1. A transport term: $-\nabla \cdot (\rho_w u)$
2. A cumulative term: $\frac{\partial(\rho_w \phi)}{\partial t}$
3. A source/sink term: q

The physical meaning of equation (2.1) is that the variation of water mass per unit volume per unit time inside a specified domain which is given by the cumulative term is equal to the difference between the inflow mass flow rate of water per unit volume going into the domain and the outflow mass flow rate of water per unit volume coming out of the domain where this difference is given by the transport term but taking into account that this temporal variation is increased by an additional mass per unit volume per unit time in case of a source term or decreased by a certain mass per unit volume per unit time in case of a sink term.

The second governing equation of this single-phase fluid flow subproblem is Darcy's law which is given by:

$$u = -\frac{K}{\mu_w}(\nabla P - \gamma_w \nabla Z) \tag{2.2}$$

where K is the absolute permeability in m^2, μ_w is the water viscosity in Pa.s, ∇P is the pressure gradient in Pa/m, ∇Z is the elevation gradient (unitless) and γ_w is the water specific gravity in $Kg/(m^2.s^2)$

By substituting the Darcy flow equation (2.2) in equation (2.1), we get the following pressure equation:

$$\nabla \cdot \left(\rho_w \frac{K}{\mu_w} (\nabla P - \gamma_w \nabla Z) \right) = \frac{\partial (\rho_w \phi)}{\partial t} - q \tag{2.3}$$

In reservoir engineering, it is a common practice to introduce the volume factors which represent the ratio between the subsurface volume of a fluid to its volume at standard or stock tank conditions. Moreover, the source/sink term has a positive sign in case of production and a negative sign in case of injection. However, it should be noted that the concept of volume factors is not used in Dumux and that the reservoir engineering convention for the sign of the source/sink term is actually opposite to the one in Dumux. Consequently, the following equations (2.4) and (2.5) may be valid only for ECLIPSE and are reported here just for a more common representation of the pressure equation within the reservoir engineering domain.

Remembering that

$$\rho_w = \frac{\rho_{wst}}{B_w} \tag{2.4}$$

where B_w is the water formation volume factor representing the ratio of the volume of a certain mass of water at reservoir conditions to its volume at stock tank conditions.

By substituting equation (2.4) in equation (2.3) and then eliminating ρ_{wst} from all terms, we get the following equation:

$$\nabla \cdot \left(\frac{K}{B_w * \mu_w} \cdot (\nabla P - \gamma_w \nabla Z) \right) = \frac{\partial \left(\frac{\phi}{B_w} \right)}{\partial t} - q' \tag{2.5}$$

where the units of all terms in the differential equation (2.5) is s^{-1}

2.2 Heat Transport Subproblem

Energy transfer through the porous medium can occur either by convection or conduction where the solid portion propagates energy only by conduction while the fluid portion can transport energy by both convection and conduction (Bringedal, 2015). Both convective and conductive heat transfer involve the transport of heat from one molecule to another and thus boosting the molecular energy; however, conduction does not result in the translation of molecules due to the high strength of the intermolecular bonds while convection allows molecular translation due to the weak bonds between molecules (English, 2001). That's why conduction is more common in solids compared to fluids as the close contact between the solid molecules allows the transfer of thermal energy due to molecular vibration while in liquids -aside from liquid metals- and gases, the molecules are far apart from each other which diminishes the possibility for molecular collision and the consequent thermal energy transfer (Sahu et al., 2018).

Bringedal, (2015) states that for a porous medium within a representative elementary volume, considering a fluid temperature T_w and a solid (rock matrix) temperature T_s leads to the development of equations (2.6) and (2.7) for the conservation of energy as follows.

Energy conservation equation for the fluid:

$$\frac{\partial(\phi\rho_w U_w)}{\partial t} - \nabla\cdot\left(\rho_w h_w \frac{K}{\mu_w}(\nabla P - \rho_w g)\right) = \nabla\cdot(\phi\lambda_w \nabla T_w) + \nabla\cdot(H(T_s - T_w)) + q^h \quad \textbf{(2.6)}$$

Energy conservation equation for the solid (rock matrix):

$$\frac{\partial((1-\phi)\rho_s c_s T_s)}{\partial t} = \nabla\cdot\left((1-\phi)\lambda_s \nabla T_s\right) + \nabla\cdot(H(T_s - T_w)) \quad \textbf{(2.7)}$$

where U_w is the specific internal energy of the fluid phase in J/Kg, h_w is the specific enthalpy of the fluid phase in J/Kg, ∇T_w and ∇T_s are the temperature gradients of the fluid and the solid respectively in °K/m, λ_w and λ_s are the thermal conductivities of the fluid and the solid respectively in J/(s.m.°K), c_s is the specific heat capacity of the solid in J/ (Kg.°K), q^h is the heat source/sink term in J/(s.m^3) and H is the heat transfer coefficient between the fluid and the solid in J/(s.m^2.°K).

In equations (2.6) and (2.7) above, we can find the following main terms.

1. The convective term or advective heat transfer term: $\left(\rho_w h_w \frac{K}{\mu_w}(\nabla P - \rho_w g)\right)$
2. The conductive heat transfer terms for the fluid and solid respectively: $(\phi\lambda_w \nabla T_w)$ and $\left((1-\phi)\lambda_s \nabla T_s\right)$
3. The thermal energy cumulative terms for the fluid and solid respectively: $\frac{\partial(\phi\rho_w U_w)}{\partial t}$ and $\frac{\partial((1-\phi)\rho_s c_s T_s)}{\partial t}$

Assuming a condition of local thermal equilibrium inside each representative elementary volume such that $T_s = T_w = T$ due to the fine size of the matrix grains and low fluid flow velocity results in the formulation of a single energy conservation equation that applies to both the fluid and the solid rock matrix (Bringedal, 2015; El-Amin, 2019). The resulting equation (2.8) comes from the summing up of the two equations (2.6) and (2.7) as follows.

$$\frac{\partial(\phi\rho_w U_w)}{\partial t} + \frac{\partial((1-\phi)\rho_s c_s T)}{\partial t} - \nabla\cdot\left(\rho_w h_w \frac{K}{\mu_w}(\nabla P - \rho_w g)\right) - \nabla\cdot\left(\lambda_{pm}\nabla T\right) - q^h = 0 \quad \textbf{(2.8)}$$

where $\lambda_{pm} = \phi\,\lambda_w + (1-\phi)\lambda_s$ is the porosity averaged thermal conductivity of the porous medium. The unit for all the left-hand-side terms in equation (2.8) is J/(s.m^3)

Finally, it should be noted that for equations (2.3) and (2.8) in Dumux, the porosity is taken out of the derivative and the porous medium is considered incompressible.

2.3 Initial and Boundary Conditions

Our numerical model that depicts the geothermal aquifer is a symmetrical rectangular domain whose specific parameters are to be discussed in detail in chapter 3. Scenario A models the geothermal system with only one vertical layer for the aquifer, thus neglecting any conduction between the aquifer and the surrounding layers. For scenario B, additional caprock and bedrock were introduced to the model to monitor the temperature behavior for the case of heat transfer by pure conduction. The larger number of cells used in scenario B compared to scenario A due to the modeling of the caprock and bedrock layers caused the computational time to be impractically long. Dumux, which is based on a Finite Volume scheme, is characterized by an already higher computational cost compared to the Finite Difference scheme used in ECLIPSE. For this reason, in Dumux simulation of scenario B, a half-domain was considered, exploiting symmetry conditions along the X-direction. 3D schematic representations of the full domain employed in scenario A and the half-domain employed in scenario B are shown in Figure 2.1 and Figure 2.2 respectively.

It's worth noting that the primary variables for which the initial and boundary conditions are set are pressure and temperature. Pressure is involved in both equations (2.3) and (2.8) while temperature is involved only in equation (2.8).

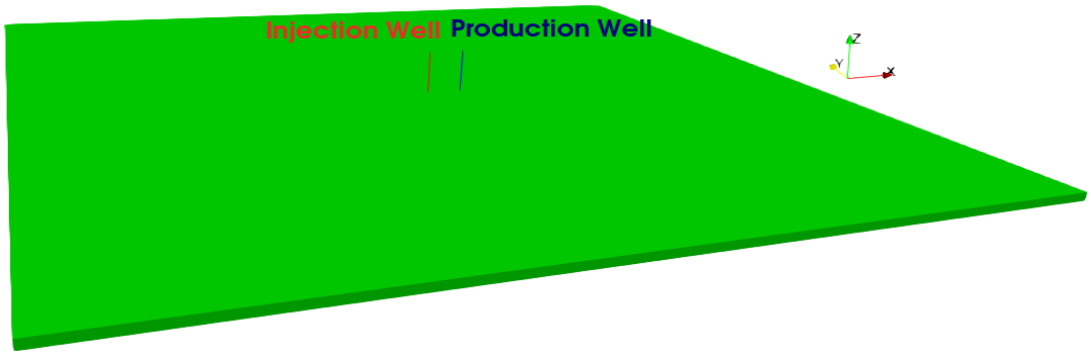


Figure 2.1: 3D full domain schematic used in both Dumux and ECLIPSE for scenario A

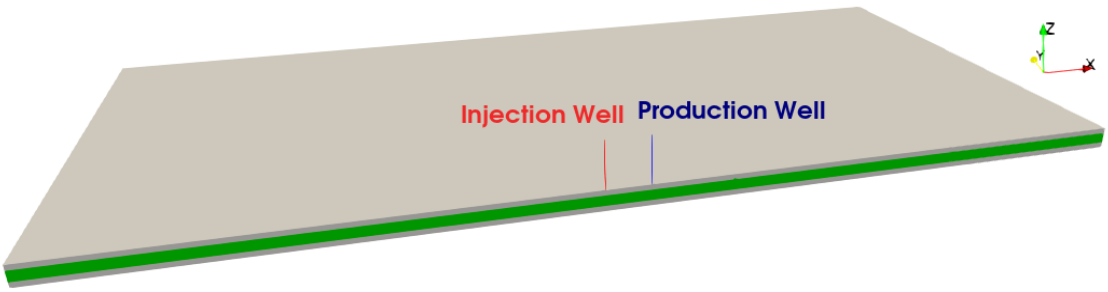


Figure 2.2: 3D half-domain schematic used in Dumux only for scenario B

2.3.1 Initial Conditions

2.3.1.1 Initial pressure

The aquifer layer was modeled with a thickness of 30 m. A hydrostatic pressure gradient of 9.7119 KPa/m or 0.097119 bar/m has been set for the initial pressure condition of the model. The reference pressure is equal to 101.456785 barsa at the center of the aquifer layer which corresponds to a depth of 1015 m. The top of the domain corresponds to a depth of 1000 m in case of scenario A and to a depth of 985 m in case of scenario B due to the presence of a 15 m thick caprock which has the same thickness as the bedrock in that scenario. The pressure on top of the domain is thus 100 barsa in case of scenario A and equal to 98.543215 barsa for scenario B.

2.3.1.2 Initial Temperature

A single initial temperature value of 50.45 °C has been assigned to the whole computational domain in both scenarios. Figure 2.3 and Figure 2.4 show a simplified 2D representation of the initial conditions for the two scenarios investigated within this study.

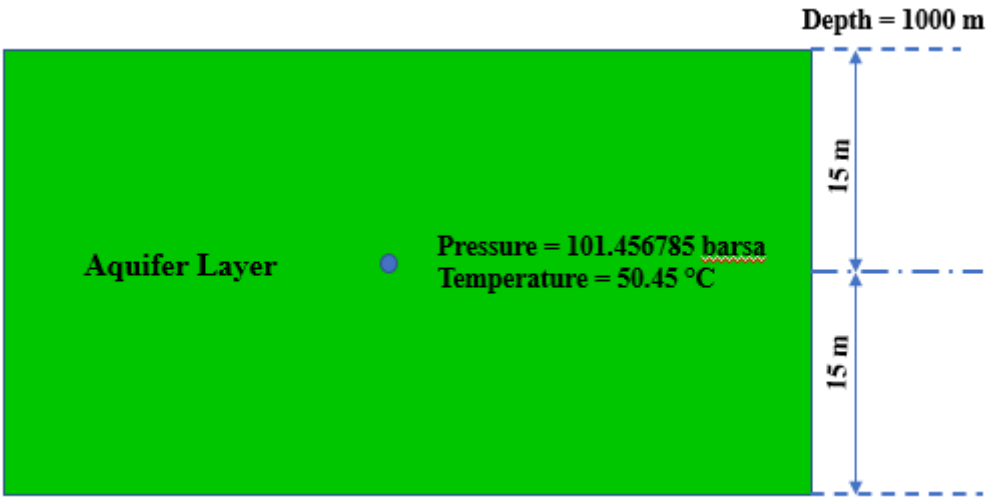


Figure 2.3: 2D schematic for the initial conditions of scenario A

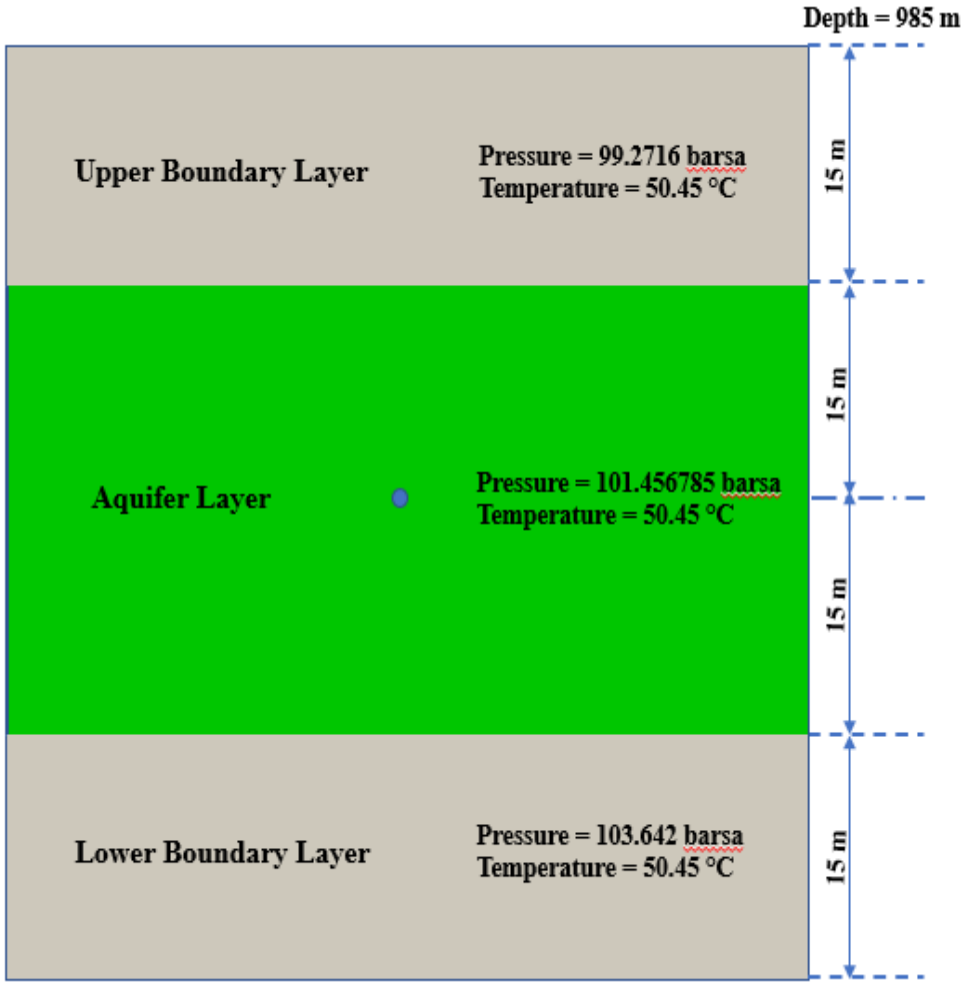


Figure 2.4: 2D schematic for the initial conditions of scenario B

2.3.2 Boundary Conditions for Scenario A

In this scenario, the main focus is modeling the effect of heat transfer by convection, and thus heat transported mainly due to fluid motion. The heat conduction is neglected. As a result, only the aquifer layer was modeled and no caprock or bedrock layers were considered. For the top and bottom boundaries, Neumann no-flow boundary conditions were used for both pressure and temperature such that no fluid flow or heat transport is allowed across those boundaries. Instead, for the lateral boundaries, Dirichlet boundary conditions were used which were assumed to be the same as the initial conditions for pressure and temperature to mimic physically undisturbed boundaries. Figure 2.5 shows a schematic for the boundary conditions applied within scenario A in Dumux. It should be noted that the convention for the Z-direction in the Dumux code is positive upwards unlike ECLIPSE, however, this does not have any effect on any aspect of the work. In Figure 2.5, the red color of the arrows refers to Neumann no-flow boundary conditions while the blue one refers to Dirichlet boundary conditions.

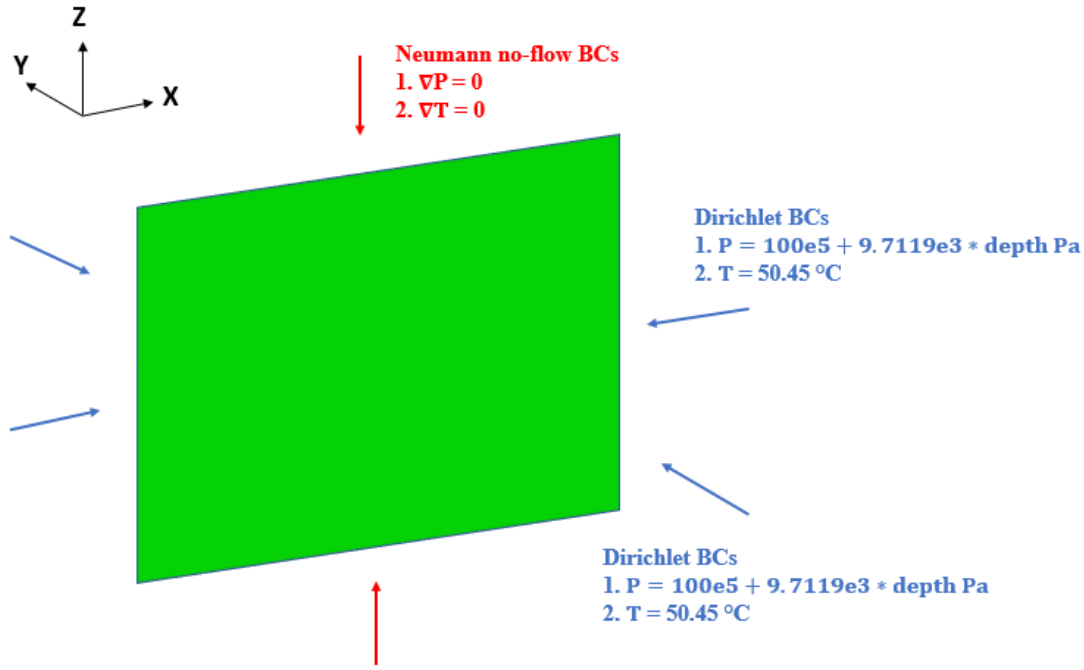


Figure 2.5: Schematic for the boundary conditions of scenario A in Dumux

2.3.3 Boundary Conditions for Scenario B

In this scenario, the phenomenon of heat transfer by conduction is monitored through the introduction of caprock and bedrock to the geothermal aquifer layer. It would make sense physically to set a Neumann no-flow boundary condition for pressure and a Dirichlet boundary condition for temperature for the top and bottom boundaries. However, for the cell-centered Finite Volume scheme that was used for spatial discretization in the Dumux code of this study, Dumux does not allow setting different kinds of boundary conditions for the different equations on the same boundary. As a result, Dirichlet boundary conditions were used for both pressure and temperature on the top and bottom boundaries. The Dirichlet boundary conditions were set to be identical to the initial conditions. In this concern, it was taken into account that the gridding applied to the caprock and bedrock layers ensures undisturbed top and bottom boundaries. This is to be discussed in detail in chapter 3. Regarding the lateral boundaries, Dirichlet boundary conditions exactly the same as the initial pressure, and temperature conditions were set for all boundaries except one. This one lateral boundary is the symmetry plane along which the computational domain was cut in half along the X-direction. Symmetry conditions were approximated by assigning Neumann no-flow boundary conditions for both pressure and temperature across that symmetry plane. The main purpose of applying symmetry conditions and using a half-domain for the Dumux simulation of scenario B is reducing the computational time which will be impractically long after the caprock and bedrock are taken into account due to the high number of

cells. The computational cost is an issue that has to be carefully considered in Dumux since the Dumux code is based on a Finite Volume scheme which is more costly than the Finite Difference approach used in ECLIPSE from the computational point of view.

It should also be noted here that no fluid flow is allowed from the geothermal aquifer layer into the caprock and bedrock layers and thus heat can only be transported by conduction. This was implemented easily in ECLIPSE by setting zero transmissibility along the Z-direction at the layer interfaces while in Dumux, it was not an easy task as it required the modification of the Darcy law C++ class in the Dumux core where the procedure will be discussed in detail in chapter 5. Figure 2.6 shows a schematic for the boundary conditions applied within scenario B in Dumux.

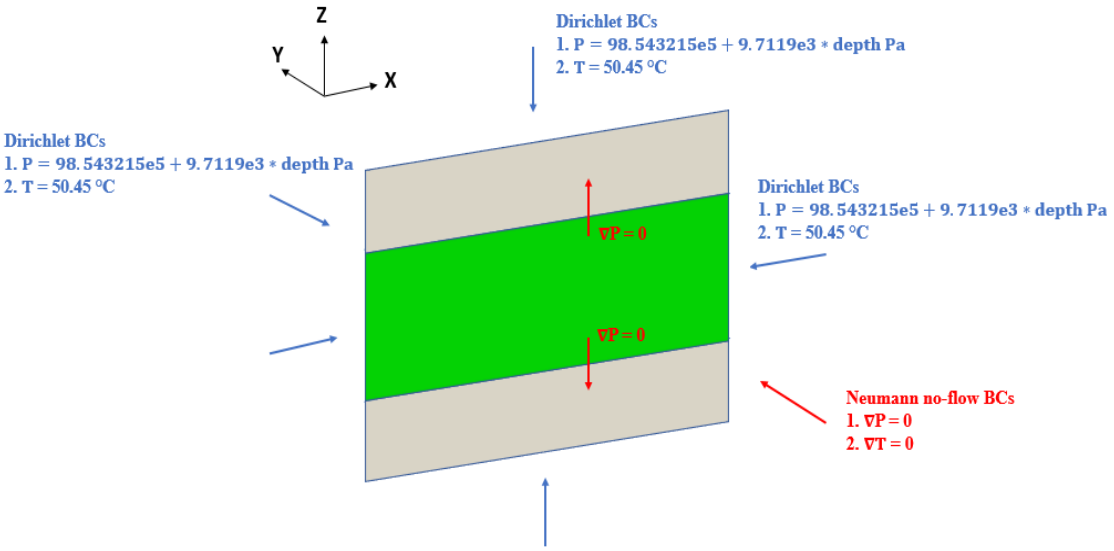


Figure 2.6: Schematic for the boundary conditions of the half-domain used for scenario B in Dumux

3 SYNTHETIC CASE

For the purpose of conducting this study in which non-isothermal fluid flow in a geothermal aquifer is simulated using Dumux, a synthetic case was formulated such that a simplified model geometry was defined for the computational domain. The domain was subdivided into a number of cells to generate a structured grid. The mesh cells were then characterized with physical and thermal properties for both the solid (rock matrix) and the fluid (liquid water).

3.1 The Computational Domain and Well Locations

The geothermal reservoir is modeled as a right-angled parallelepiped whose dimensions are shown in Table 3.1 for each of the scenarios investigated within this study. Production and injection wells are 120 m apart, located along the X-direction, so the injection well is at X=1342 m and the production well is at X=1462 m. They are located at the middle of the Y-direction in the full domain and at the lateral boundary in the half-domain as shown in the schematics of Figure 2.1 and Figure 2.2 respectively.

Table 3.1: Dimensions of the computational domain

<table>
<tr><td rowspan="2">Parameter</td><td colspan="2">Modeled scenario</td></tr>
<tr><td>Scenario A</td><td>Scenario B</td></tr>
<tr><td>Length [m]</td><td colspan="2">2804</td></tr>
<tr><td>Width [m]</td><td>2804</td><td>1402</td></tr>
<tr><td>Aquifer thickness [m]</td><td>30</td><td>30</td></tr>
<tr><td>Caprock thickness [m]</td><td rowspan="2">No caprock and bedrock layers</td><td>15</td></tr>
<tr><td>Bedrock thickness [m]</td><td>15</td></tr>
<tr><td>Total thickness [m]</td><td>30</td><td>60</td></tr>
</table>

3.2 The Grid

The same grid discretization was used for both Dumux and ECLIPSE. The aquifer layer was modeled with only one vertical numerical layer in both scenarios, neglecting gravity effects. Instead, each of the caprock and bedrock layers of scenario B is represented by three numerical vertical layers of thickness 5 m. The use of three numerical layers is aimed at maintaining undisturbed top and bottom boundaries within that scenario to respect the imposed Dirichlet boundary conditions of pressure and temperature.

For all numerical layers, a gradual spatial grid refinement along both X and Y directions from the outer lateral boundary to the inner well area was achieved by using three grid zones such that the outermost zone has the largest cell dimension (200 m x 200 m) and the innermost zone has the finest cell dimension (4 m x 4 m) while the middle zone has a cell dimension that is in between (20 m x 20 m). Grid refinement around the wells is meant to achieve a better simulation accuracy due to the large expected variations of the pressure and temperature gradients within a limited spatial scale (Mehl et al., 2006). On the contrary, grid coarsening is applied to the outermost zone because no significant variations of pressure and temperature are expected. The grid size and cell dimensions are shown in detail in Table 3.2 for both scenarios.

Figure 3.1 shows a schematic of the 3D grid used for scenario A while Figure 3.2 shows a schematic of the 3D grid used for scenario B.

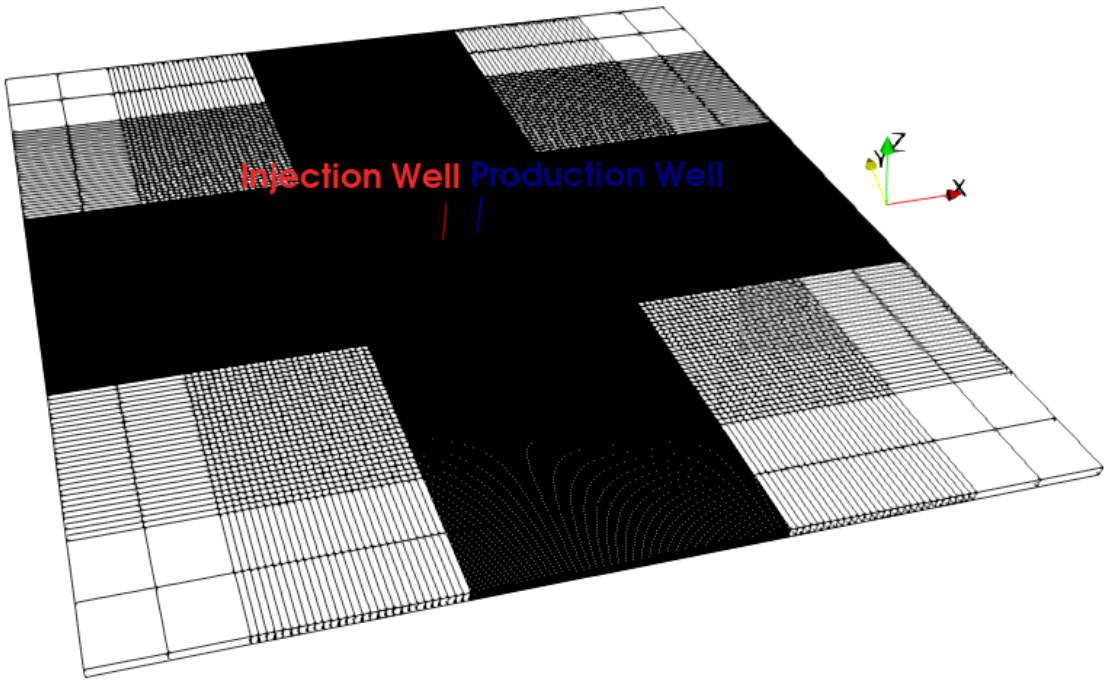


Figure 3.1: 3D grid used for scenario A

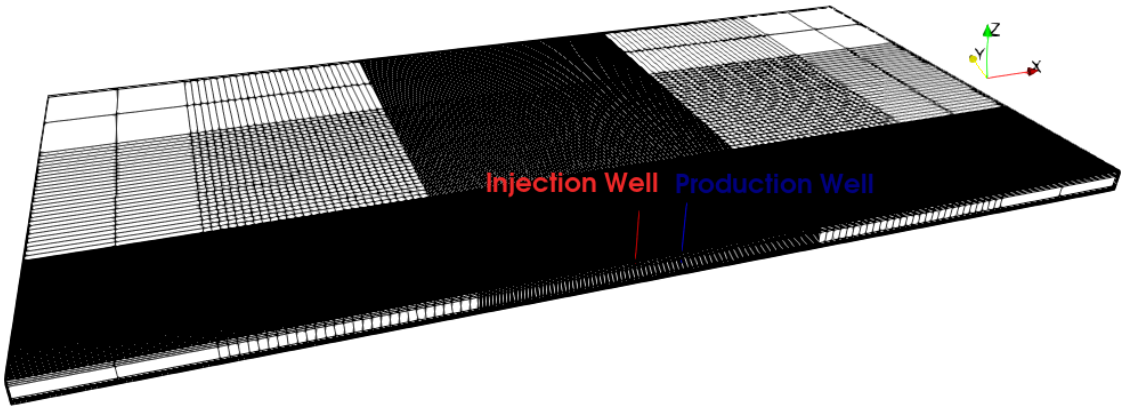


Figure 3.2: 3D grid used for scenario B

In Figure 3.3, a side view from the Y-direction shows the only vertical numerical layer representing the geothermal aquifer for scenario A while in Figure 3.4, the same view reveals the 7 vertical numerical layers for scenario B where each of the caprock and bedrock is represented by 3 vertical numerical layers.

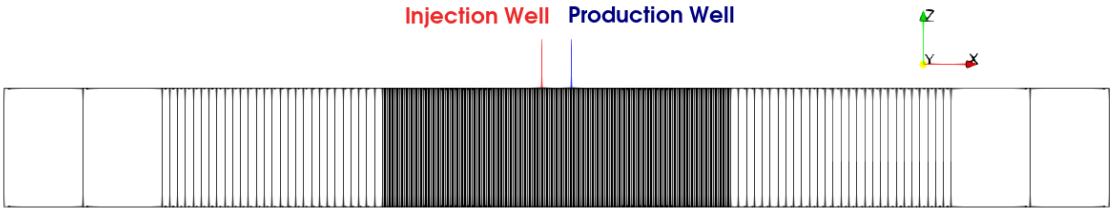


Figure 3.3: Side view from the Y-direction for the grid of scenario A

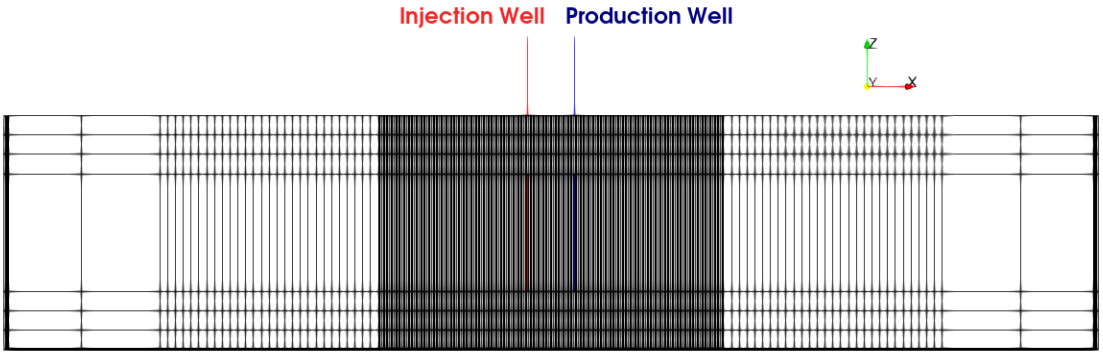


Figure 3.4: Side view from the Y-direction for the grid of scenario B

Table 3.2: Grid size and spatial discretization for both scenarios

Parameter		Modeled scenario	
		Scenario A	Scenario B
Grid size		281x281x1	281x141x7
Discretization in X-direction	Aquifer layer	2 cells of 200 m 28 cells of 20 m 221 cells of 4 m 28 cells of 20 m 2 cells of 200 m	2 cells of 200 m 28 cells of 20 m 221 cells of 4 m 28 cells of 20 m 2 cells of 200 m
	Caprock layer	-	Same as the aquifer layer
	Bedrock layer		
Discretization in Y-direction	Aquifer layer	2 cells of 200 m 28 cells of 20 m 221 cells of 4 m 28 cells of 20 m 2 cells of 200 m	1 cell of 2 m 110 cells of 4 m 28 cells of 20 m 2 cells of 200 m
	Caprock layer	-	Same as the aquifer layer
	Bedrock layer		
Discretization in Z-direction	Aquifer layer	1 cell of 30 m	1 cell of 30 m
	Caprock layer	-	3 cells of 5 m
	Bedrock layer		

3.3 Physical and Thermal Properties of Rock and Water

The initial water density has been set equal to 990 Kg/m^3 based on the value used by Aboulela et al., (2020). All water properties are evaluated in Dumux at each time step according to the pressure and temperature of the control volume at that specific time step based on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam (The DuMux developers, 2020c).

For ECLIPSE, instead, a table of water viscosity as a function of temperature was introduced to the ECLIPSE input file in the PROPS section where the viscosity values were calculated for a temperature range of 16 °C to 56 °C at the initial aquifer reference pressure of 101.456785 barsa using an online water and steam property calculator employing the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties

of Water and Steam for the sake of consistency between both simulators (Kretzschmar et al., 2018). Table 3.3 shows the water viscosity values used in the ECLIPSE input table as a function of temperature. The temperature range for viscosity calculation was chosen so as to include the values of the maximum initial aquifer temperature of 50.45 °C and the minimum temperature of 20 °C of the injected cooled water. Furthermore, the water thermal conductivity and specific heat values used in ECLIPSE were also evaluated using the same online calculator. The water specific heat capacity computed with the online calculator at the initial aquifer reference pressure of 101.456785 barsa varies slightly from a value of 4.15713 KJ/Kg.°K at the initial geothermal system temperature of 50.45 °C to a value of 4.15471 KJ/Kg.°K at 20 °C. Due to the slight variation, a constant value of 4.15713 KJ/Kg.°K was assigned for the water specific heat capacity in the ECLIPSE input file. Regarding water thermal conductivity, it's interesting to note that it's not entered into the ECLIPSE input file as an independent value, but it's rather used as part of the porosity weighted average of rock and water thermal conductivities under the keyword THCONR. In any case, a value of 55.828 KJ/m.day.°K was calculated for the water thermal conductivity at the aquifer's initial reference pressure and temperature of 101.456785 barsa and 50.45 °C respectively.

Physical and thermal properties of solid rock are reported in Table 3.4. The values of rock density and aquifer porosity were taken from Aboulela et al., (2020). The rock thermal conductivity value was taken a little above the average value of Hammam Faroun geothermal aquifer in the same paper which is equal to 2.63 [W/m.°K]. The value of the aquifer's specific heat capacity is the same as the value used by Ganguly et al., (2017). It is pointed out that even if the same thermal properties are imposed for rock in both scenarios A and B, thermal exchange between water and rock becomes significant only in the Scenario B because of the interaction with the caprock and bedrock.

Table 3.3: Water viscosity vs temperature ECLIPSE input table

Temperature [°C]	Water viscosity [cp]
16	1.10343
18	1.04899
20	0.998741
22	0.952261
24	0.909171
26	0.869142
28	0.831883
30	0.797139
32	0.764684
34	0.734318
36	0.70586
38	0.679153
40	0.654052
42	0.630429
44	0.608168
46	0.587167
48	0.567329
50	0.54857
52	0.530811
54	0.513984
56	0.498022

It's evident from Table 3.4 that due to the negligible porosity value of the caprock and bedrock layers, the calculated water thermal conductivity will almost have no impact on the porosity weighted average value of thermal conductivity of those layers especially that the rock thermal conductivity which is equal to 2.8 W/m.°K or 241.92 KJ/m.day.°K is one order of magnitude higher than that of water which is equal to 55.828 KJ/m.day.°K at the aquifer's initial pressure and temperature. The values used in ECLIPSE for the porosity weighted average thermal conductivity for each layer can be seen in Table 3.5

Table 3.4: Physical and thermal properties of rock

Parameter		Value
Porosity [-]	Aquifer layer	0.1
	Caprock layer	0.01
	Bedrock layer	
Lateral permeability [mD]	Aquifer layer	249.26
Vertical permeability [mD]		100
Permeability [mD]	Caprock layer	0.1
	Bedrock layer	
Density [Kg/m^3]	Aquifer layer	2750
	Caprock layer	
	Bedrock layer	
Thermal conductivity [W/m.°K]	Aquifer layer	2.8
	Caprock layer	
	Bedrock layer	
Specific heat capacity [J/Kg. °K]	Aquifer layer	800
	Caprock layer	
	Bedrock layer	

Table 3.5: Porosity weighted average thermal conductivity values used in ECLIPSE for each layer

Layer	Porosity weighted average thermal conductivity [KJ/m.day.°K]
Aquifer layer	223.312
Caprock layer	240.05808
Bedrock layer	240.0624

3.4 Production/Injection History

A volumetric rate of 100 m^3/day is used for both water injection and extraction from the geothermal aquifer. The injection temperature of the cooled water is set to 20 °C.

4 DUMUX OVERVIEW

4.1 Dumux History

Dumux is part of a wider framework called DUNE which stands for Distributed Unified Numerics Environment (Koch, Gläser, et al., 2020). The development of DUNE was initiated in 2003 by Peter Bastian from Heidelberg University in Germany where other individuals made contributions later on till the project became distribued -as its name states- among a wide audience (Sander, 2020). DUNE is a free open-source modular software used for solving partial differential equations by employing methods based on grids such as Finite Differences, Finite Volumes, and Finite Elements (Bastian et al., 2021). It's composed of some libraries where each of those libraries is called a module and provides part of the software functionality (Sander, 2020). Dumux was developed as a DUNE module for the simulation of multi-phase or multi-component or multi-physics or even multi-domain flow and transport processes in porous media. It depends on the DUNE core modules (Koch, Gläser, et al., 2020).

After the first release of Dumux 1.0 in July 2009, according to Koch, Gläser, et al., (2020), the code base has undergone several enhancements to improve the modularity and usability of the code and thus its sustainability. Koch, Gläser, et al., (2020) state that the enhancements that were applied to the 2.X release series were in the form of improving the modeling capabilities of the software and providing it with several discretization schemes. The authors mentioned that Dumux 3.0 was then released in December 2018 followed by Dumux 3.1 in October 2019. Finally, Dumux 3.3 which was released in November 2020 and which was used for this numerical study has some advantages such as encompassing a rich library of multi-component and multi-phase models for flow and transport processes in porous media, having a modular framework for constitutive equations for materials and fluids, and having the ability to couple between different computational domains such as Darcy and Navier-Stokes domains (Koch, Gläser, et al., 2020).

4.2 Dumux Literature Applications

Due to its multipurpose nature, Koch, Gläser, et al., (2020) state that Dumux has been successfully applied to a wide variety of applications including gas storage (Hagemann et al., 2016; Nordbotten et al., 2012; Walter et al., 2012), intercommunication between soil and root systems (Koch et al., 2018; Mai et al., 2019), fluid flow through fractured porous media (Andrianov & Nick, 2019; Fourno et al., 2019; Gläser et al., 2017; Schwenck et al., 2015; Stadler et al., 2012), coupling between subsurface and atmosphere domains (Fetzer et al., 2016; Mosthaf et al., 2011), biochemically driven mineral precipitation (Cunningham et al., 2019; Hommel et al., 2015) and transport of curative agents through living tissues (Erbertseder et al., 2012; Støverud et al., 2012).

Dumux was also employed in numerical studies which aimed at evaluating the efficiency of CO_2 storage in the North German Basin and it was also part of CO_2 storage simulation studies performed on the Ketzin pilot site in Brandenburg, Eastern Germany which has been the longest in operation compared to other European onshore CO_2 storage locations (Tatomir et al., 2019). Kempka et al., (2013) explain in detail the results of the numerical study that was done on the Stuttgart formation which is the saline aquifer at the Ketzin pilot site into which CO_2 is injected.

Weishaupt et al., (2016) used Dumux to investigate the extent of the thermal influence radius, the volume of the steam chamber, and the breakthrough time of the steam through the groundwater table in case of pre-heating the saturated zone that is polluted by NAPL (Non-Aqueous Phase Liquid) prior to steam injection.

Furthermore, many of the Dumux example applications can be found in the dumux-lecture module which is used for academic purposes at the Department of Hydromechanics and Modeling of Hydrosystems at Stuttgart University where this module is subdivided into 3 types of problems: Multiphase Modeling, Modeling of Hydrosystems, and Environmental Fluid Mechanics (The DuMux developers, 2021).

4.3 Dumux Models, Features and Discretization Schemes

The fact that Dumux is based on the modular system DUNE grants the capability of using various grid implementations and linear solvers through the DUNE core modules with no need to worry about the private data structures of each single implementation (Flemisch, 2013). It's further elaborated by Flemisch, (2013) that this allowed Dumux to be more focused on the realization of physical and mathematical models and to have capabilities like grid adaptivity and parallel simulation at a minimum extra programming expense.

Dumux offers a variety of models which can be subdivided into four categories: porous medium flow models, free flow models, geomechanics models and the multidomain module (Flemisch & Class, 2019). The porous medium flow models encompass mainly the single and multi-phase Darcy flow models in porous media besides other models such as Richards flow models, a non-isothermal model, a mineralization model, a model for the storage of supercritical CO_2 and others (The DuMux developers, 2020j). The free flow models according to The DuMux developers, (2020f) are mainly concerned with single-phase Navier-Stokes flow. Geomechanics models consider the solid deformity with the possibility to account for or ignore the fluid pressure as illustrated by The DuMux developers, (2020g). Finally, the multi-domain module provides three different modes for coupling typical Dumux problems (The DuMux developers, 2020h).

The Dumux support for the simulation of coupled models is one of its key features and one of the main motives for its development (Koch, Gläser, et al., 2020). The purpose of such capability as explained by Koch, Gläser, et al., (2020) is to solve a coupled system of partial differential equations such that a subset of this system belongs

to a domain with unsimilar dimensionality or a domain for which an unsimilar mesh or spatial discretization scheme is defined. Another advantage provided by Dumux is the possibility to run a parallel simulation. This capability exploits multicore systems such that the simulation domain is partitioned into sub-domains where each sub-domain has its own local problem and all local problems are solved in parallel (The DuMux developers, 2021). However, parallelization in Dumux is only possible because of the DUNE support for the Message Passing Interface (MPI) which sends and receives data between sub-domains where parallelization can be applied only if the parallel linear solver Dumux::AMGBiCGSTABBackend is selected as explained by The DuMux developers, (2021). Moreover, Dumux offers the capability of grid adaptivity in time (Flemisch & Class, 2019). This capability can be attractive as it can save the computational cost especially in the case of large simulation domains with long simulation time spans. However, grid adaptivity is not supported by all DUNE grids such as the Yasp grid which has been implemented in the Dumux code of this study (Koch & Schneider, 2015). Finally, an interesting feature of Dumux is that it allows realistic grids to be used for Dumux applications through its support for corner-point grids which are typically adopted by the industrial simulator ECLIPSE (The DuMux developers, 2021). The use of the Eclipse Grid Format (GRDECL), however, requires the installation of an additional module which is the opm-grid module along with its dependencies as outlined by The DuMux developers, (2021).

Dumux offers three main approaches for spatial discretization which are the box method, cell-centered finite volume methods and the staggered grid scheme (The DuMux developers, 2021). The box method exploits the benefits of both Finite Element and Finite Volume methods such that a primary unstructured Finite Element grid can be used along with a secondary mass conservative Finite Volume grid (Flemisch & Class, 2019). In cell-centered Finite Volume methods, each element/cell of the mesh acts as a control volume (The DuMux developers, 2021). These methods include a two-point flux approximation method (TPFA) and a multi-point flux approximation method (MPFA) (The DuMux developers, 2020a). The staggered grid method is a Finite Volume method which assigns the velocity components control volumes which are shifted from those assigned for the scalar quantities and it's used only for free flow models (Navier-Stokes flow) in Dumux (Flemisch & Class, 2019). It's worth mentioning that only the cell-centered Finite Volume and the box methods support the grid adaptivity feature (The DuMux developers, 2021).

4.4 Dumux Code Structure

The fundamental elements of any simulation in Dumux are in the form of C++ classes aside from the main function of the code (Koch, Gläser, et al., 2020). The authors illustrate that those classes are:

- The Problem class (problem.hh): it contains the imposed initial and boundary conditions besides the source/sink terms. The properties of the system such as the mathematical model, spatial discretization scheme, grid type and fluid

type are also usually merged inside the problem file instead of being defined in a separate file.

- The SpatialParams class (spatialparams.hh): it identifies the parameters of the porous medium that vary spatially, but not temporally, such as porosity and permeability.

Additional files can be created to introduce user-defined classes for particular needs. This is the main added value of an open-source code, which allows a significant flexibility with respect to a compiled specialized software designed for oil & gas reservoir simulation, like ECLIPSE. For instance, it allows the implementation of user-defined boundary conditions or equations of state. On the other hand, the multipurpose nature of Dumux often requires some personalization to meet the user requirements, since several options already taken into account in a specialized software like Eclipse are not available. Such options include the definition of zero transmissibility barriers, well bottom hole pressure calculation, well head pressure calculation and well control mode.

The main function of the program (main.cc file) is an essential component of any Dumux application as it's responsible for solving the system of partial differential equations by calling the assembler and solvers besides that it exports the simulation results to the output files which are of VTK format by initializing the VTK output module (Scholz et al., 2018; The DuMux developers, 2021).

Some of the simulation parameters can be analyzed by the software at run time instead of compilation time if they're defined in an input/parameters file (params.input) where those parameters can be easily changed and the code can be re-run without the need for code re-compilation (The DuMux developers, 2021). A table for this set of parameters is provided by the Dumux code documentation (The DuMux developers, 2020i). The table structure clearly shows that the set of available parameters is further subdivided into subsets such that each subset of parameters should be defined in the input file below a certain group/keyword.

Finally, to build the model, a configuration file (CMakeLists.txt) which uses the CMake language is required to exist in the same directory as the previously mentioned files where CMake is a tool dedicated for software building tasks (Scholz et al., 2018).

The pre-mentioned files (problem.hh, spatialparams.hh, params.input, main.cc and CMakeLists.txt) represent the total list of files that have to be compiled in order to run a Dumux simulation.

4.5 Dumux vs Other Simulators

Benchmarking studies which hold intercomparisons between different simulators for well-known problems are very important for the validation of numerical simulators (Tatomir et al., 2019). Dumux has taken part in some of these studies and comparisons where its performance in three of them is to be discussed briefly in this section.

4.5.1 The Benchmark Study by Class et al., (2009)

In this large study, the authors addressed three benchmark problems related to CO2 injection and storage in geological formations. However, we focus our attention on the first problem only in which Dumux was used as Dumux was still being developed and didn't have the features required to model the other problems. The problem targets the quantification of the leakage rate of CO2 that is spreading due to advection after being injected into a bottom aquifer where the leakage takes place through a leaky well that is connecting the bottom aquifer to a top aquifer such that the two aquifers are separated by an aquitard. A 2D schematic of the problem obtained from Class et al., (2009) is shown in Figure 4.1.

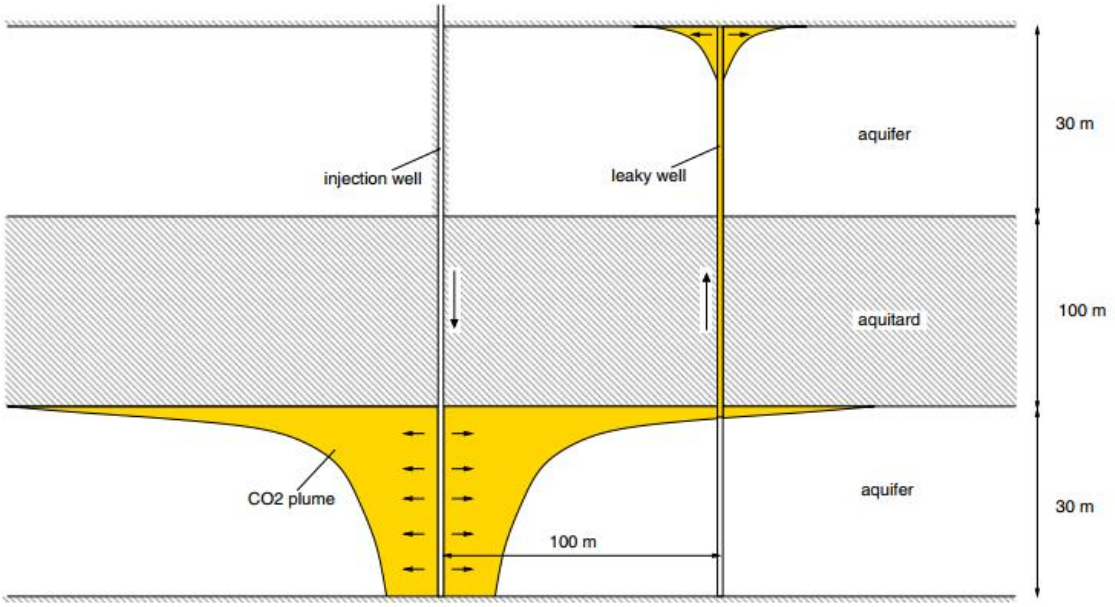


Figure 4.1: Model setup used for the first problem (Class et al., 2009)

According to Class et al., (2009), the numerical domain for this first problem is 3D with 1000 m x 1000 m lateral dimensions and with vertical thicknesses as elaborated in Figure 4.1 where the leaky well was modeled as a porous medium with a permeability that is higher compared to that of the surrounding porous medium. Nine different simulators were involved in the comparison against Dumux. The participating simulators are CO_2 Reservoir Environmental Simulator, ECLIPSE, Estimating Leakage Semi-Analytically, Finite Element Heat and Mass Transfer Simulator, IPARS-CO2, MUFTE, RockFlow, TOUGH2, and Vertical Equilibrium with Sub-scale Analytical. For Dumux, a 2-phase model with a fully implicit solution scheme was used to address the problem. Class et al., (2009) point out that Dumux was compared against the other simulators under simplified conditions where the fluid properties were assumed to be constant, non-isothermal effects were neglected and capillary effects were ignored. The results are reported by Class et al., (2009) as shown in Figure 4.2 in terms of the percentage ratio of the leaked to the injected CO_2

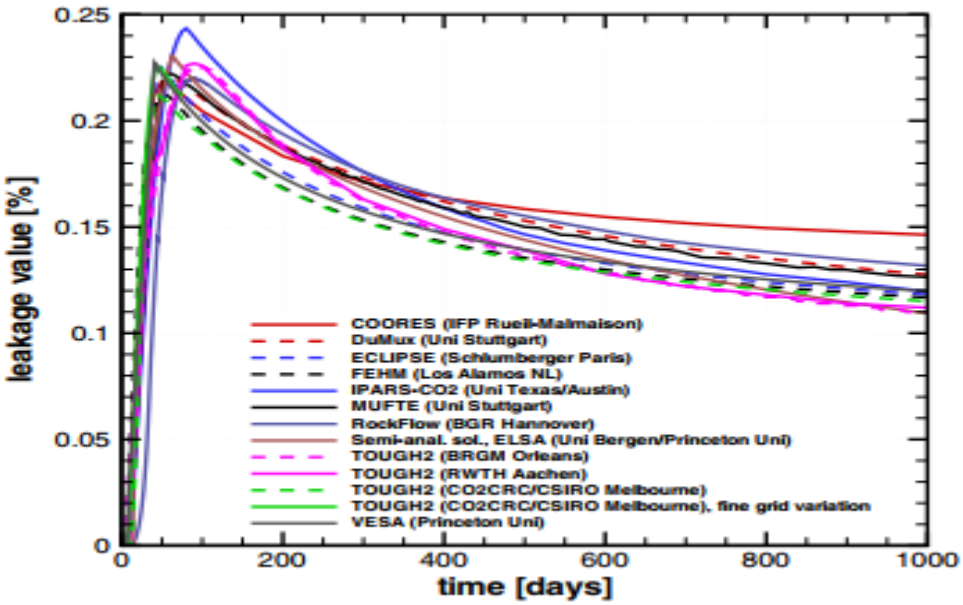


Figure 4.2: Computed CO2 leakage ratios from Class et al., (2009)

It can be seen that the curve predicted by Dumux shows a good match with the curves of the other simulators and Class et al., (2009) confirm the agreement of the results by all simulators and that the observable deviations are small.

4.5.2 Dumux vs COMSOL Multiphysics

Zhou et al., (2020) conducted a benchmark study to compare the hydro-mechanical effects simulated by both Dumux and the commercial simulator COMSOL Multiphysics due to isothermal single-phase water injection into a multi-layered geological domain. The model is composed of 5 layers which are: the upper aquifer, the caprock, the reservoir layer, the bottom confining layer, and finally the bottom aquifer where all layers are considered perfectly horizontal and also isotropic and homogeneous regarding their mechanical and hydraulic characterization with water injection taking place at the left boundary of the domain into the reservoir layer. Four observation points were set by Zhou et al., (2020) as indicated by Figure 4.3 to monitor mainly the variation of overpressure due to injection as a function of time. Figure 4.3 also shows the different boundary conditions where a constant vertical stress of 24 MPa was applied at the upper boundary which is equal to the lithostatic pressure on top of the domain assuming a constant rock density of 2400 Kg/m^3 resulting in a gradient for the effective vertical stress equal to 24 MPa/Km that was used to assign initial stress conditions with a value for the effective horizontal stress that is half the vertical one.

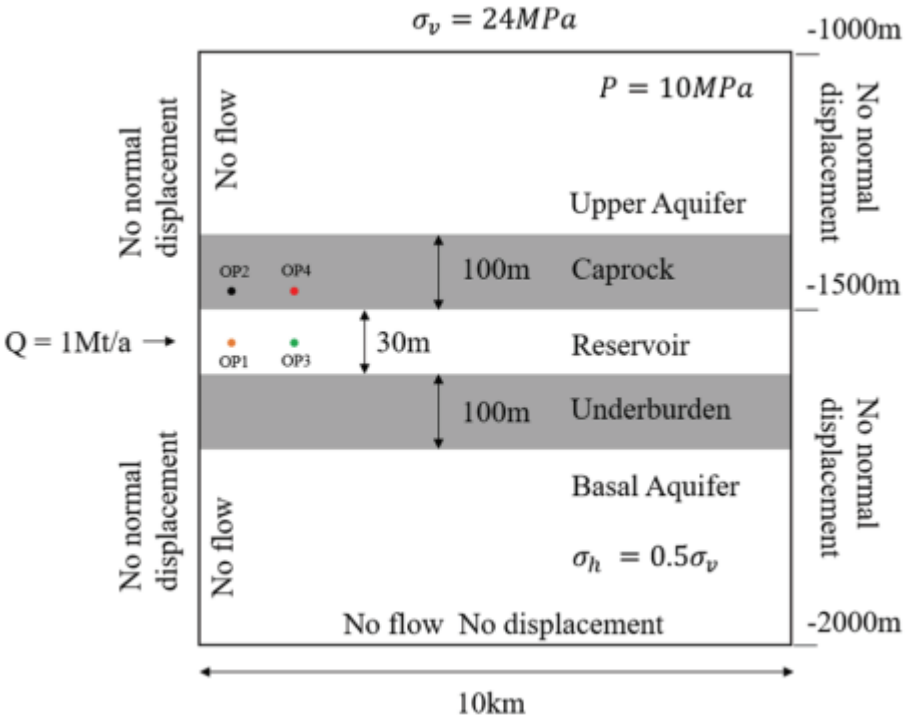


Figure 4.3: Schematic of the geometry of the model, initial and boundary conditions from Zhou et al., (2020)

A very good match of the overpressure variation with time calculated by the two simulators at the observation points was reported by Zhou et al., (2020) as shown in Figure 4.4 and Figure 4.5

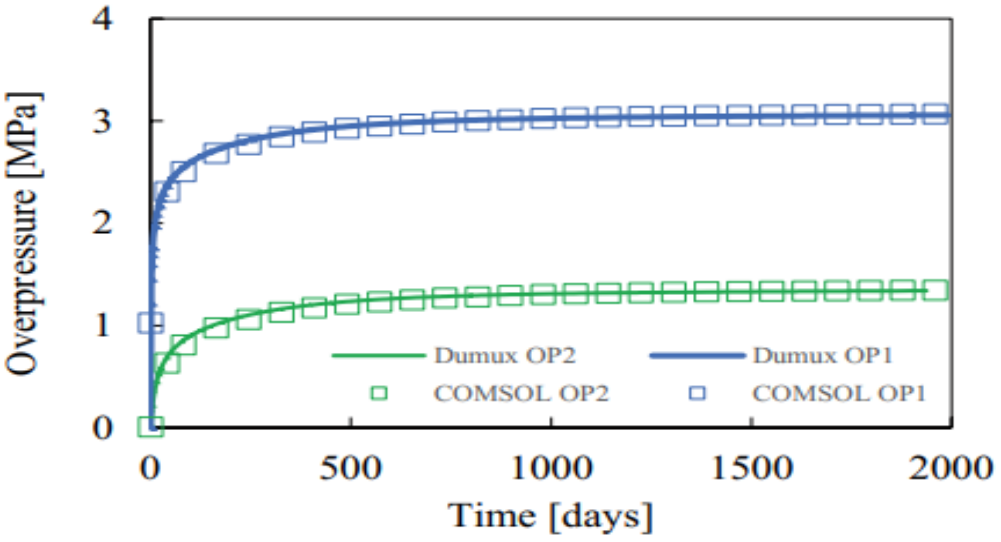


Figure 4.4: Comparison between Dumux and COMSOL Multiphysics for the overpressure variation at observation points 1 and 2 (Zhou et al., 2020)

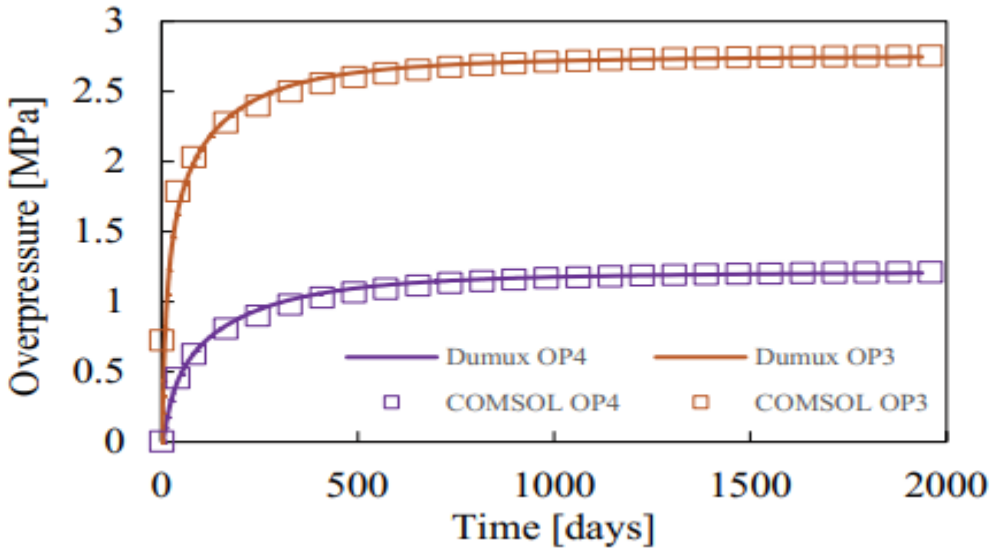


Figure 4.5: Comparison between Dumux and COMSOL Multiphysics for the overpressure variation at observation points 3 and 4 (Zhou et al., 2020)

The reported results also included the vertical displacement quantification over the interface between the reservoir and the caprock and the interface between the caprock and the upper aquifer as shown in Figure 4.6 and Figure 4.7

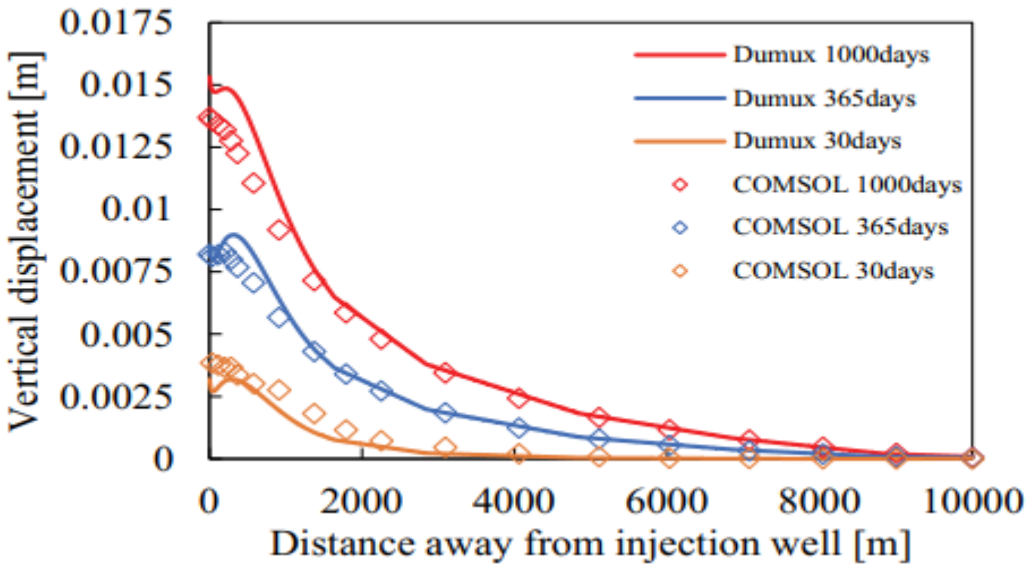


Figure 4.6: Comparison between Dumux and COMSOL Multiphysics for the vertical displacement over the reservoir-caprock interface (Zhou et al., 2020)

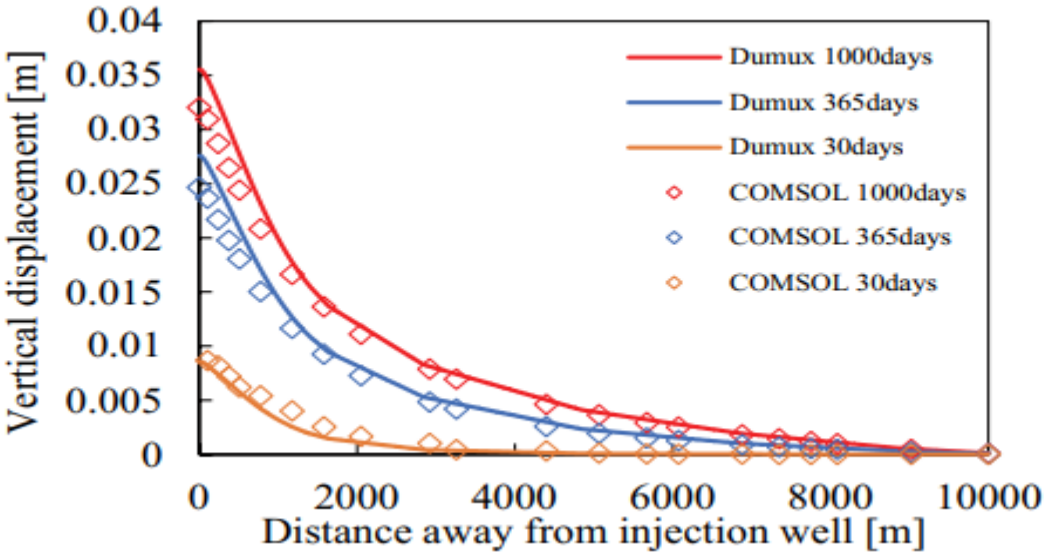


Figure 4.7: Comparison between Dumux and COMSOL Multiphysics for the vertical displacement over the caprock-upper aquifer interface (Zhou et al., 2020)

Again, the authors report that the vertical displacement results of both simulators match very well highlighting that deviations are more pronounced in the near-wellbore area and at earlier times. It's worth noting that the two simulators used different spatial discretization schemes where a Finite Element scheme was implemented in COMSOL while Dumux used the Box method for that specific study which is a mixture between Finite Element and Finite Volume methods (D. Zhou et al., 2020).

4.5.3 Dumux vs ECLIPSE 100 and TOUGH2

Kempka et al., (2013) did a numerical study on the Stuttgart formation which is the saline aquifer at the Ketzin pilot site into which CO2 is injected. Three workgroups have been formed to make an intercomparison between the results obtained by ECLIPSE 100, TOUGH2, and Dumux. According to Kempka et al., (2013), the workgroup from Stuttgart university used an isothermal Dumux model including compositional effects. The authors conclude that the simulation results obtained by Dumux exhibit a good to excellent matching between simulated and observed pressures at the injection and monitoring wells. However, the results showed a deviation of the calculated free gas phase by Dumux that is 10% higher compared to ECLIPSE and TOUGH2 results after around four years of simulation time as indicated in Figure 4.8

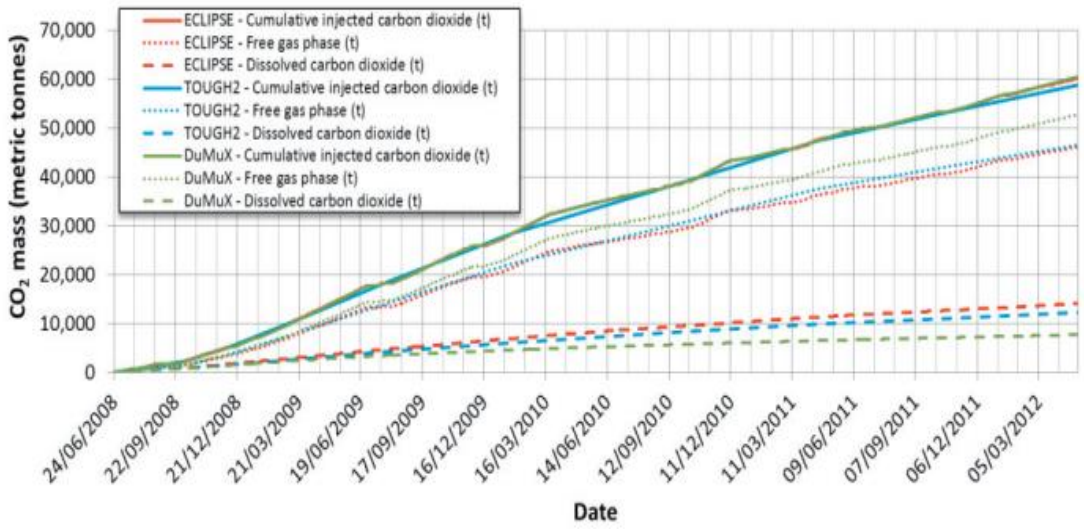


Figure 4.8: CO2 mass balance calculated with ECLIPSE, TOUGH2, and Dumux simulators (Kempka et al., 2013)

5 SIMULATION SET-UP WITH DUMUX

This chapter highlights the necessary elements for setting up the Dumux simulation of the considered case studies. As previously mentioned, Dumux is a multipurpose open-source software based on the C++ language, whose simulation setup goes through the modification/implementation of:

- the main function (main.cc) which is responsible of calling the assembler, the solvers, setting the temporal discretization scheme, and exporting the simulation results.
- the input file (params.input) which defines simulation parameters used at runtime, i.e. that can be changed without compiling the code again, such as grid discretization, timestep size, total simulation time, linear and non-linear solver parameters in addition to rock/solid properties.
- the problem file (problem.hh) which defines the mathematical model, the spatial discretization scheme, the grid type, initial and boundary conditions as well as the source/sink terms.
- the spatial parameters file (spatialparams.hh) which characterizes the porous medium.
- the compilation file (CMakeLists.txt), where the name of the executable is defined and this file must be located in the same directory as the previous files so that the compiler can access them.
- additional user-defined classes for particular needs.

The first five aforementioned files were modified for the Dumux code of scenario A while for scenario B, an additional file was also introduced (modifieddarcy.hh) to allow the introduction of transmissibility barriers between the aquifer and both the caprock and bedrock. Such barrier ensures no mass flux of water from the geothermal aquifer into the caprock or bedrock but allows heat transfer due to conduction. This was achieved by introducing a modification to the Darcy law class in the Dumux core.

In what follows, details on simulation set-up and the corresponding file modification/implementation are given. The main differences between Dumux and ECLIPSE regarding the model implementation are also outlined. Overview of basic C++ concepts and nomenclature is given in the appendix for reader's convenience.

5.1 Mathematical Model

The non-isothermal single-phase model (`OnePNI`) in Dumux is selected as the mathematical model that represents the modeled physical phenomena previously stated in chapter 2. This model employs the previously discussed equations (2.3) and (2.8) since it inherits the properties of the isothermal single phase model (1p) in Dumux plus the additional energy balance equation (2.8) from the non-isothermal model (The DuMux developers, 2020e). The selection of the OnePNI model is done by defining a new type tag within the `TTag` namespace in the problem file (problem.hh) under the

name OnePNITypeTag which inherits the properties of the OnePNI model as shown below.

```
namespace TTag {
struct OnePNITypeTag { using InheritsFrom = std::tuple<OnePNI>; };
```
(C5.1)

5.2 Spatial Discretization

In this study, we adopted the cell-centered Finite Volume scheme called multi-point flux approximation (MPFA) for the spatial discretization of the two governing equations (2.3) and (2.8) of the model.

The MPFA scheme has been utilized in the oil sector since the mid-1990s (Nordbotten & Eigestad, 2005). This scheme was developed to attain a sound discretization of the flow equations for the case of non-perpendicular grids with a general alignment of the principal axes of the permeability tensor and thus it was considered suitable to be applied for practical flow problems in actual reservoirs (Aavatsmark, 2002; Nordbotten & Eigestad, 2005).

Being a Finite-Volume based discretization method, MPFA ensures local conservation of the physical quantities such as mass and energy in a way that is analogous to how the equations of reservoir simulation are developed with no numerical sources/sinks created for such quantities (Ambrus et al., 2010; Moog, 2013). This is achieved by imposing flux continuity between the cells of the grid (Starnoni et al., 2019). Moreover, the solution over the control volume is averaged and the result is assigned as the value of the variable at the center of the control volume (Fontes, 2018). The cell-centered Finite Volume schemes are thus different from the simple Finite Difference approach implemented in the industrial simulator ECLIPSE in which there is no continuity between the cells neither in terms of flux nor in terms of variables. It also differs from finite Elements where the continuity of a variable across a cell can be imposed by means of a shape function (P. Zhou, 1993).

To use MPFA in the Dumux simulation, a new type tag called OnePNICCMpfa had to be defined in the problem file (problem.hh) to inherit the properties of the newly created OnePNITypeTag plus those of the cell-centered multi-point flux approximation model (CCMpfaModel) as follows.

```
struct OnePNICCMpfa { using InheritsFrom = std::tuple<OnePNITypeTag,
CCMpfaModel>; };
} // end namespace TTag
```
(C5.2)

In cell-centered Finite Volume methods, the elements/cells of the grid are used as control volumes; each control volume is subdivided into sub-control volumes (Koch, Gläser, et al., 2020; The DuMux developers, 2021). The sub-control volumes are generated by connecting the center of each cell in the grid to the midpoints of its faces resulting in a dual grid (Starnoni et al., 2019). Figure 5.1 shows the schematic in 2D of the original grid (solid line) and the dual grid (dashed line). Each cell of the original grid represents a control volume, while each cell in the dual grid is termed interaction region (Nordbotten & Eigestad, 2005). It can be seen from Figure 5.1 that all the inner

and boundary faces of the control volumes are split by the dual grid into two sub-control-volume faces for this 2D case (Koch, Gläser, et al., 2020; Nordbotten & Eigestad, 2005). This is illustrated in the figure by the solid line representing one of the four sides of a cell/control volume cut in half by the dashed line of the dual grid.

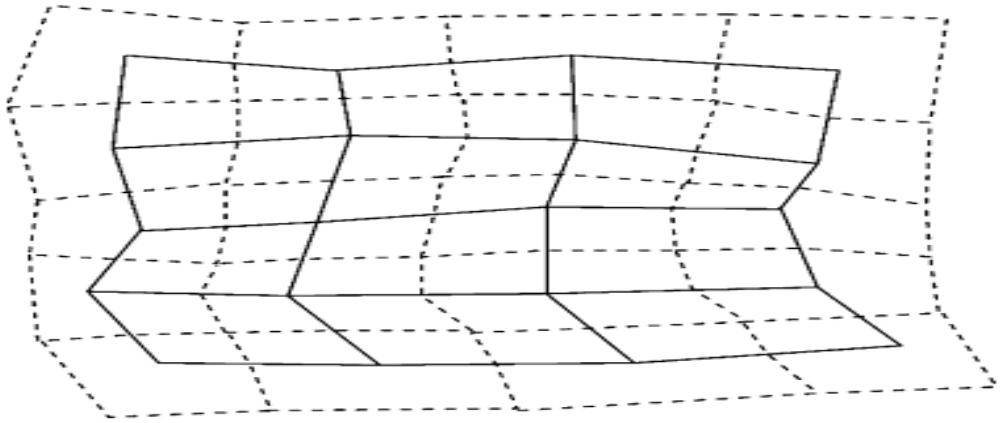


Figure 5.1: A 2D schematic of the original grid of control volumes (solid lines) and the dual grid (dashed lines) (Nordbotten & Eigestad, 2005)

For each interaction region, the local intercommunication between the sub-control volumes leads to the evaluation of the transmissibility coefficients of the sub-control volume faces inside that region (Aavatsmark, 2002). The interaction between the sub-control volumes is governed by certain regulations as explained by Starnoni et al., (2019). Firstly, flux continuity is imposed across the sub-control volume faces. Secondly, the pressure is considered to be linear in each sub-control volume. Finally, pressure continuity is imposed at the midpoints of the control volume faces which correspond to the points of intersection of the dual grid with the original grid.

Following the MPFA approach, Nordbotten & Eigestad, (2005) show that the continuous flux across a sub-control volume face can be approximated by the following expression in terms of velocity.

$$u_i = \sum_{j \in \pi} T_{ij}\, p_j \qquad \textbf{(5.1)}$$

where T_{ij} are the transmissibility coefficients which accommodate permeability, viscosity and grid dimension (Negara et al., 2014), p_j is the pressure at the center of cell j and π is composed of 6 cells in a 2D scheme as shown in Figure 5.2 while it's composed of 18 cells in case of a 3D problem (Negara et al., 2014). In Figure 5.2, the 6-point stencil of the MPFA method in a 2D problem is shown: the numbers represent the cell centers, and the long arrow represents the flux from cell 12 to cell 22. As shown in the figure, the pressures of the 6 cells will all influence the calculation of the flux across the interface of cells 12 and 22. In fact, each of the two interaction regions delimited by vertices 12 22 23 13 and 12 22 21 11 respectively, will have a flux contribution across its own half of the interface of cells 12 and 22 denoted by the two small arrows and those contributions have to be summed up to get the requested flux as per equation

(5.1). In case of a 3D problem, four interaction regions will contribute to the flux calculation (Aavatsmark, 2002).

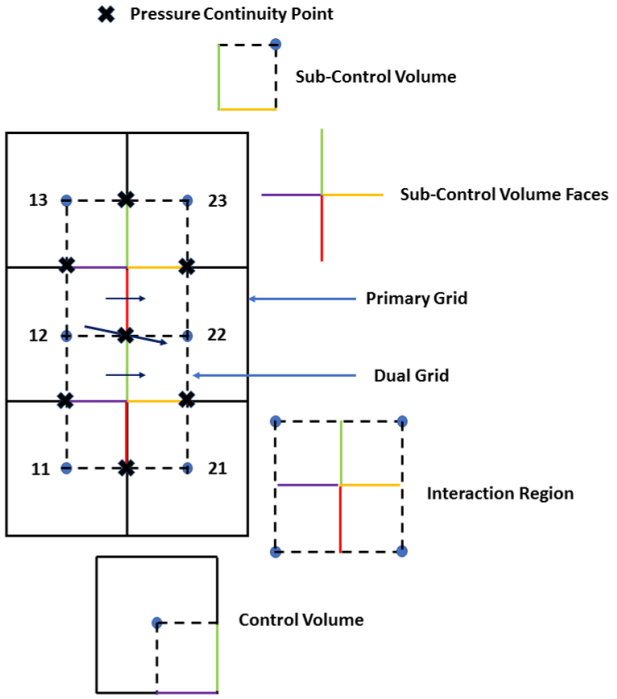


Figure 5.2: The 6-point stencil of the MPFA method in a 2D problem

Considering the single interaction region shown in Figure 5.3 and taking into account the pre-mentioned principles of the MPFA method (pressure and flux continuity and the linear approximation of pressure inside each sub-control volume), the velocity u_x^- for instance can be approximated as follows (Negara et al., 2014):

$$u_x^- = -T_{11}^x(P_x^- - P_{11}) - T_{11}^x(P_y^- - P_{11}) \quad \textbf{(5.2)}$$

$$u_x^- = -T_{21}^x(P_{21} - P_x^-) - T_{21}^x(P_y^+ - P_{21}) \quad \textbf{(5.3)}$$

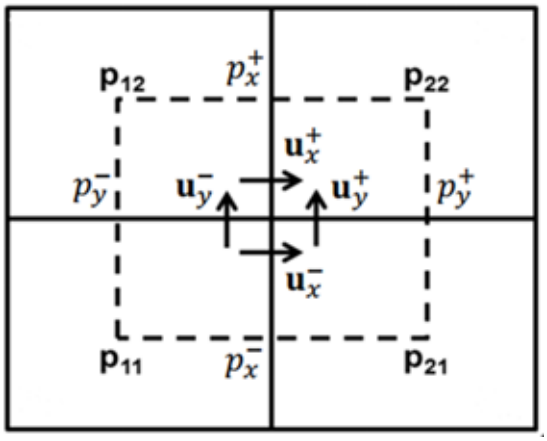


Figure 5.3: A single interaction region in the MPFA method for a 2D case (Negara et al., 2014)

Due to the enforced flux continuity, equations (5.2) and (5.3) can be equalized and the same can be done for u_x^+, u_y^- and u_y^+ (Aavatsmark, 2002) resulting in a system of equations that can be represented in the following matrix form (Nordbotten & Eigestad, 2005)

$$\mathrm{A}p_\sigma = \mathrm{B}p \tag{5.4}$$

where p_σ refers to the vector of pressures of the continuity points, p refers to the vector of cell center pressures and both A and B are 4 x 4 matrices for a 2D case. Due to the matrix A being invertible, the interface pressures (pressures of the continuity points) can be removed by expressing them in terms of the cell center pressures as follows (Nordbotten & Eigestad, 2005):

$$p_\sigma = \mathrm{A}^{-1}\mathrm{B}p \tag{5.5}$$

Considering only one of the two equations (5.2) and (5.3) for u_x^- and similarly for u_x^+, u_y^- and u_y^+, the velocities can be expressed as (Aavatsmark, 2002):

$$u = \mathrm{C}p_\sigma - \mathrm{D}p \tag{5.6}$$

such that each of C and D is a 4 x 4 matrix for 2D problems. By substituting equation (5.5) in equation (5.6), the following expression for velocities is obtained (Aavatsmark, 2002; Nordbotten & Eigestad, 2005).

$$u = (\mathrm{C}\mathrm{A}^{-1}\mathrm{B} - \mathrm{D})p \tag{5.7}$$

By considering equation (5.1) in the following matrix form.

$$u = \mathrm{T}p \tag{5.8}$$

and by relating the two equations (5.7) and (5.8), the transmissibility coefficients of the sub-control volume faces within a single interaction region are expressed as follows (Aavatsmark, 2002):

$$\mathrm{T} = (\mathrm{C}\mathrm{A}^{-1}\mathrm{B} - \mathrm{D}) \tag{5.9}$$

The elements of each row of the resulting transmissibility matrix given by equation (5.9) represent the weighting factors of the cell center pressures contributing to the approximation of the flux across the corresponding sub-control volume face (Nordbotten & Eigestad, 2005). For a 3D problem as in the case of this study, the interaction region will consist of eight sub-control volumes and 12 sub-control volume faces as shown in Figure 5.4 (Aavatsmark, 2002). As a result, each of A and C will be a 12 x 12 matrix while each of T, B and D will be a 12 x 8 matrix.

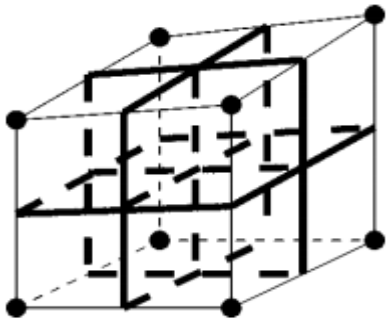


Figure 5.4: A single interaction region in the MPFA method for a 3D case (Aavatsmark, 2002)

5.3 Temporal Discretization

The simple approach of first order difference quotient is used in Dumux for the discretization of the storage/cumulative terms. Considering a general storage term $\frac{\partial S(L)}{\partial t}$ where S is storage and L is the unknown quantity as per The DuMux developers, (2021), the temporal derivative can be expressed as follows.

$$\frac{\partial S(L)}{\partial t} = \frac{1}{\Delta t}\left[S(L_{n+1}) - S(L_n) \right] \tag{5.10}$$

where n+1 refers to the current time step while n refers to the previous time step.

The default mode for temporal discretization in Dumux and which has also been used in this study is a backward difference (implicit Euler). The mode can be changed to forward difference (explicit Euler) through the third template argument of the class FVAssembler in the main function of the code as it can be set to true or false. However, in the code snippet (C5.3), it can be seen that only two template arguments were used for the FVAssembler class because the default mode (implicit Euler) was desired and thus there was no need to input a third argument.

```
using Assembler = FVAssembler<TypeTag, DiffMethod::numeric>;
```
(C5.3)

Using the implicit Euler method for temporal discretization means that by substituting equation (5.10) in the following general balance equation from (Flemisch & Class, 2019)

$$\frac{\partial S(L)}{\partial t} + \nabla . F(L) + Q(L) = 0 \tag{5.11}$$

where F is the flux and Q is the source, the equation becomes:

$$\frac{1}{\Delta t}\left[S(L_{n+1}) - S(L_n) \right] + \nabla . F(L_{n+1}) + Q(L_{n+1}) = 0 \tag{5.12}$$

It should be pointed out here that Dumux uses an adaptive time stepping based on the difficulty of solution convergence where the user can set in the parameters/input file (params.input) the maximum time step and the maximum number of times the time step

size is halved in case of unachieved convergence under the keywords **[TimeLoop]** and **[Newton]** respectively.

5.4 Grid

The grid adopted in this study is a structured cube grid called Yasp, (Yet Another Structured Parallel), which is made available by DUNE besides other possible grid implementations (Sander, 2020). The Yasp grid is available in two different versions: a simple one and a more feature-rich one that allows, among others, splitting the grid into a number of zones, controlling the number of cells in each of those zones, grading the cell dimensions within a certain zone according to a factor assigned in the input/parameters file (Coltman et al., 2021).

The second version was used in this study. It was set in the Dumux code in the problem file (problem.hh) by assigning the corresponding grid type to the node TTag::OnePNITypeTag that has been created previously as shown in the code snippet below.

```
template<class TypeTag>
struct Grid<TypeTag, TTag::OnePNITypeTag> { using type = Dune::YaspGrid<3,
Dune::TensorProductCoordinates<double, 3> >; };
```

(C5.4)

In the code snippet (C5.4), the number 3 indicates that the grid is three-dimensional; double refers to the data type of the grid coordinates; TensorProductCoordinates represents a specific coordinate system provided by DUNE used by the more feature-rich version of the Yasp grid (Coltman et al., 2021).

The grid discretization is provided to Dumux in the input file (params.input) under the **[Grid]** keyword as shown in the code snippets (C5.5) and (C5.6) for the full domain of scenario A and the half-domain of scenario B respectively.

```
[Grid]
Positions0 = 0 400 960 1844 2404 2804
Positions1 = 0 400 960 1844 2404 2804
Positions2 = 0 30
Cells0 = 2 28 221 28 2
Cells1 = 2 28 221 28 2
Cells2 = 1
```

(C5.5)

```
[Grid]
Positions0 = 0 400 960 1844 2404 2804
Positions1 = 0 2 442 1002 1402
Positions2 = 0 15 45 60
Cells0 = 2 28 221 28 2
Cells1 = 1 110 28 2
Cells2 = 3 1 3
```

(C5.6)

Keyword Positions allows to subdivide the domain in zones, which are discretized each according to its corresponding keyword Cells (Coltman et al., 2021). Subdivisions are specified for each coordinate direction: for the applied coordinate system, 0 refers to the X-direction, 1 refers to the Y-direction and 2 refers to the Z-direction.

For example, in the code snippet (C5.5), corresponding to scenario A, a single numerical layer of 30 m is defined (Positions2 = 0 30, Cells2 = 1). It is discretized in X and Y directions with a non-uniform grid: very fine in the near well zone (from 960 m to 1844 m, 221 cells are required), fine in the intermediate zones (from 400 m to 960 m as well as from 1844 m to 2404 m, 28 cells are required) and coarse near to the boundaries (from 0 to 400 m as well as from 2404 to 2804 m, only two cells are required).

Conversely, in the code snippet (C5.6), corresponding to scenario B, the vertical domain is subdivided in three zones (from 0 to 15 m, from 15 to 45 m, from 45 to 60 m; Positions2 = 0 15 45 60); the central zone has a single numerical layer but the upper and lower zones are discretized with three layers each (Cells2 = 3 1 3).

ECLIPSE -on the other side- uses the corner-point grid which is widely employed by commercial reservoir simulators for its ability to describe reservoirs with complex geometry (Menezes Farias et al., 2019). However, the cell configuration for the ECLIPSE model of this study was specified using the simple block center description using the keywords DX, DY and DZ due to the simplicity of the model geometry.

5.5 Reservoir and Fluid Properties

Parameters for rock/solid physical and thermal properties are defined under the group/keyword **[Component]** of the input file as shown in the code snippet (C5.7).

```
[Component]
SolidDensity = 2750 # [Kg/m^3]
SolidThermalConductivity = 2.8 # [Watt/m.°K]
SolidHeatCapacity = 800 # [Joule/Kg.°K]
```

(C5.7)

Both porosity and permeability are defined in the spatial parameters file (spatialparams.hh). To assign the porosity and permeability values for scenario B, it is required to recognize where a certain location is with respect to the 3 layers: aquifer layer, caprock layer and bedrock layer. This is because the caprock and bedrock layers will have different porosity and permeability from those of the aquifer layer. For this purpose, two functions have been defined in the spatial parameters file: the function isInBottom() which distinguishes whether the location is inside the bedrock layer and the function isInTop() which distinguishes whether the location is inside the caprock layer where the two functions are given in the code snippet (C5.8).

```
bool isInBottom(const GlobalPosition &globalPos) const
  {
    return globalPos[2] < aquiferBottom_ + eps_;
  }
bool isInTop(const GlobalPosition &globalPos) const
```

(C5.8)

```
  {
    return globalPos[2] > aquiferTop_ - eps_;
  }
```

In the code snippet (C5.8), globalPos[2] refers to the position along the Z-direction, the epsilon variable eps_ has a value of 1e-6, the variable aquiferBottom_ has a value of 15.0 while the variable aquiferTop_ has a value of 45.0 Each of the two functions returns "true" if the inequality condition inside the body of the function is satisfied, otherwise, it returns "false".

Permeability is then defined as the return value of the function permeabilityAtPos() and porosity is defined as the return value of the function porosityAtPos() as illustrated in the code snippets (C5.9) and (C5.10) respectively. The globalPos parameter in both functions refers to the position inside the grid. Both functions work in the same way. Considering the permeabilityAtPos() function for instance; if the value of the globalPos parameter renders the isInBottom() function true, the bedrock permeability value (bottomlayerK_) is returned. If the isInBottom() function turns out to be false, the isInTop() function is checked next where the caprock permeability value (toplayerK_) is returned if it turns out to be true. If both isInBottom() and isInTop() functions are false, the permeabilityAtPos() function will return the aquifer permeability (aquiferK_).

```
PermeabilityType permeabilityAtPos(const GlobalPosition& globalPos) const

  {
    if (isInBottom(globalPos)) {return bottomlayerK_;}
    else if (isInTop(globalPos)) {return toplayerK_;}
    else   {return aquiferK_;}

  }
```

(C5.9)

```
Scalar porosityAtPos(const GlobalPosition& globalPos) const
  {
     if (isInBottom(globalPos)) {return bottomlayerPorosity_;}
    else if (isInTop(globalPos)) {return toplayerPorosity_;}
    else   {return aquiferPorosity_;}

  }
```

(C5.10)

The aquifer permeability based on the direction is defined as follows in the spatial parameters file.

```
// intrinsic permeabilities
    aquiferK_[0][0] = 2.46e-13;  // Permeability along X = 249.26 mD
    aquiferK_[1][1] = 2.46e-13;  // Permeability along Y = 249.26 mD
    aquiferK_[2][2] = 9.87e-14;  // Permeability along Z = 100 mD
```

(C5.11)

such that aquiferK_[0][0], aquiferK_[1][1] and aquiferK_[2][2] correspond to K_{xx}, K_{yy} and K_{zz} respectively which means they are the values of the main diagonal of the permeability tensor. The caprock and bedrock permeabilities are defined in a similar

way with the only difference of having the same permeability value in all directions (isotropic layers). The porosities for the 3 layers are defined as follows where all of them are homogeneous.

```
// porosities
    toplayerPorosity_ = 0.01;
    bottomlayerPorosity_ = 0.01;
    aquiferPorosity_ = 0.1;
```

(C5.12)

The assignment of porosity and permeability values in scenario A is much easier compared to scenario B as there is only one layer which is the aquifer layer and thus there is no need to define the isInBottom() and isInTop() functions. In this case, only the aquifer permeability (aquiferK_) is returned by the permeabilityAtPos() function and the aquifer porosity (aquiferPorosity_) is returned by the porosityAtPos() function.

The fluid system is set to a single liquid phase comprised of only one component which is pure water. This is done by defining the property FluidSystem for the node TTag::OnePNITypeTag in the problem file by a partial template specialization for that property. The type of the property is set inside the struct body to the single liquid phase-single component class template called FluidSystems::OnePLiquid (The DuMux developers, 2020d). The component type is selected as a template argument of that class template and is set to the pure water class named Components::H2O which employs the IAWPS relations to compute the water properties (The DuMux developers, 2020c). This is illustrated in the code snippet (C5.13).

```
template<class TypeTag>
struct FluidSystem<TypeTag, TTag::OnePNITypeTag>
{
  using type = FluidSystems::OnePLiquid<GetPropType<TypeTag,
Properties::Scalar>,
                        Components::H2O<GetPropType<TypeTag,
Properties::Scalar>> >;
};
```

(C5.13)

5.6 Boundary Conditions

The types of boundary conditions are first set using the boundaryTypesAtPos() function which is defined inside the problem file based on the globalPos parameter which indicates the position inside the grid. For the full domain, A Neumann boundary condition is assigned for both the top and bottom boundaries while a Dirichlet boundary condition is assigned for the other boundaries. This is illustrated in the code snippet (C5.14). The epsilon variable eps_ in that code snippet has a value of 1e-6, globalPos[2] refers to the position along the Z-direction and this->gridGeometry().bBoxMax()[2] refers to the Z-coordinate with the maximum value which in this case is 30.0 An if..else statement has been used such that if the value of the globalPos parameter is less than 1e-6 (corresponding to the lower boundary) or higher than 29.999999 (corresponding to the

top boundary), a Neumann boundary condition is assigned; otherwise, a Dirichlet boundary condition is assigned. Holding a floating-point comparison using the epsilon variable is just a more efficient way than writing an equality which might fail.

```
BoundaryTypes boundaryTypesAtPos(const GlobalPosition &globalPos) const
  {
      BoundaryTypes bcTypes;
    if (globalPos[2] < eps_ || globalPos[2] > this-
>gridGeometry().bBoxMax()[2] - eps_)
       bcTypes.setAllNeumann();
    else
       bcTypes.setAllDirichlet();

    return bcTypes;
  }
```

(C5.14)

The same methodology has been applied to set a Neumann boundary condition at the symmetry plane in the half-domain and a Dirichlet boundary condition elsewhere as illustrated in the code snippet (C5.15) where globalPos[1] refers to the position along Y-direction.

```
BoundaryTypes boundaryTypesAtPos(const GlobalPosition &globalPos) const
  {
   BoundaryTypes bcTypes;
     if (globalPos[1] < eps_)
   bcTypes.setAllNeumann();
    else
   bcTypes.setAllDirichlet();

    return bcTypes;
  }
```

(C5.15)

For both the full domain and half-domain, the Dirichlet boundary condition is set using the function dirichletAtPos() inside the problem file to be the same as the initial condition for both pressure and temperature. This is shown in the code snippet (C5.16) in which the dirichletAtPos() function returns the same output as the initialAtPos() function inside which the initial conditions are defined.

```
PrimaryVariables dirichletAtPos(const GlobalPosition &globalPos) const
  {
    return initialAtPos(globalPos);
  }
```

(C5.16)

The Neumann boundary condition is also the same for both the full and half domains given as a no-flow boundary condition. It is set using the function neumannAtPos() in the problem file as shown in the code snippet (C5.17).

```
NumEqVector neumannAtPos(const GlobalPosition &globalPos) const
  {
    NumEqVector values(0.0);
```

(C5.17)

```
    return values;
  }
```

5.7 Initial Conditions

Both the half and full domains were assigned the same initial hydrostatic pressure gradient of 9.7119 KPa/m or 0.097119 bar/m and the same initial single temperature of 50.45 °C (323.6 °K). The imposed hydrostatic pressure gradient corresponds to a water density of 990 Kg/m^3 for a gravitational acceleration of 9.81 m/s^2. Initial conditions for pressure and temperature were set using the initialAtPos() function which is defined inside the problem file as presented in the code snippet (C5.18) illustrating the defined initial conditions for the half-domain. The only difference for the full domain is that the pressure on top of the domain is assigned a value of 100e5 Pascal instead of 98.543215e5 Pascal due to the absence of the 15-meter-thick caprock in the full domain model geometry which makes the top boundary deeper where the aquifer pressure of 101.456785 barsa is the reference pressure. The reason the pressure on top of the domain is followed by a negative sign in the hydrostatic pressure calculation is because the expression this->spatialParams().gravity(globalPos)[2] returns a negative value for the gravitational acceleration.

```
PrimaryVariables initialAtPos(const GlobalPosition &globalPos) const
  {
      PrimaryVariables values(0.0);
      Scalar densityW = 990;  // Kg/m^3
      Scalar depth = this->gridGeometry().bBoxMax()[2] - globalPos[2];
      // Hydrostatic Pressure
      values[pressureIdx] = 98.543215e5 - densityW*this-
>spatialParams().gravity(globalPos)[2]*depth;  //Pascal
      // Temperature
      values[temperatureIdx] = 323.6;  //Kelvin = 50.45 degree celsius

      return values;
  }
```

(C5.18)

5.8 The Well Model

Being a multipurpose simulator, Dumux does not offer the large set of features provided by ECLIPSE for modeling a well. It can be said that well modeling is much easier in ECLIPSE than Dumux due to the fact that ECLIPSE is a very popular simulator which is particularly tailored for oil and gas reservoir simulation problems and has already been in operation by the petroleum industry for several years. ECLIPSE almost mimics physical wells by allowing the user to set various well specifications such as wellhead position, wellbore diameter, bottom-hole intervals open to flow, skin factor, well status, well control mode, flow rate or pressure targets and many others. On

the contrary, Dumux does not consider a true well model but instead point sources are used to model wells. This necessarily means that Dumux doesn't employ the Peaceman well model which is used in ECLIPSE to relate the downhole volumetric flow rate to the difference between the grid block pressure and the bottom-hole flowing pressure. For this reason, the Dumux pressure results in this study were compared against the well-block pressure (WBP) in ECLIPSE and not against the well bottom-hole pressure (WBHP). It should be noted however that the Peaceman well model can be programmed inside the Dumux code if needed thanks to the flexibility deriving from the open-source and C++ code-based nature of Dumux.

In the Dumux model, solution-dependent point sources located at the center of the aquifer vertical thickness were used to depict the production and injection wells. The X, Y and Z coordinates of the point sources in meters were first entered inside the addPointSources() function in the problem file (problem.hh) as shown in the code snippets (C5.19) and (C5.20) representing scenario A and scenario B respectively

```
void addPointSources(std::vector<PointSource>& pointSources) const
  {
      // The injection well (source term)
      pointSources.push_back(PointSource({1342, 1402, 15}));

      // The production well (sink term)
      pointSources.push_back(PointSource({1462, 1402, 15}));
  }
```

(C5.19)

```
void addPointSources(std::vector<PointSource>& pointSources) const
  {
      // The injection well (source term)
      pointSources.push_back(PointSource({1342, 0, 30}));

      // The production well (sink term)
      pointSources.push_back(PointSource({1462, 0, 30}));
  }
```

(C5.20)

After that, the volumetric production and injection rates (volumeSource) were provided in m^3/s inside the pointSource() function such that the rates imposed in scenario B are always half of those imposed in scenario A due to the applied symmetry condition in scenario B. Inside the same function, the mass rate (massSource) is then calculated via multiplying the pre-defined volumetric rate by the water density of the control volume at a specific time step based on the pressure and temperature of the control volume. It's clear that the temperature is only known in the case of injection since an injection temperature of 20 °C (293.15 °K) is imposed but it will be solution-dependent in the case of production. On the other hand, the pressure of each of the

injection and production cells/control volumes will be dependent on the imposed well rate and thus solution-dependent. The mass rate in Kg/s is then multiplied by the water enthalpy in J/Kg evaluated by Dumux at the existing pressure and temperature of the control volume to obtain the energy source/sink terms (energySource) in J/s. Hence in Dumux, heat is provided or removed from the aquifer through the water enthalpy of the control volume which is evaluated in time. It's worth noting that the injection/production rates provided inside the pointSource() function in Dumux are essentially downhole rates because the point sources are already located inside the aquifer layer. As a result, the water compressibility in the ECLIPSE model was set equal to zero and the water formation volume factor was assigned a value of 1 in order to guarantee that the surface rates applied in the ECLIPSE model will yield the same downhole rates as those of Dumux. The code snippet below shows the pointSource() function used in the Dumux code of scenario A for a simulation run with 100 m^3/day for both injection and production rates.

```
template<class ElementVolumeVariables>

  void pointSource(PointSource& source,
                   const Element &element,
                   const FVElementGeometry& fvGeometry,
                   const ElementVolumeVariables& elemVolVars,
                   const SubControlVolume &scv) const
{
    const auto& pos = source.position();
    const auto& volVars = elemVolVars[scv];
    if (pos[0] < 1350.0)  // Injection
    {
     const Scalar volumeSource = 1.157407407e-3; // Injection rate is positive
& in m^3/s
     const Scalar massSource =
volumeSource*IapwsH2O::liquidDensity(293.15, volVars.pressure(0));

     const Scalar energySource =
massSource*IapwsH2O::liquidEnthalpy(293.15, volVars.pressure(0));

     source = NumEqVector({ massSource, energySource });
    }

    else // production
    {
     const Scalar volumeSource = -1.157407407e-3; // Production rate is
negative and in m^3/s
      const Scalar massSource = volumeSource*volVars.density(0); // using
existing water density of the control volume
     const Scalar energySource = massSource*volVars.enthalpy(0); // using
existing water enthalpy of the control volume
```

(C5.21)

```
        source = NumEqVector({ massSource, energySource });
      }
}
```

It can be observed from the code snippet (C5.21) that distinction is made between the injection point source and the production point source according to their positions along the X-direction by using the if…else statement. It is also pointed out that according to the Dumux convention, the production rate assumes a negative value while the injection rate assumes a positive value.

In order to calculate the point sources, the function computePointSourceMap() must be called inside the main function of the program in the (main.cc) file as shown in the code snippet (C5.22)

```
problem->computePointSourceMap();
```
(C5.22)

5.9 Solution Strategy and Solvers

A fully implicit solution scheme is employed in Dumux as the solution strategy for the non-isothermal flow problem of this study. This results in a coupled system of non-linear equations which has to be solved simultaneously and iteratively. It can be noticed here that the non-linearity is not that high in the pressure equation (2.3) since water density and viscosity are not strong functions of pressure. However, the coupling between the two governing equations (2.3) and (2.8) is mainly due to the presence of water viscosity in the pressure equation (2.3), which is strongly affected by temperature, and the presence of pressure itself as a primary variable in the heat transport equation (2.8). This explains why the monolithic solution of this coupled system of non-linear equations may be favored. Fully implicit methods are generally preferred in the oil industry for their unconditional stability which permits the possibility of larger time stepping (Moortgat, 2017). However, they are not completely advantageous since they cause a higher memory drainage and are more costly from the computational point of view especially in the case of several coupled unknowns (Moortgat, 2017).

ECLIPSE -on the contrary- does not solve the heat transport equation simultaneously with the pressure equation but instead, the heat transport equation is solved after the convergence of a timestep has already been reached where the mesh cell temperatures are then updated (Schlumberger, 2017).

5.9.1 Non-linear Solver

Newton's iterative method is used in Dumux to solve the non-linear system of equations. The method was suggested by Newton in 1669 and it is based on the concept of linearization (Polyak, 2007). Considering the balance equation (5.11), an initial guess is made of the solution L after which the residual R(L) which is equal to the output of the left hand side of equation (5.11) is calculated. To reduce the error, the Jacobian matrix J(L) which corresponds to the derivative of the residual with respect to the

solution is then computed (The DuMux developers, 2021). In Dumux, numeric differentiation was used for the calculation of the Jacobian matrix as observed in the second argument of the FVAssembler class in the code snippet (C5.3). The following equation adapted from (Flemisch & Class, 2019) is then used to obtain the solution for the new iteration L_{k+1}

$$J(L_k)(L_{k+1} - L_k) = -R(L_k) \tag{5.13}$$

where $k + 1$ refers to the current Newton iteration while k refers to the previous Newton iteration.

The procedure proceeds till convergence between two subsequent iterations is reached and thus the solution is obtained for the new time step. In Dumux, one can choose between a maximum relative shift or an absolute residual criterion for convergence where the shift criterion represents the maximum allowable difference between the values of a primary variable for two subsequent iterations whereas the residual criterion sets a minimum threshold that the absolute residual has to be below for convergence to be declared (The DuMux developers, 2020k). The user can also ask for both criteria to be met. In this study, a maximum relative shift criterion was used and was set equal to 1e-5 in all the simulation runs as seen in the code snippet (C5.23) from the parameters file (params.input) under the group **[Newton]**.

```
[Newton]
MaxRelativeShift = 1e-5
```
(C5.23)

The Newton method is the approach also used by ECLIPSE for linearizing and solving the system of non-linear equations. Additionally, ECLIPSE applies a convergence criterion that is dependent on a maximum residual (Schlumberger, 2017)

5.9.2 Linear Solver

Solving a linear system may take a lot of time and thus the performance of the non-linear solver may be judged based on the speed of the linear solver (Chen et al., 2009). It is further pointed out by Chen et al., (2009) that an ideal situation occurs when a non-linear solver which exhibits fast convergence is used along with a linear solver which consumes less CPU time. The solver Dumux::AMGBiCGSTABBackend was selected in this study to solve the linearized simultaneous equations as shown in the code snippet (C5.24) from the main function (main.cc file).

```
using LinearSolver =
AMGBiCGSTABBackend<LinearSolverTraits<GridGeometry>>;
```
(C5.24)

This solver is based on the biconjugate gradient stabilized method (BiCGSTAB) and the AMG preconditioner (The DuMux developers, 2020b). BiCGSTAB is an iterative algorithm used to solve large asymmetric linear systems (Ocłoń et al., 2013). However, preconditioning techniques are needed by iterative solvers to enhance their effectiveness (Ocłoń et al., 2013); preconditioning converts the initial linear system into a new one which can be solved more efficiently. The AMG preconditioner is based on the

algebraic multigrid method which is originally an iterative method used to solve large-scale linear systems (Boyle et al., 2010).

ECLIPSE -on the other side- uses the ORTHOMIN solver which is an iterative method developed mainly for solving sparse linear systems within the domain of reservoir simulation (Schlumberger, 2017; Vinsome, 1976). Furthermore, preconditioning in ECLIPSE is performed by Nested Factorization (Schlumberger, 2017).

5.10 Barriers

The main purpose of introducing the caprock and bedrock in scenario B was to investigate whether the heat conduction owing to the presence of the caprock and bedrock layers will have a significant impact on the temperature distribution in the geothermal aquifer and to check if the heat conduction phenomenon will be modeled similarly by both simulators Dumux and ECLIPSE. As a result, it can be identified whether the added computational cost of modeling the caprock and bedrock due to the use of a larger mesh size is justified or can simply be avoided without seriously affecting the simulation results. Heat conduction effect is also present in scenario A due to the contact between the geothermal water and the solid matrix inside the porous aquifer. However, in this case, the effects of convection and conduction cannot be split from one another. Moreover, the heat convection effect will be more dominant for scenario A especially over short durations. For this reason, it was of interest to introduce the two scenarios A and B such that in scenario B, pure solid conduction can be monitored and the effect of incremented solid conduction due to the intercommunication between the water and the caprock and bedrock layers can be well-judged.

Among the aquifer and the caprock and bedrock layers, only heat exchange is allowed. Consequently, mass flux has to be inhibited. This was simply applied in ECLIPSE by setting zero transmissibility values in the Z-direction at the interfaces separating the aquifer layer from the caprock and bedrock layers. The value of the Z-direction transmissibility is controlled in ECLIPSE via the keyword TRANZ after defining a box for the numerical layer just above the requested interface. The implementation in ECLIPSE is illustrated in the snippet (C5.25) from the EDIT section in the ECLIPSE input file for scenario B.

```
EDIT

BOX
1 281 1 281 3 3/

TRANZ
78961*0/
```

(C5.25)

```
BOX
1 281 1 281 4 4/

TRANZ
78961*0/
```

Dumux -on the other hand- does not offer such easy implementation for imposing zero mass flux of water across the layer interfaces as in ECLIPSE. However, Dumux is based on the concept of modularity which allows the substitution of a simulation element with another without major changes applied to the application code (Koch, Gläser, et al., 2020). The modularity feature thus provides further flexibility for the modeler to tailor a Dumux problem/application according to the desired needs. The open-source nature of Dumux together with its modular-based structure were exploited in order to make the Dumux problem comparable to the ECLIPSE one in terms of mass flux restriction across the interfaces. This was executed by modifying the Darcy law C++ class implemented in the Dumux core. The procedure was to copy the original Darcy law file into the directory of the Dumux problem for scenario B, rename it to modifieddarcy.hh, change the name of the Darcy law class inside the file to ModifiedDarcy, introduce a transmissibility factor in the problem file to hold a zero value at the aquifer interfaces with the caprock and bedrock and a value of one elsewhere, multiply the flux in the modified Darcy file (modifieddarcy.hh) by the introduced transmissibility factor and finally change the AdvectionType property in the problem file by setting it to the new modified Darcy class.

The main idea behind the modification is to define a transmissibility factor inside the problem class (OnePNIProblem) such that the value of this factor depends on the position inside the grid. The transmissibility factor will assume a value of zero when the Z-coordinate of the center of a sub-control volume face is equal to 15 or 45 which are the Z-coordinates for the lower and upper interfaces respectively. Otherwise, the transmissibility factor will assume a value of 1. Within the scope of our problem, the condition resulting in a zero value of the transmissibility factor will be true for the horizontal sub-control volume faces located at the interface between the aquifer and each of the caprock and bedrock layers. The code snippet (C5.26) shows the definition of the pre-discussed transmissibility factor as a function called transmissibilityFactor().

```
Scalar transmissibilityFactor(SubControlVolumeFace scvf) const
  {

  if (Dune::FloatCmp::eq(scvf.center()[2], 45.0)) {return 0.0;}
  else if (Dune::FloatCmp::eq(scvf.center()[2], 15.0)) {return 0.0;}
  else {return 1.0;}


  }
```

(C5.26)

After that, the value returned by the transmissibilityFactor() function based on the position will be accessed inside the ModifiedDarcy class and assigned to a variable

called tFactor where this variable will be multiplied by the flux across a sub-control volume face. This will result in a zero flux across the horizontal sub-control volume faces coinciding with the interfaces between the aquifer and the caprock and bedrock layers. Consequently, no mass flux of water is allowed to pass from the aquifer layer into the caprock or bedrock. The code snippet (C5.27) illustrates the modification inside the flux function in the ModifiedDarcy class where an object of the problem class is used to access the value of the transmissibilityFactor() function.

```
    const auto tFactor = problem.transmissibilityFactor(scvf);

    if (fluxVarsCache.usesSecondaryIv())
      return flux_(problem, fluxVarsCache,
fluxVarsCache.advectionSecondaryDataHandle(), phaseIdx)*tFactor ;
    else
      return flux_(problem, fluxVarsCache,
fluxVarsCache.advectionPrimaryDataHandle(), phaseIdx)*tFactor ;
  }
```

(C5.27)

As a final step, the AdvectionType property in the (problem.hh) file is set to the modified Darcy class (ModifiedDarcy) as shown in the code snippet (C5.28)

```
// Set Advection type
template<class TypeTag>
struct AdvectionType<TypeTag, TTag::OnePNITypeTag> { using type =
ModifiedDarcy<TypeTag, DiscretizationMethod::ccmpfa>; };
```

(C5.28)

A preliminary comparison was made between two cases for scenario B in Dumux where in the first case no modification of the Darcy law class was done while in the second case, the modified Darcy law class was employed in the code. The purpose was to check whether such a modification is necessary or the limited advective flux from the aquifer into the caprock and bedrock layers due to the permeability contrast will not affect the simulation outcomes. The results are presented in Figure 5.5 in the form of pressure color maps for a zoomed front view along the X-direction at Y=0 m where the simulated duration was only 15 days with production and injection rates of 50 m^3/day.

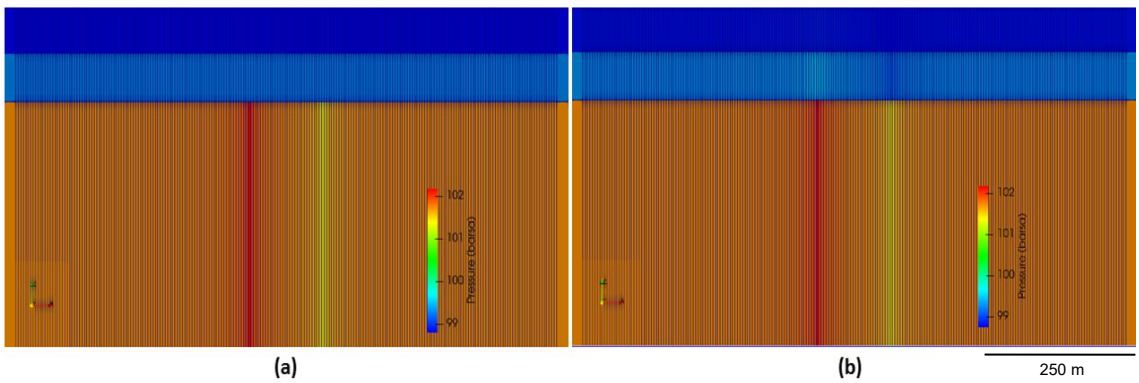


Figure 5.5: Pressure color maps for a zoomed front view along X-direction at Y=0 m for scenario B for (a) modified Darcy law class implemented and (b) no modification of Darcy law class

It can be observed from Figure 5.5b that for the case where no modification was considered for the Darcy law class in the Dumux core, a small pressure change can be observed in the numerical boundary layer just above the aquifer layer, which can be attributed to a not null advective fluid flux from the geothermal aquifer into the caprock/bedrock. Conversely, after the modified Darcy law implementation (Figure 5.5a), the pressure change is no longer seen and thus the Dumux problem for scenario B in this case becomes comparable to the one implemented in ECLIPSE. Consequently, the modification of the Darcy class in the Dumux core for scenario B was necessary to isolate the pure heat conduction component and be able to compare it against the one simulated by ECLIPSE.

6 RESULTS AND DISCUSSION

Three different simulation runs have been performed for both scenarios A and B in the two simulators Dumux and ECLIPSE. The details of those runs are stated in Table 6.1

Table 6.1: Details of the performed simulations

Simulation ID	Scenario	Duration, days	Imposed maximum time step, days	Injection and Production Rates, m^3/day
1-A	Scenario A	180 (~ 6 months)	1	100
1-B	Scenario B			
2-A	Scenario A	3600 (~ 10 years)	30	100
2-B	Scenario B		90	
3-A	Scenario A	3600 (~ 10 years)	30	1000
3-B	Scenario B		90	

The simulation 2-A can be considered as the base case run. The simulation 2-B introduces the caprock and bedrock to the model geometry of the base case to study the sensitivity of the simulation variables to the caprock and bedrock inclusion into the model. The main purpose of the two runs 1-A and 1-B is to investigate the effect of changing the maximum allowed time stepping on the simulation outcome of Dumux. It should be pointed out here that Dumux will always start with a small time step of 10000 seconds for all the runs due to the higher variability of variables in the first time steps and then the time step increases as the convergence becomes easier till the time step reaches its imposed maximum value. The two runs 3-A and 3-B were performed in order to examine whether using a much higher rate (10 times the base case) would induce noticeable differences between the simulation results of Dumux and ECLIPSE. Another reason for performing the high rate run was to examine the impact of high injection and production rates on the behavior of the simulation variables. The two runs 2-B and 3-B were assigned a maximum time stepping that is a little higher (90 days) in comparison to the corresponding runs 2-A and 3-A (30 days) so as to save the computational time in the Finite-Volume-based Dumux due to the larger number of

cells (277347 cells) in the mesh of the half-domain of scenario B compared to the mesh of the full domain of scenario A (78961 cells).

The variables addressed in the simulation results are pressure and temperature in addition to water viscosity as a secondary variable. The 3D visualization software FloViz created by Schlumberger oilfield services company was used to extract the spatial distribution of the ECLIPSE results. As for the Dumux results, all the post-processing workflow was carried out using the open-source visualization software ParaView (ParaView, 2021). Paraview was mainly developed by Kitware company specialized in software solutions in addition to other collaborating governmental and academic bodies (Kitware, 2021; ParaView, 2021). ParaView was employed to visualize the Dumux results both temporally and spatially and extract them. Several ParaView filters were utilized in the post-processing phase including:

- Calculator filter: it was used to convert the Dumux units to the equivalent units used in ECLIPSE such as from Pa.s to cp for water viscosity.
- Transform filter: it was used to stretch the Z-direction by 10 times for an enhanced visualization of cross-sections due to the much smaller vertical extent of both the full and half domains compared to the lateral one.
- Text filter: it was used for labeling the production and injection wells.
- Slice filter: it was used for taking cross-sections at different positions in the computational domain.
- Extract Selection filter: it was needed for extracting the aquifer layer and removing the caprock and bedrock from the cross-sections applied to the half-domain so that only the cells of the aquifer layer can be selected. This is a required step for the spatial plotting of variables along the considered cross-section of the aquifer layer at a selected time step.
- Plot Data filter: it was used for the spatial plotting of variables along the cross-section of the aquifer layer at a certain time step after the cells of the aquifer layer have been selected. This also allows the extraction of such data through the SpreadSheet View capability in ParaView.
- Plot Selection Over Time filter: It was used for the temporal plotting of a certain variable for a certain cell such as plotting the injection pressure of the injection cell over time.

6.1 Base Case Simulation – Aquifer Only (Run 2-A)

This is the base case simulation run which means that neither caprock nor bedrock were considered; so only the aquifer layer was modeled. As a result, the effect of additional thermal conduction due to the interaction between the geothermal water and both the caprock and bedrock was completely neglected.

A top view of the computational domain reveals the progression of the cooled-water thermal front in time as shown in Figure 6.1 where the status of the thermal front is visually checked after (a) 1 year and (b) 10 years from the start of injection.

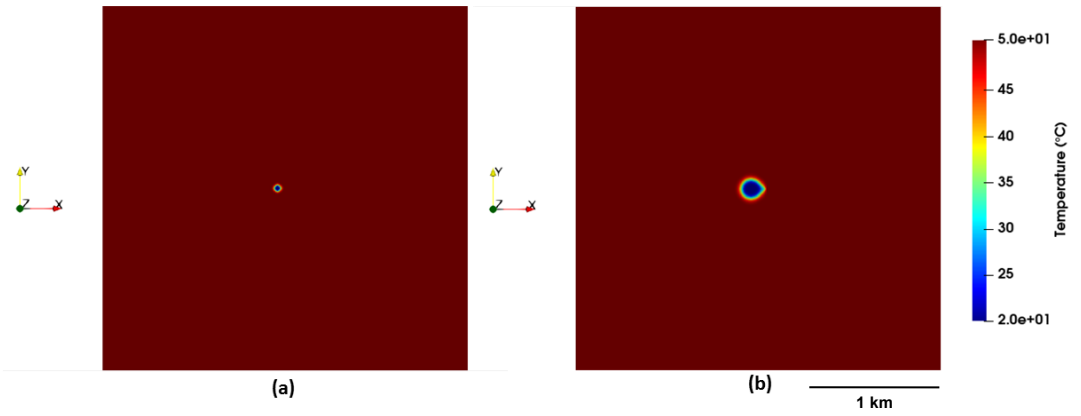


Figure 6.1: Top view for temperature distribution after (a) 1 year and (b) 10 years

Corresponding to the cooled zone that forms around the injection well and grows in time, a high-water-viscosity zone can be observed as illustrated in Figure 6.2

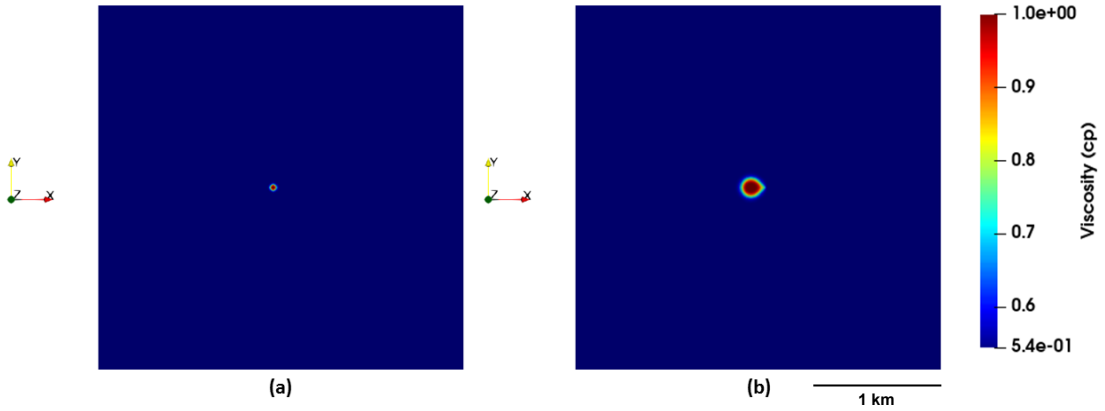


Figure 6.2: Top view for viscosity distribution after (a) 1 year and (b) 10 years

The color maps for pressure distribution on top of the domain clearly depict the production well with its drainage area and the injection well with its area of influence as illustrated in Figure 6.3.

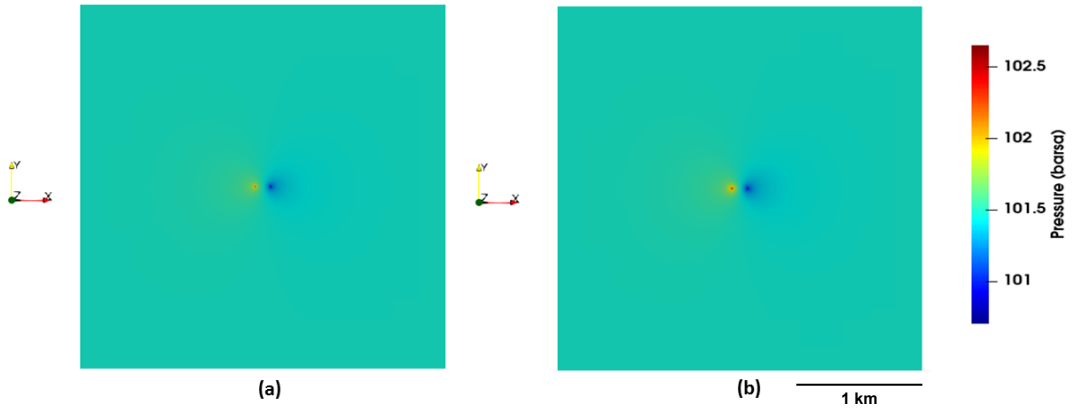


Figure 6.3: Top view for pressure distribution after (a) 1 year and (b) 10 years

A top view zoom-in on the thermal front position with respect to the wells after (a) 1 year and (b) 10 years can be seen in Figure 6.4. Furthermore, for an enhanced

visualization of the behavior of the different variables relative to well locations, a cross-section is taken along the X-direction at the middle of the Y-direction (Y=1402 m) where the wells are situated. Color maps for temperature, pressure and viscosity distributions over the considered cross-section are shown in Figure 6.5, Figure 6.6 and Figure 6.7 respectively considering (a) 1 year and (b) 10 years of the operation of the geothermal doublet.

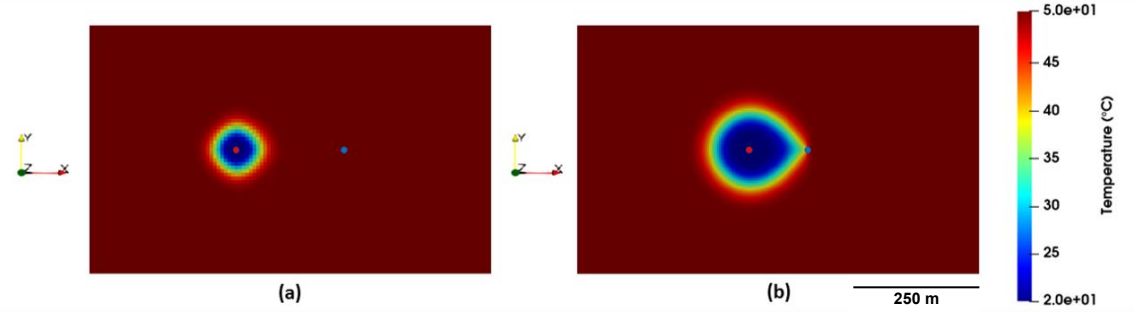


Figure 6.4: Zoomed top view for thermal front position relative to the wells for the run 2-A after (a) 1 year and (b) 10 years.

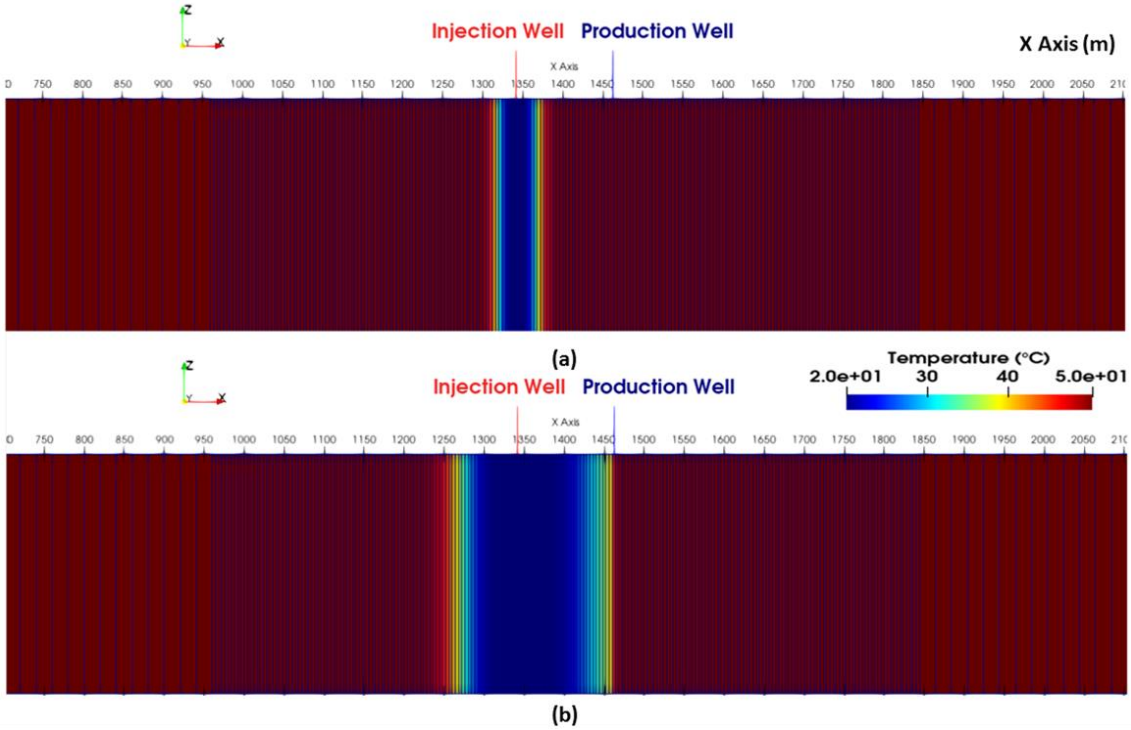


Figure 6.5: Color map for temperature distribution over the cross-section along the X-direction at the plane joining the two wells (Y=1402 m) for the run 2-A after (a) 1 year and (b) 10 years

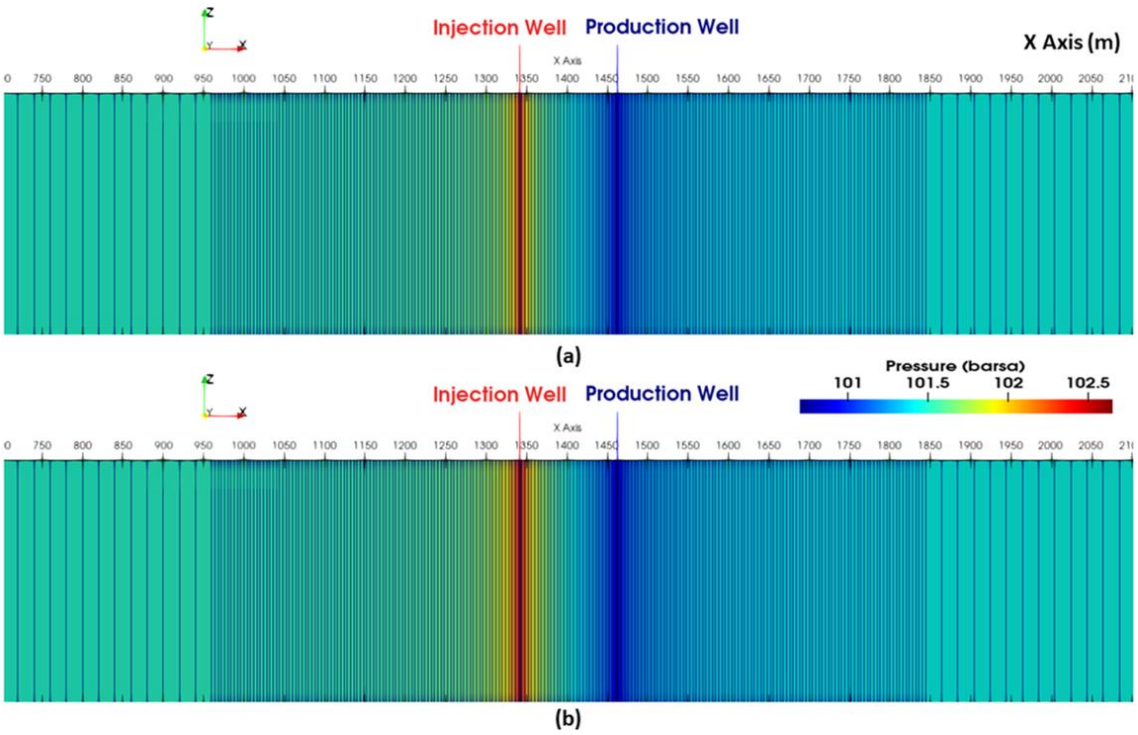


Figure 6.6: Color map for pressure distribution over the cross-section along the X-direction at the plane joining the two wells (Y=1402 m) for the run 2-A after (a) 1 year and (b) 10 years

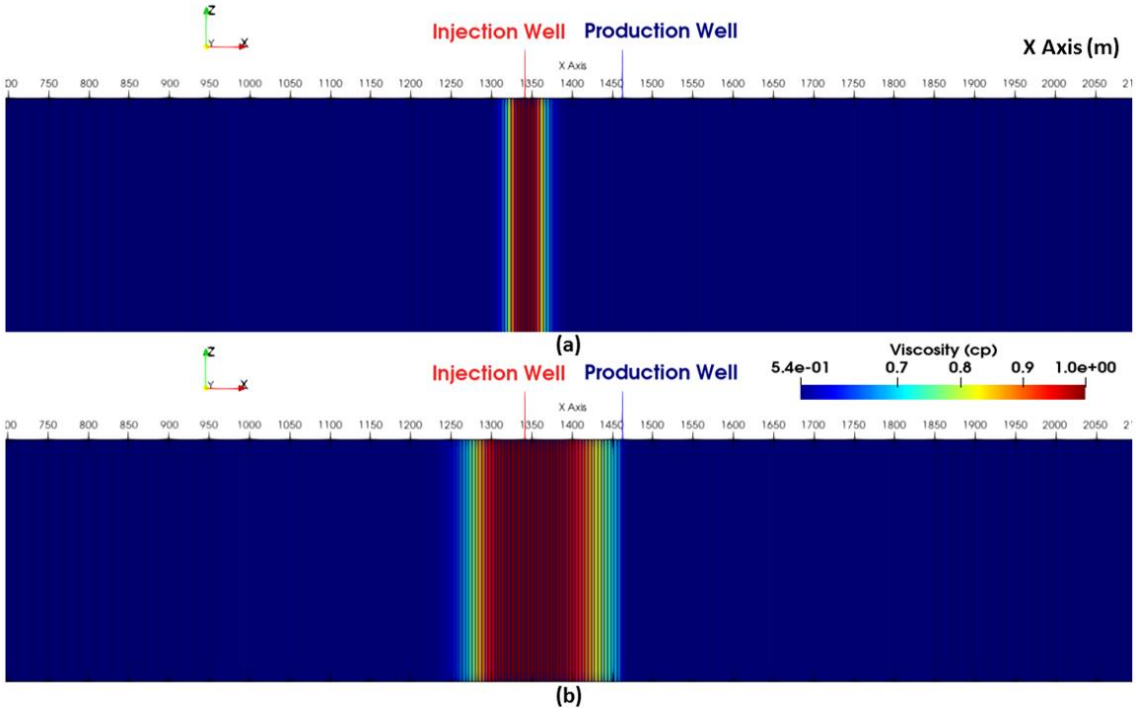


Figure 6.7: Color map for viscosity distribution over the cross-section along the X-direction at the plane joining the two wells (Y=1402 m) for the run 2-A after (a) 1 year and (b) 10 years

It is seen from Figure 6.5 how the cooled-water thermal front has advanced through the geothermal aquifer layer over time. However, it can be noted from Figure 6.4b and Figure 6.5b that the thermal front has exhibited a slightly asymmetrical distribution around the injector well where it has migrated faster on the side where the production well is located. This is mostly due to the depletion effect of the production well which arises from the pressure difference between the producing pressure and the reservoir pressure. The depletion effect induced by the production well leads to a higher Darcy velocity of geothermal water in the vicinity of the production well as proposed by the Darcy law in equation (2.2). This in turn leads to higher advective heat transfer as suggested by the advective/convective heat transfer term in equation (2.6). As a result, the cold thermal energy is transmitted faster on the side of the injection well where the extraction well is acting. However, it can be observed from the color grades in Figure 6.5b that even after 10 years of the simulated operation of the geothermal system, the cooled-water thermal front itself, i.e. the front characterized by a temperature equal to the injected value, has not reached the production well yet. In any case, the production well is clearly affected by the front-influenced zone of reduced temperature that is about to be swept by the thermal front.

Figure 6.7 presents the viscosity color maps that are in full agreement with the corresponding temperature color maps in Figure 6.5 such that the reduced temperature corresponds to a higher water viscosity. This is because of viscosity being a strong function of temperature. The asymmetry of the front is minimum in the beginning of the simulation and increases with time as realized by visually comparing Figure 6.4a to Figure 6.4b or Figure 6.5a to Figure 6.5b. or even Figure 6.7a to Figure 6.7b. As a result, further studies will be needed to verify if a well test monitoring could yield a reasonable average value of the distance of the thermal front from the injection well. In fact, a well test interpretation would assume a radial composite model, considering an inner zone of high water viscosity (worse mobility) and an outer zone of low water viscosity (enhanced mobility). However, such an approach may be hindered by the asymmetric distribution of the thermal front with respect to the injection well due to the presence of the water producer.

In Figure 6.6, it can be observed that the two pressure color maps (a) and (b) do not exhibit significant differences. The similarity suggests that the steady state for pressure distribution in the geothermal aquifer has already been reached before the 1-year time point considered in Figure 6.6a. This means that the pressure disturbance propagates much faster than the thermal disturbance. Considering the current simulation run (2-A), it was found out that steady state was reached before the first 30-day time step printed out by Dumux. This was verified by observing a pressure change at the coarse boundary cells in X and Y directions, followed by a constant pressure value till the end of the simulation time.

6.2 Sensitivity to Maximum Time Step Size

An investigation was carried out to verify whether using a different imposed maximum time step will have a significant impact on the outcome of Dumux

simulations. For this purpose, a comparison was held between the output of the simulation run 2-A employing an imposed maximum time step of 30 days and that of the simulation run 1-A employing an imposed maximum time step of only 1 day. It should be stressed here that Dumux will have the same initial time step size of 10000 seconds for both simulations where this value can be set by the user in the input file through the parameter DtInitial under the group **[TimeLoop]**. Dumux thus starts with a time step size for both simulation runs that is relatively small. The time step progression afterwards will show a gradual automatic increase of the time step size such that both simulation runs 2-A and 1-A will have equal time step sizes during the initial time steps. The increase of the time step size will continue till the imposed maximum value is reached. Smaller time step sizes are used in the beginning of the simulation because of the large variation threshold of the simulation variables during the first few time steps. It can be understood that despite the different imposed maximum time step sizes for both simulation runs 2-A and 1-A, the initial time steps characterized by a significant changeability of the variables will be treated by the simulator similarly in both cases.

The results generated by Dumux for simulation runs 2-A and 1-A were compared in terms of temperature and pressure distributions along the X-direction over the plane joining the two wells (Y=1402 m) at 3 and 6-month simulation times as shown in the line plots in Figure 6.8 and Figure 6.9 respectively. A very good match was obtained between the simulation results of both runs as noted from the two figures. Analogous results were also obtained by comparing the B scenario simulation runs (1-B and 2-B). This clearly indicates that the use of a different maximum time step size did not affect the integrity of the results modeled by Dumux.

A possible factor contributing to the consistency of the Dumux results in spite of the different time stepping employed for the largest part of the simulation duration for both cases (1-A vs. 2-A) and (1-B vs. 2-B) is the fully implicit solution scheme selected for the Dumux model of this study. The unconditional stability of this scheme enables the use of a larger time stepping without affecting the solution convergence.

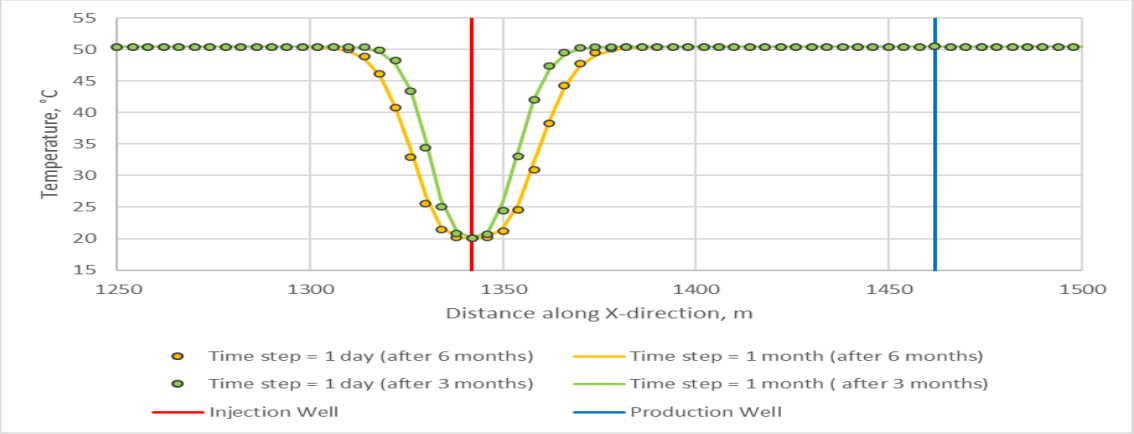


Figure 6.8: Dumux line plot comparison between the simulation runs 2-A and 1-A of temperature distribution along the X-direction over the plane joining the two wells (Y=1402 m)

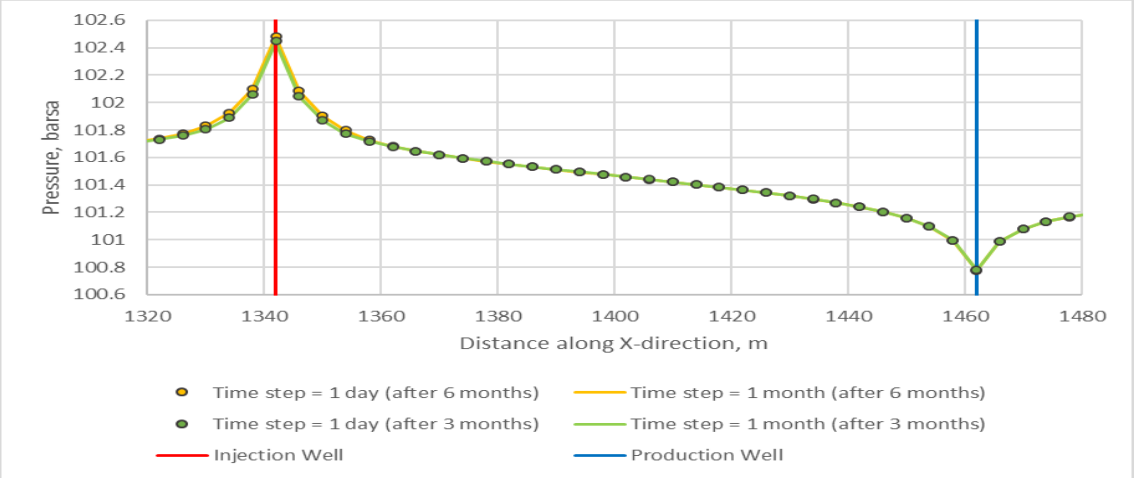


Figure 6.9: Dumux line plot comparison between the simulation runs 2-A and 1-A of pressure distribution along the X-direction over the plane joining the two wells (Y=1402 m)

6.3 High Rate Impact

The response of the simulation variables was checked for the case when markedly higher injection and production rates were imposed for the geothermal doublet. The same volumetric flow rate of 1000 m^3/day (10 times the base case) was used for both injection and production.

A comparison is made between (a) run 3-A and (b) run 2-A (base case) in terms of the color map depicting temperature distribution on the top of the domain after a 10-year simulation duration as presented in Figure 6.10. The high rate impact on the temperature behavior is further analyzed by two more comparisons: comparison of the color maps of the same runs for temperature distribution over the plane joining the two wells (Y=1402 m) after a 10-year simulation as shown in Figure 6.11 and comparison of zoomed top views of both runs for the thermal front position relative to the wells after the same duration as illustrated in Figure 6.12.

Top views of Figure 6.10 and Figure 6.12 as well as the cross sections of Figure 6.11 clearly show that after a 10-year simulation, the cooled zone around the injection well of the run 3-A has a much larger lateral extension compared to the corresponding zone of the base case run 2-A. This is due to the considerably higher injection rate employed in the run 3-A which in turn corresponds to a higher Darcy velocity of the injected water and consequently a higher advective heat transfer as implied by the convective term in the heat transport equation (2.8). As a result, the cooled-water thermal front travels much faster through the geothermal aquifer so that at a certain point in time after the start of injection, the area covered by the thermal front of the run 3-A will be larger than that covered by the front of the run 2-A.

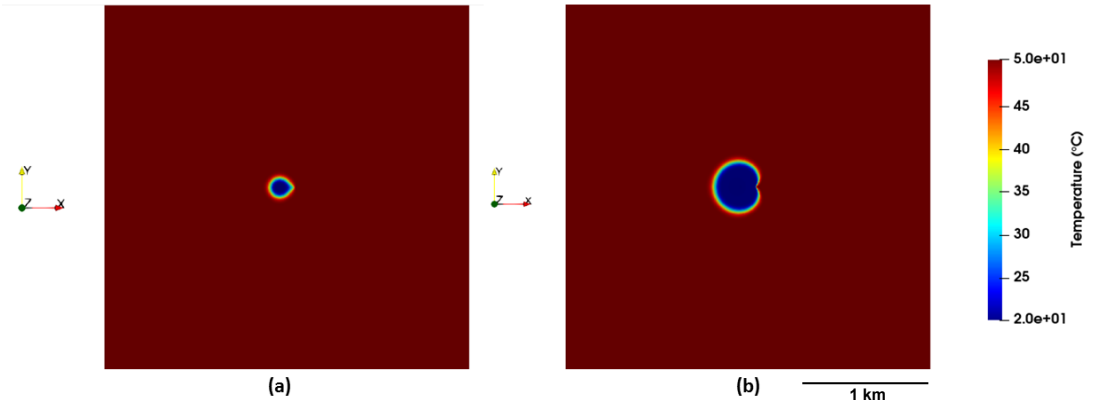


Figure 6.10: Top view for temperature distribution for (a) run 2-A and (b) run 3-A after a 10-year simulation.

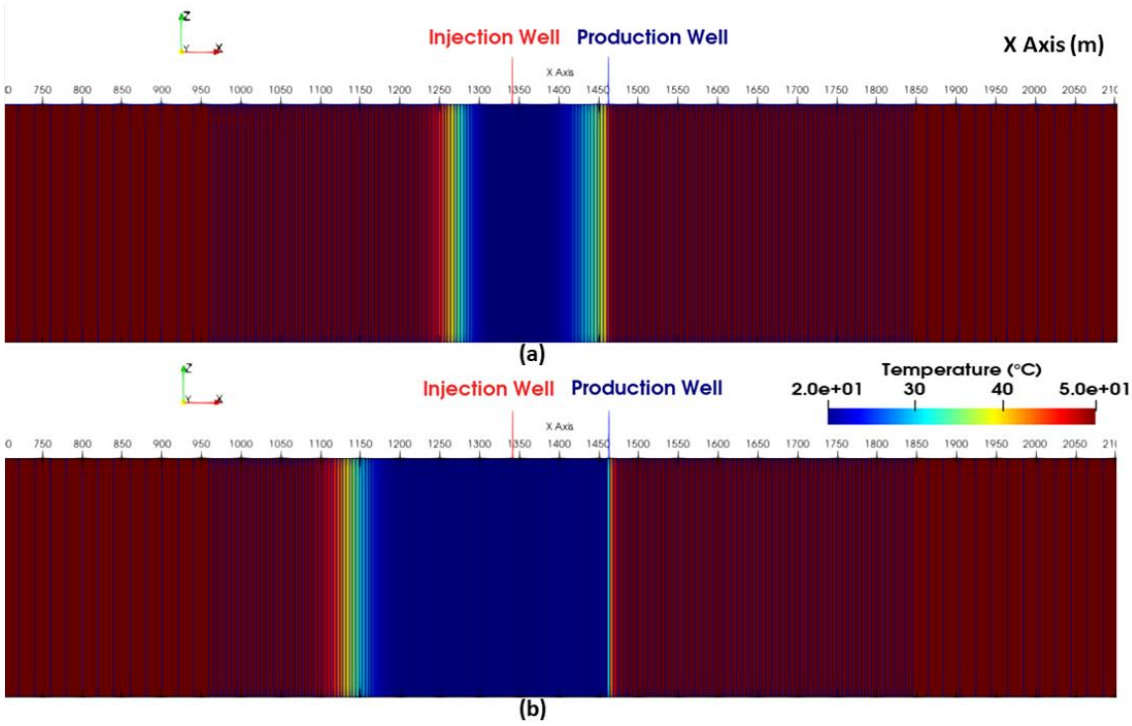


Figure 6.11: Color maps for temperature distribution over the plane joining the two wells for (a) run 2-A and (b) run 3-A after a 10-year simulation duration.

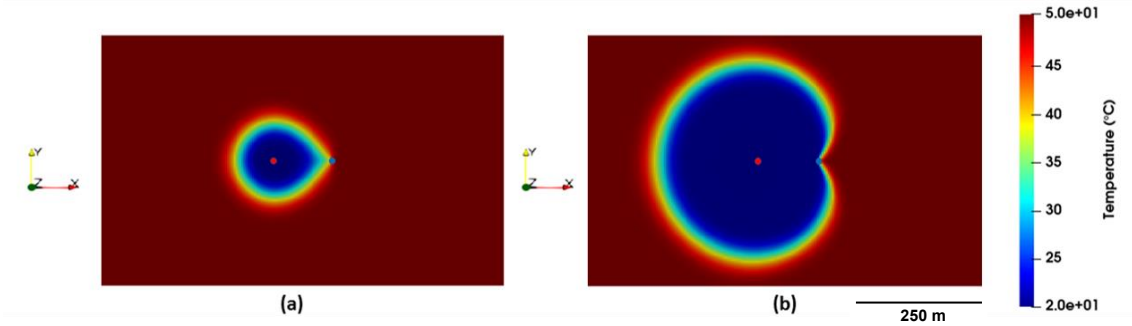


Figure 6.12: Zoomed top view for thermal front position relative to the wells for (a) run 2-A and (b) run 3-A for a 10-year simulation.

It can be realized from both Figure 6.11b and Figure 6.12b that the thermal breakthrough has already taken place thanks to the faster propagation of the cooled-water thermal front through the aquifer due to the higher imposed injection and production rates. The higher production rate assists the faster occurrence of the thermal breakthrough by creating a higher depletion effect due to the much lower production pressure compared to that of the base case 2-A which in turn means a higher pressure difference between the producing pressure and the aquifer pressure. Since the production well acts as a sink for both the mass and cold energy of the geothermal water, the thermal front cannot propagate beyond the production well as observed from both Figure 6.11b and Figure 6.12b. However, the thermal front continues to propagate on the other side of the injection well where no sinks exist such that it reaches a lateral extent notably larger than that of the base case run 2-A. As a consequence, run 3-A after 10 years still shows a slightly asymmetric thermal front (Figure 6.12b), but the shape differs from the one obtained for run 2-A (Figure 6.12a).

The pressure response to the higher injection and production rates was also investigated by holding a line plot comparison between the two runs 3-A and 2-A in terms of the pressure distribution over the plane joining the two wells after a 10-year simulation. The results are presented in Figure 6.13 where it is evident that the higher flow rate (1000 m^3/day) used for both injection and production in the 3-A run results in an injection pressure that is significantly higher than the corresponding pressure in the 2-A case and a production pressure that is significantly lower. As a result, the pressure variation range has become wider than that of the 2-A case.

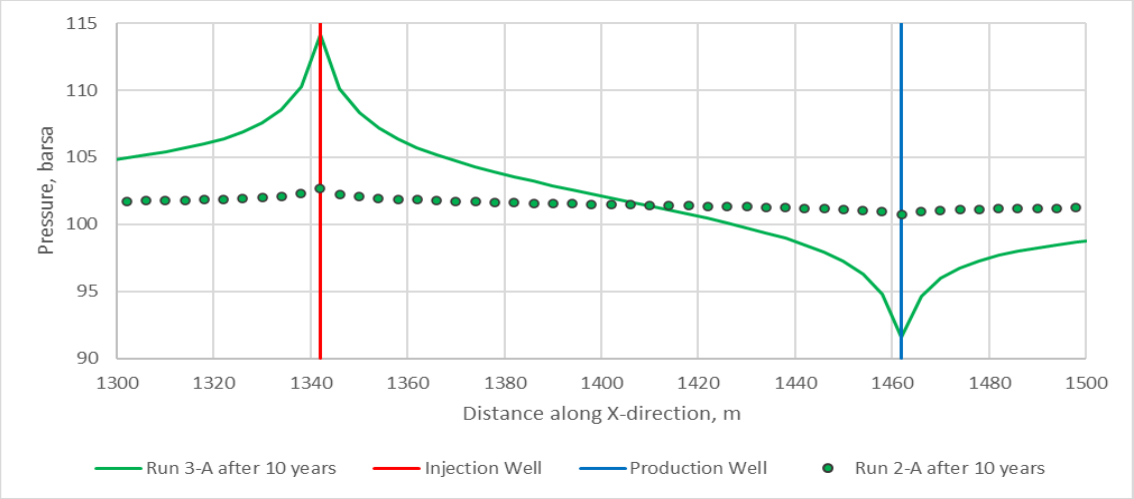


Figure 6.13: Line plot comparison between run 3-A and run 2-A in terms of pressure distribution over the plane joining the two wells (Y=1402 m) after a 10-year simulation

6.4 Effect of Caprock and Bedrock Conduction

It is of interest to analyze the impact of caprock and bedrock thermal conduction on the simulation outcome. To this end, caprock and bedrock were included into the geometry of the numerical model for runs 1-B, 2-B and 3-B.

In order to verify the impact of caprock and bedrock thermal conduction on the asymmetry of the cooled-water thermal front, a comparison was held between the two simulation runs (a) 2-A and (b) 2-B as well as the two runs (a) 3-A and (b) 3-B in terms of zoomed top view of the thermal front position relative to well locations after 10 years as illustrated in Figure 6.14 and Figure 6.15 respectively. For a general understanding of the influence of caprock and bedrock conduction on the temperature behavior and distribution in the geothermal aquifer, another comparison was considered: Figure 6.16 and Figure 6.17 show a comparison between the simulation runs (a) 2-A (aquifer only), (b) 2-B (aquifer plus caprock and bedrock) and (c) 3-B (aquifer plus caprock and bedrock with higher injection and production rates) in terms of the color maps for temperature distribution over the plane joining the two wells after 1 year and 10 years respectively.

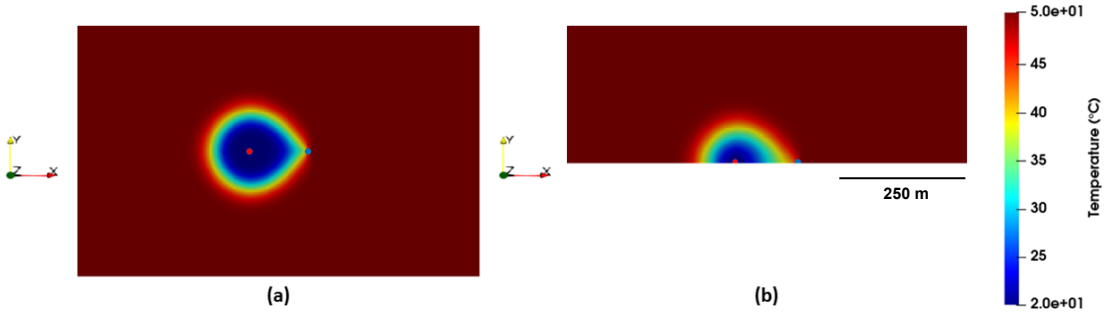


Figure 6.14: Zoomed top view for thermal front position relative to well locations for (a) run 2-A and (b) run 2-B after 10 years.

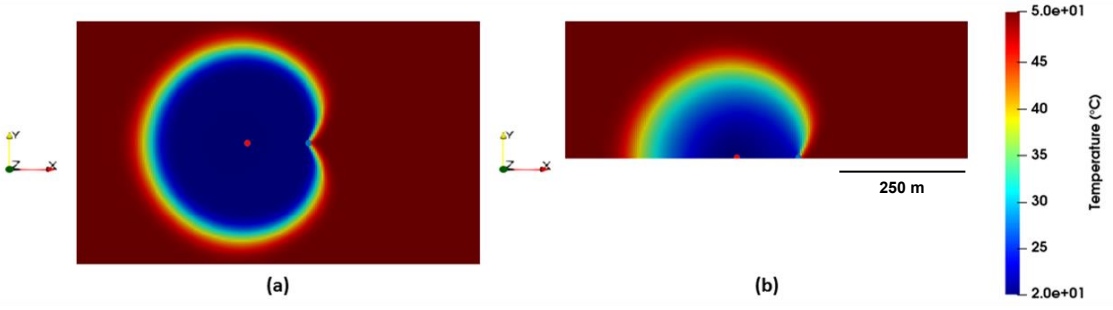


Figure 6.15: Zoomed top view for thermal front position relative to well locations for (a) run 3-A and (b) run 3-B after 10 years.

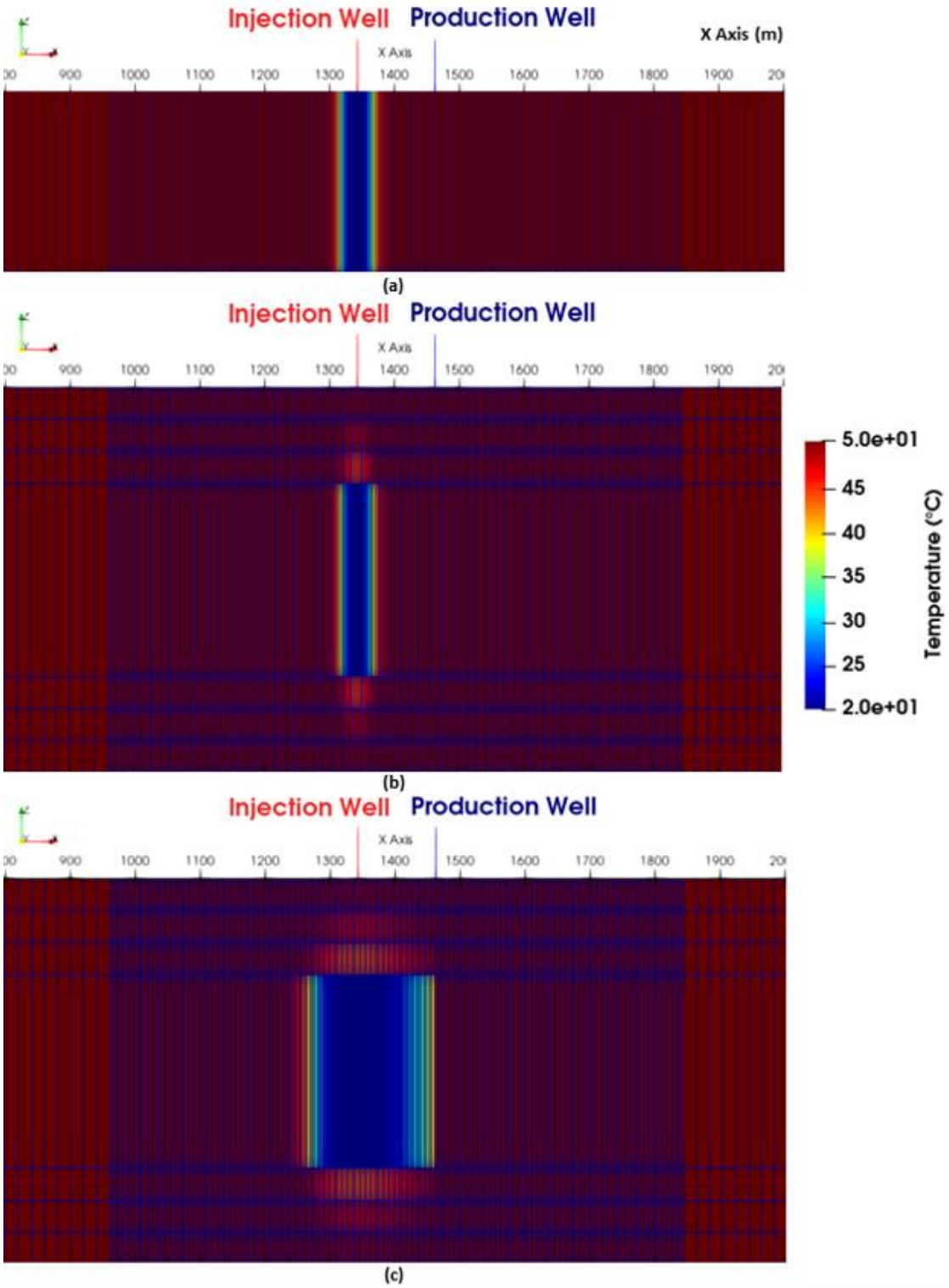


Figure 6.16: Color maps for temperature distribution over the plane joining the two wells after 1 year for (a) run 2-A (aquifer only), (b) run 2-B (aquifer plus caprock and bedrock) and (c) run 3-B (aquifer plus caprock and bedrock with higher injection and production rates)

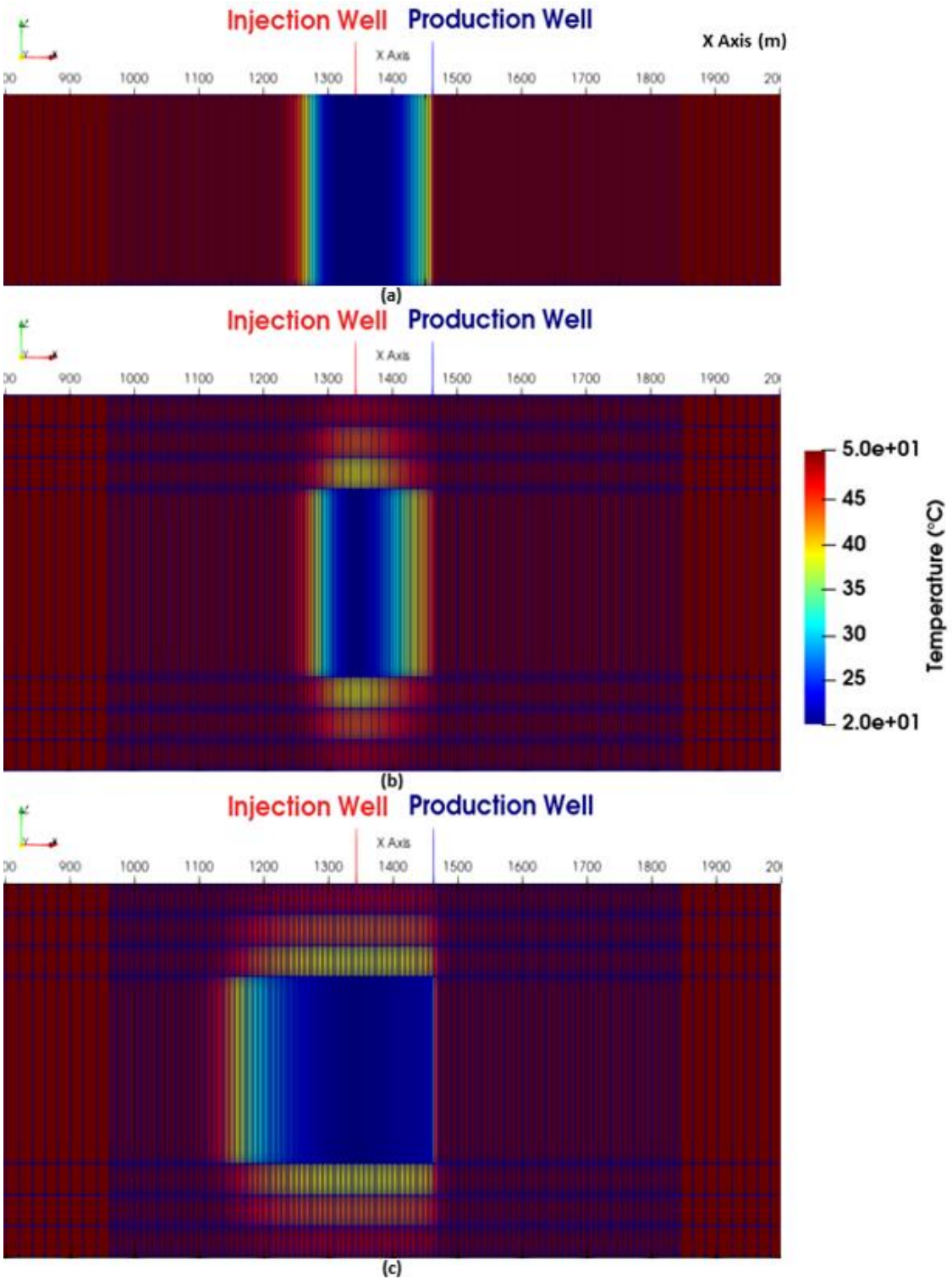


Figure 6.17: Color maps for temperature distribution over the plane joining the two wells after 10 years for (a) run 2-A (aquifer only), (b) run 2-B (aquifer plus caprock and bedrock) and (c) run 3-B (aquifer plus caprock and bedrock with higher injection and production rates)

It is quite clear that the interaction between the cooled injection water and the caprock\bedrock results in conductive heat transfer which in turn leads to the reduction of temperature of the boundary cell just above\below the injection cell and some of the surrounding cells in the same numerical layer. Afterwards, conductive heat transfer continues to alter the temperature of the next upper\lower numerical boundary layer. By comparing Figure 6.16b to Figure 6.17b and Figure 6.16c to Figure 6.17c, it can be noted that the zones influenced by the temperature reduction in the caprock/bedrock expand in time as the cooled-water thermal front becomes more distant from the injection well. This signifies that more and more of the heat content of those boundary rocks is being drained in time by the thermal front traveling in the aquifer via thermal conduction.

It can be observed from Figure 6.16a and Figure 6.16b that both simulation runs 2-A and 2-B result in almost similar temperature distributions over the considered plane after a 1-year simulation duration. It can also be seen that the cooled-water thermal front for both runs is almost symmetric around the injection well despite the depletion effect on the side where the water producer is situated. On the contrary, Figure 6.17a and Figure 6.17b show different temperature distributions produced by the two simulation runs considering a 10-year simulation time span where it is obvious that the thermal front of the run 2-A is steeper than that of the run 2-B which appears to be smoother with a more gradual change of the amplitudes. Moreover, it is noted from Figure 6.17a and Figure 6.14a that the thermal front is asymmetric with respect to the injection well while the asymmetricity of the front is less pronounced in both Figure 6.17b and Figure 6.14b although it still does exist. The reduced asymmetry of the front was again observed for the run 3-B (Figure 6.15b) compared to the run 3-A (Figure 6.15a). Furthermore, Figure 6.15b shows a smoother front with a larger transition zone between the injected temperature and the aquifer temperature compared to Figure 6.15a in which the front is sharper. The previous observations demonstrate that both runs 2-B and 3-B, which account for caprock and bedrock conduction, have exhibited the same behavior of a smoother front of reduced asymmetry compared to the corresponding runs 2-A and 3-A in which caprock and bedrock conduction is ignored.

The main reason for the deviation between the behaviors of the two thermal fronts of the runs 2-A and 2-B over time is the thermal conduction effect caused by the intercommunication between the cooled injected water and both of the caprock and bedrock modeled in scenario B. The thermal conduction in the run 2-B leads to the loss of a part of the cold energy contained in the cooled injected water to cool down the bounding rocks and drain their heat content. As a result, the progression of the cooled-water thermal front through the aquifer will be retarded. On the opposite side, no caprock or bedrock are modeled in scenario A and thus no energy is lost by the cooled injected water to the confining rocks which allows the thermal front of the run 2-A to progress faster through the aquifer. Furthermore, the added caprock and bedrock thermal conduction in the run 2-B acts as a counteracting effect to the higher advective heat transfer on the side where water is being extracted. This is because a part of the cold thermal energy of the injected water is lost to the caprock and bedrock by conduction instead of being transmitted by convection due to the motion of the injected

water through the porous geothermal aquifer. This in turn results in the reduced asymmetricity of the thermal front in case of the run 2-B since the thermal front cannot propagate towards the production well as fast as in the case of the run 2-A owing to the reduced advective heat transfer by caprock and bedrock conduction. To summarize, the incorporation of the caprock and bedrock into the geometric model of the geothermal aquifer leads to the slowing down of the cooled-water thermal front due to the added caprock and bedrock thermal conduction resulting in a smoother front with reduced asymmetry.

It should be pointed out that the thermal front retardation due to caprock and bedrock conduction will be more significant for long simulation durations. This is because for short simulation durations, the amount of cold energy lost from the injected cooled water to the bounding rocks will not be large enough to cause a delay of the thermal front of the run 2-B with respect to the corresponding front of the run 2-A. This is clearly demonstrated by the approximate similarity of the temperature distributions resulting from both runs which are presented in Figure 6.16a and Figure 6.16b considering a 1-year simulation duration. Consequently, the front retardation by boundary thermal conduction is a cumulative effect just like the asymmetricity of the front due to increased advective heat transfer.

Conversely, for the simulation run 3-B, the conduction effect is already significant after 1 year as shown in Figure 6.16c. It can also be seen in Figure 6.17c how thermal conduction has affected a large portion of the caprock and bedrock after 10 years. This is obviously attributed to the higher injection rate in the run 3-B that results in a faster propagation of the cooled-water thermal front. This in turn translates to a more laterally extended aquifer cooled zone exhibiting conductive heat transfer with the caprock and bedrock.

To evaluate the sensitivity of the pressure behavior to the modeled caprock and bedrock in scenario B, a line plot comparison was made between the two runs 2-A and 2-B in terms of pressure distribution over the plane joining the two wells (Y=0 m in the half domain and Y=1402 m in the full domain) after 1 and 10 years of the operation of the geothermal doublet. The results are presented in Figure 6.18

Both simulation runs 2-A and 2-B exhibit a very similar pressure distribution over the considered plane for both the 1 and 10-year simulation cases (Figure 6.18); a slight separation could be observed for the 10-year case at the positions of the wells. Analogous results were found out by comparing the pressure distributions of the 3-A and 3-B runs. This similarity of the simulated pressure distributions for both the A and B scenarios clearly demonstrates that pressure is almost unaffected by the inclusion of the bedrock and caprock into the geometry of the numerical model.

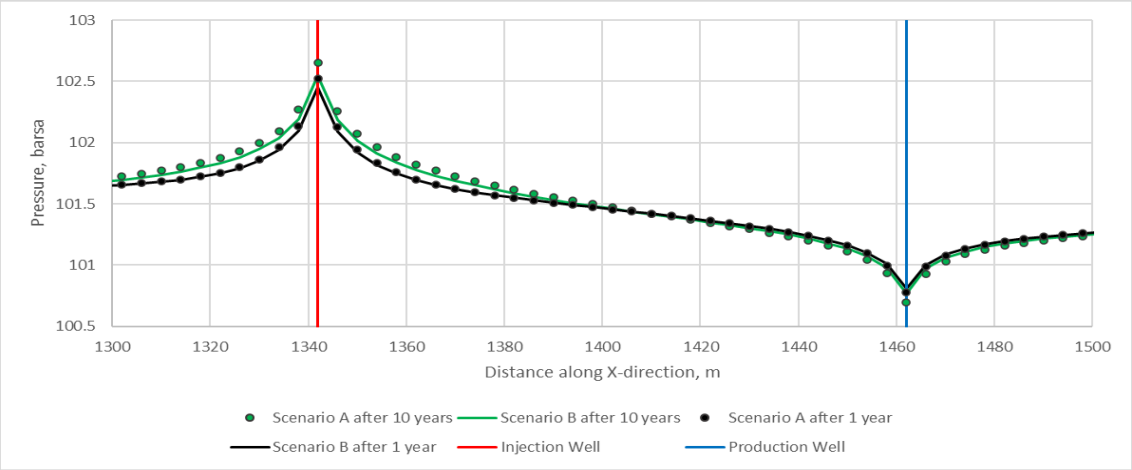


Figure 6.18: Line plot comparison between the runs 2-A and 2-B in terms of pressure distribution over the plane joining the two wells after 1 and 10 years of simulation time

For a more in-depth investigation of the influence of the thermal conduction phenomenon across the caprock and the bedrock on the temperature distribution in the geothermal aquifer and the consequent effects on the temperature-dependent water viscosity, a line plot comparison was made between the two runs 2-A and 2-B in terms of temperature and viscosity distributions over the plane joining the two wells as illustrated in Figure 6.19 and Figure 6.20 respectively considering simulation durations of 1, 5 and 10 years.

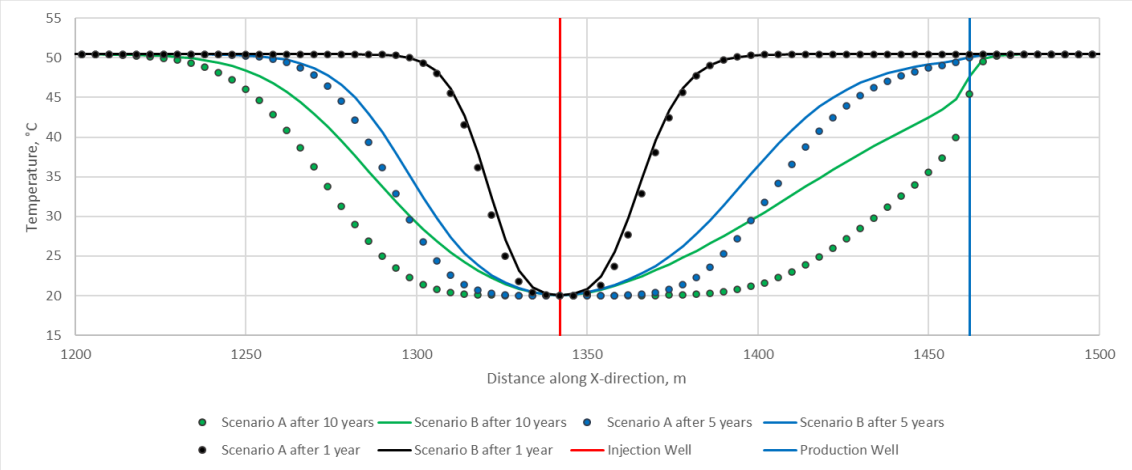


Figure 6.19: Line plot comparison between the runs 2-A and 2-B in terms of temperature distribution over the plane joining the two wells after 1, 5 and 10 years of simulation time

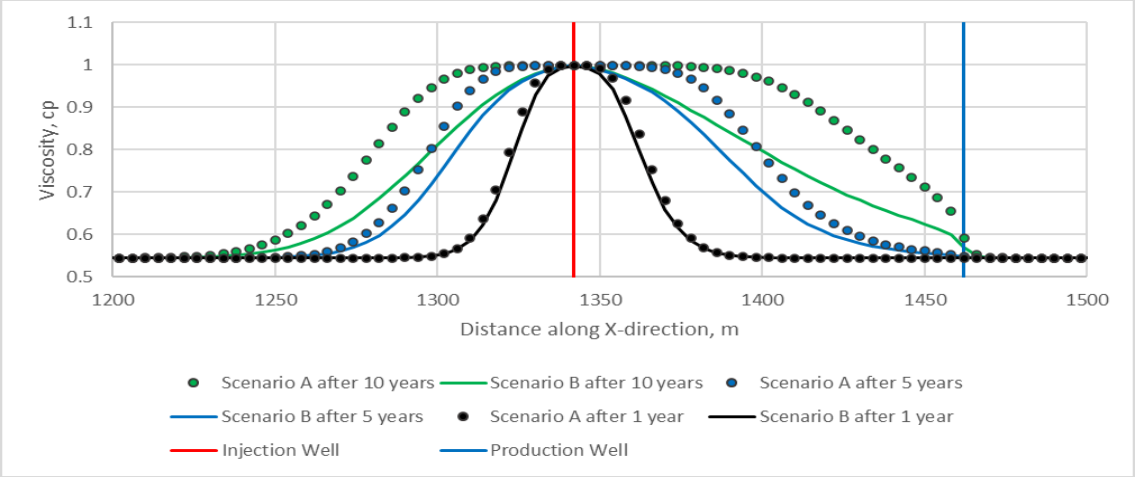


Figure 6.20: Line plot comparison between the runs 2-A and 2-B in terms of viscosity distribution over the plane joining the two wells after 1, 5 and 10 years of simulation time

It can be noted from Figure 6.19 how the thermal front of the run 2-A will be ahead of the corresponding front of the run 2-B considering the 5 and 10-year simulations and how the 2-B front retardation becomes less pronounced as the simulation duration shortens where a negligible separation between the two fronts can be seen for the 1-year simulation case. Furthermore, it can also be noted how the asymmetricity of the front increases in time with a reduced asymmetricity for scenario B with respect to scenario A because the increased advective heat transfer caused by depletion is being opposed by boundary thermal conduction.

It is observed from Figure 6.20 how the viscosity distribution curves at the different simulation times exhibit a mirrored reflection of the corresponding temperature distribution curves such that they show an increase where temperature decreases and vice versa. For longer simulation times, the lateral extent of the high viscosity zone around the injection well becomes larger for the run 2-A with respect to that of the run 2-B. This is due to the more laterally extended cooled zone of the run 2-A as no cold energy is lost to the bounding layers. Moreover, the asymmetricity of the high viscosity zone is more evident at longer simulation times and is more pronounced for scenario A compared to scenario B in which caprock and bedrock thermal conduction lessens the effect of increased convective thermal transfer towards the production well.

Figure 6.19 was used to quantify the maximum percent error for the predicted temperature distribution of the run 2-A with respect to that of the run 2-B at the simulation durations of 1, 5 and 10 years. A similar analysis was performed for the high-rate runs 3-A and 3-B. The aim was to evaluate the impact of caprock and bedrock thermal conduction by numerically quantifying the maximum error that could result from ignoring the incorporation of such phenomenon into the numerical model. The error was calculated by applying equation (6.1) for each cell along the considered plane in Figure 6.19 and then picking the maximum value.

$$\text{Error}(\%) = \frac{|\text{Temperature}_{\text{scenario B}} - \text{Temperature}_{\text{scenario A}}|}{\text{Temperature}_{\text{scenario B}}} \text{ x } 100 \qquad \textbf{(6.1)}$$

The results are presented in Table 6.2. It can be seen that the maximum percent error of temperature distribution prediction resulting from ignoring the thermal conduction across the boundaries in a geothermal system may reach up to 29.39% for a 10-year simulation period. This error may even increase up to 35.44% for the same simulation duration if significantly high rates are employed in the operation of the geothermal doublet as in the case of run 3. This clearly shows that ignoring caprock and bedrock conduction will be more critical in terms of the accuracy of the simulation outcome in case of long simulation periods and considerably high injection and production rates.

Table 6.2: Maximum error for the predicted temperature distribution in case of ignored caprock and bedrock thermal conduction

	Scenarios	1-year simulation	5-year simulation	10-year simulation
Maximum percent error [%]	2-A vs. 2-B	7.36	19.95	29.39
	3-A vs. 3-B	12.88	25.96	35.44

6.5 Dumux vs. ECLIPSE

All of the six simulation runs considered within the scope of this study, listed in Table 6.1, have been performed by both Dumux and ECLIPSE simulators. It was desired to check the comparability between the simulation results produced by both softwares taking also into account how they compare to each other from the computational cost aspect. For scenario B half-domain was simulated in Dumux to reduce the computational cost. Conversely, full domain was simulated with Eclipse, to validate the symmetry boundary condition imposed in Dumux.

For all simulation runs, Dumux-ECLIPSE comparisons were made in terms of line plot comparisons for:

- Spatial variation of pressure, temperature and viscosity along the X-direction over the plane joining the two wells considering simulation durations of 3 and 6 months for run 1 and simulation durations of 1, 5 and 10 years for runs 2 and 3.
- Temporal variation of injection and production pressures.

However, for the runs of scenario B, additional comparisons were made in terms of:

- Temporal variation of the temperature of the boundary cell just above the injection cell.
- Spatial variation of pressure, temperature and viscosity along the Y-direction over the plane having the X-coordinate of the injection well (X=1342 m).
- Spatial variation of pressure along the Y-direction over the plane having the X-coordinate of the production well (X=1462 m).

Starting with the base case (run 2-A), a perfect match was obtained between the Dumux and ECLIPSE results. As proof, the temperature, pressure and viscosity distributions over the plane joining the two wells which have been predicted by both simulators are presented in Figure 6.21, Figure 6.22 and Figure 6.23 respectively.

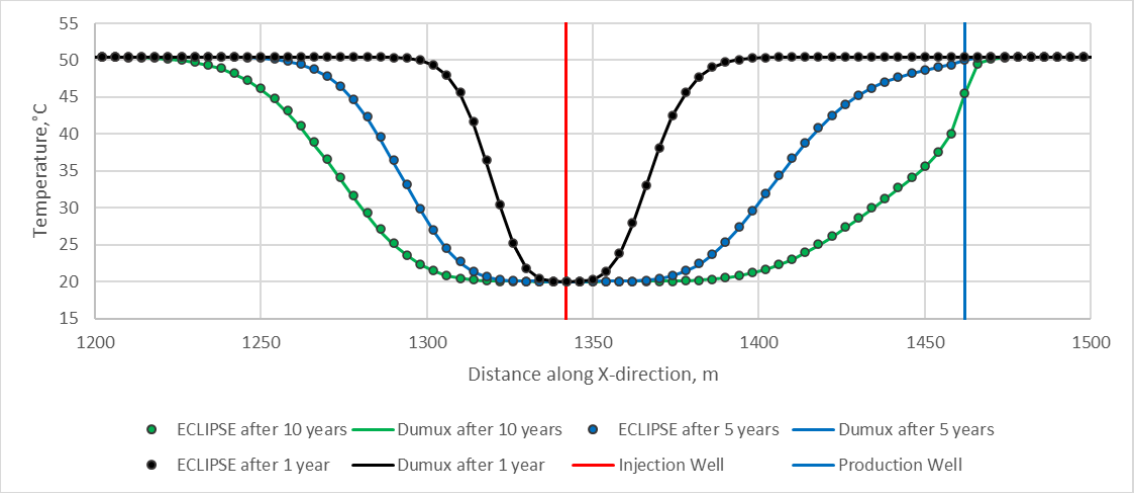


Figure 6.21: Comparison between Dumux and ECLIPSE in terms of temperature distribution over the plane joining the two wells (Y=1402 m) for the run 2-A

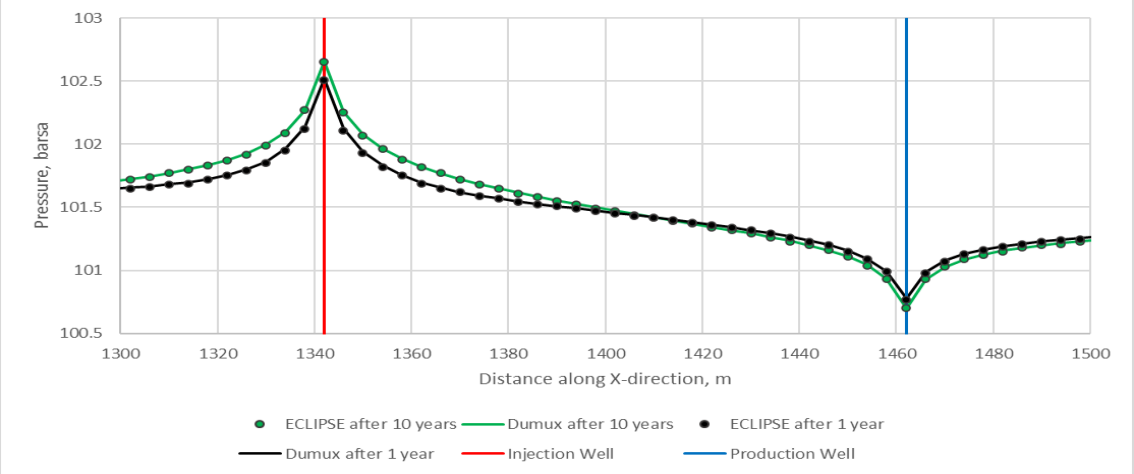


Figure 6.22: Comparison between Dumux and ECLIPSE in terms of pressure distribution over the plane joining the two wells (Y=1402 m) for the run 2-A

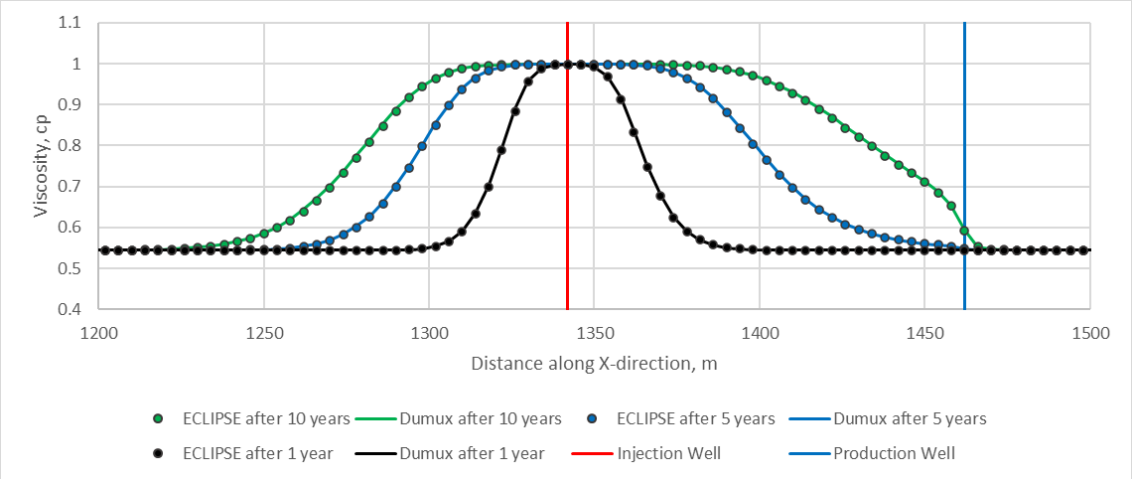


Figure 6.23: Comparison between Dumux and ECLIPSE in terms of viscosity distribution over the plane joining the two wells (Y=1402 m) for the run 2-A

The perfect matching of the viscosity distributions predicted by Dumux and ECLIPSE as illustrated in Figure 6.23 demonstrates that the viscosity table implemented in ECLIPSE (Table 3.3) is sufficiently accurate as it allowed ECLIPSE to successfully reproduce the equation of state (IAWPS relations) implemented in Dumux.

A perfect agreement of results was demonstrated also by comparing the simulation outputs of Dumux and ECLIPSE for the runs 1-A and 1-B in which the value of the maximum allowed time step size was reduced to 1 day. This clearly showed that smaller time stepping did not affect the quality of result match between Dumux and ECLIPSE.

Increasing the rate (case 3-A), the results produced by Dumux and ECLIPSE are still showing a very good match. Only very slight deviations could be observed by comparing the temperature distribution curves of the two simulators over the plane joining the two wells for the 5-year and 10-year simulations as seen in Figure 6.24. The maximum percent difference of the Dumux curve with respect to the ECLIPSE curve had a value of 3.79% for the 10-year simulation and 2.9% for the 5-year simulation. Such minimal deviations which did not occur in the base case (run 2-A) comparison of the two simulators may thus be attributed to the high injection and production rates employed in the run 3-A. However, they certainly do not influence the comparability of the results.

Conversely, when accounting for thermal exchange between the aquifer and the caprock/bedrock, some differences arise. Different trends were generated by the two simulators for the temporal trend of the temperature of the boundary cell just above the injection cell which accounts for pure thermal conduction effect as shown in Figure 6.25a.

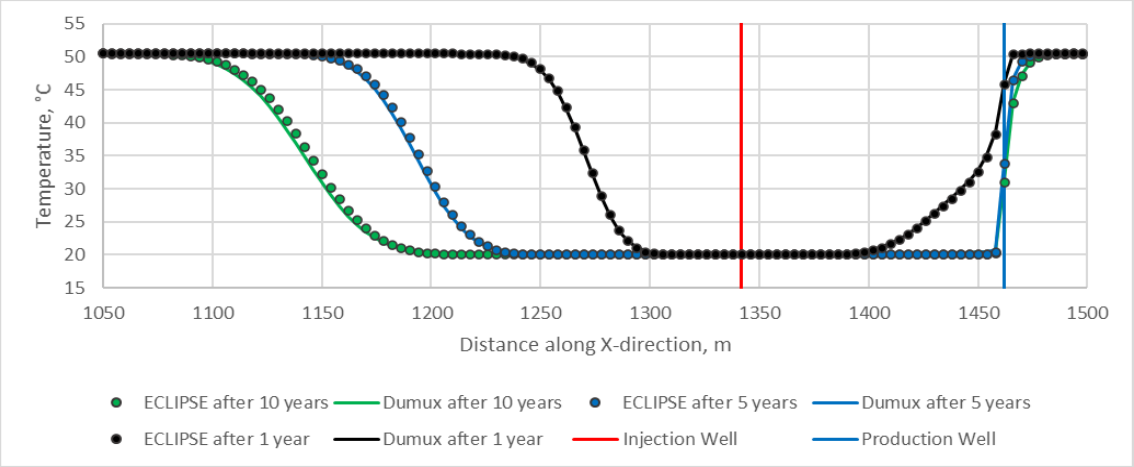
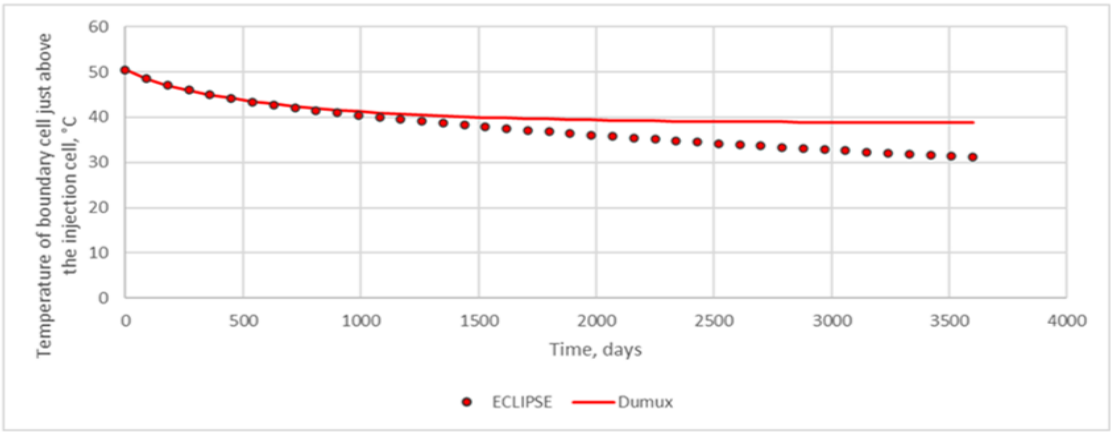
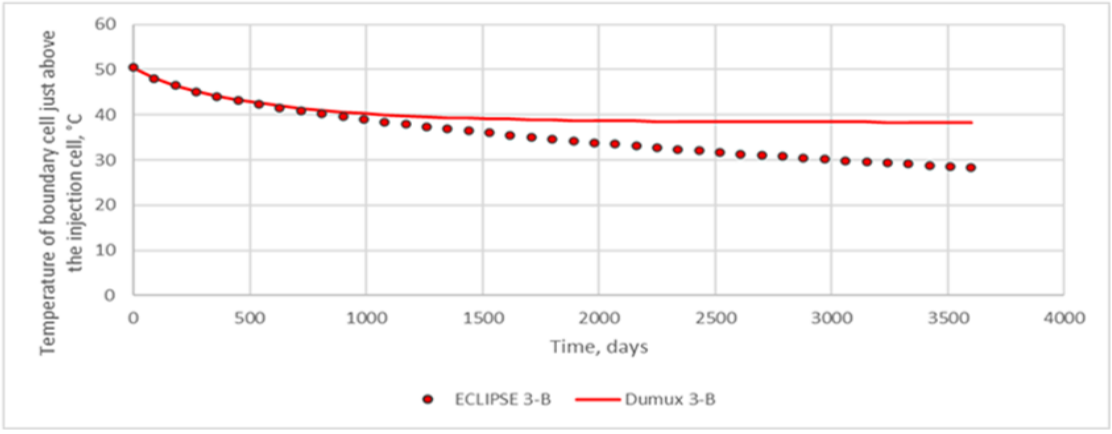


Figure 6.24: Comparison between Dumux and ECLIPSE in terms of temperature distribution over the plane joining the two wells (Y=1402 m) for the run 3-A



(a)



(b)

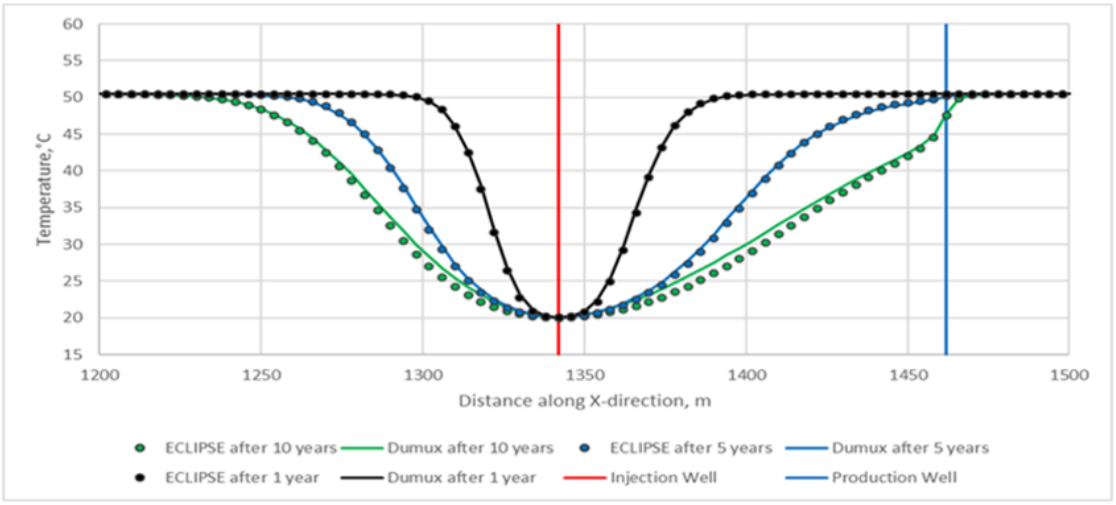
Figure 6.25: Comparison between Dumux and ECLIPSE in terms of temporal variation of the temperature of the boundary cell just above the injection cell for (a) run 2-B and (b) run 3-B

ECLIPSE follows the same trend as Dumux in the beginning of the simulation up to a point in time where deviation starts to take place and the two curves begin to follow different trends. The percent deviation was calculated for the different time points taking the ECLIPSE value as the reference one. The calculations in case 2-B showed that the percent deviation up to 990 days (~2.71 years) is less than 2% while it reaches 24.24% by the end of the whole simulation duration which is 3600 days (~10 years). Analogous behavior is observed in the case 3-B, where a slightly increased deviation between the two trends was observed (Figure 6.25b). ECLIPSE showed a further decrease of the boundary cell temperature down to 28.33 °C for the run 3-B compared to 31.21 °C for the run 2-B while Dumux showed a temperature decrease down to 38.34 °C for the run 3-B compared to 38.78 °C for the run 2-B.

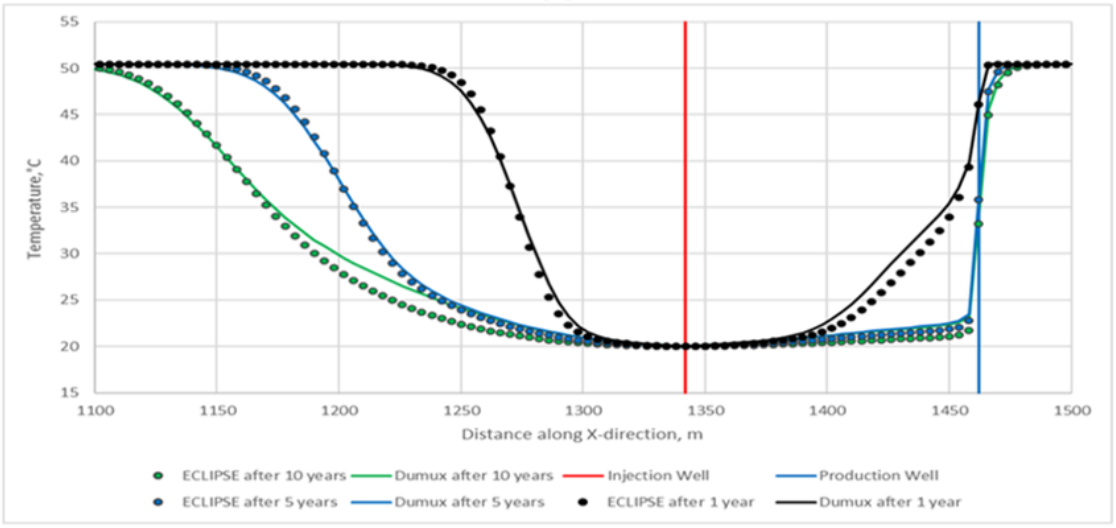
Differences in thermal conduction effects simulated by Dumux and ECLIPSE are particularly significant at long simulation durations. This can be a valid reason for a small observed deviation for the same run 2-B between the temperature distribution curves modeled by both simulators over the plane joining the two wells for the 10-year simulation case as shown in Figure 6.26a. However, the maximum percent difference for the Dumux curve with respect to the ECLIPSE curve was not critical as it assumed a value of 5.78% at X=1386 m. Such trend is highlited in case 3-B (Figure 6.26b). The maximum percent difference of the Dumux curve with respect to the ECLIPSE curve was 7.44% at X=1422 m for the 1-year simulation and 8.26% at X=1234 m for the 10-year simulation.

Comparisons on the Y-axis cross-section passing through the injector showed a very similar behavior.

To summarize, the simulation results of Dumux and ECLIPSE are generally very comparable. However, thermal conduction phenomenon appears to propagate differently in the two simulators such that the modeled conduction-controlled temperature trends by Dumux and ECLIPSE will have a significant deviation at long simulation periods. Furthermore, high flow rates may seem to cause slight discrepancies between the temperature spatial variation curves predicted by the two simulators. The observed discrepancy is probably due to the different way the two simulators model the temperature boundary condition at the caprock and bedrock external surfaces. In fact, ECLIPSE assumes adiabatic conditions, i.e. no thermal exchange exists between the simulated domain and the extenal rock. Conversely, in Dumux, undisturbed constant temperature was imposed on caprock and bedrock external surfaces. In any case, such slight deviations do not affect at all the good comparability of the Dumux and ECLIPSE results.



(a)



(b)

Figure 6.26: Comparison between Dumux and ECLIPSE in terms of temperature distribution over the plane joining the two wells (Y=0 m) for runs (a) 2-B and (b) 3-B

6.5.1 Computational Cost

A computational cost comparison is held between Dumux and ECLIPSE by considering the actual time taken by Dumux to perform each of the six simulation runs considered within the scope of this study compared to the corresponding time consumed by ECLIPSE. The results are shown in Table 6.3. It can be seen from the table that using the Finite Volume-based Dumux for numerical simulation involves a greatly higher computational cost compared to the Finite Difference-based ECLIPSE. By considering the ratio of the time consumed by Dumux to the time consumed by ECLIPSE to perform each of the six simulation runs as elaborated in Table 6.3 and then considering an average time ratio by computing the arithmetic average of all the six time ratios, a ratio of 595.64 is obtained. This means that within the scope of this study, Dumux required on average almost 600 times the computational time required by ECLIPSE to run the same simulation.

Table 6.3: Comparison of the computational time taken by Dumux vs. ECLIPSE to perform the simulation runs

	Simulation Run	Scenario A		Time ratio	Scenario B		Time ratio
		Dumux	ECLIPSE		Dumux	ECLIPSE	
Actual time taken [h]	Run 1	13.611	0.0204	667.21	136.111	0.1313	1036.64
	Run 2	13.611	0.0304	447.73	47.222	0.094	502.36
	Run 3	14.167	0.044	321.98	86.11	0.144	597.99

7 CONCLUSION

The spatial progression of the thermal front over the years in a low-temperature geothermal doublet was simulated using the Finite-Volume based code of the research simulator Dumux. The behavior of pressure and water viscosity was examined too. Sensitivities to caprock and bedrock thermal conduction as well as different scenarios for injection and production rates were analyzed.

Thermal front shape was analyzed to verify if well test could be implied to monitor the thermal front distance from the injector, in order to predict the cooled front breakthrough at the producer. In correspondence to the cooled zone around the water injector, a high viscosity zone will form and grow in time based on thermal front progression. Thus, a radial composite model characterized by two zones of different viscosity can adopted for well test interpretation. However, possible asymmetry of the cooled front could be an obstacle to such approach. A certain front asymmetry could be observed during both the pre- and post-thermal breakthrough phases but with different front shapes being minimum at the beginning of the simulation and becoming more pronounced over time. Further studies are needed to evaluate the approximation degree of a radial composite assumption in such scenarios.

The thermal conduction phenomenon across caprock and bedrock layers has proved to be particularly critical when long simulation durations or high injection and extraction rates are considered. In fact, it can influence the modeled temperature distribution and thermal front development and thus eventually affecting the predicted time point for thermal breakthrough occurrence. The effect of taking into account of caprock and bedrock conduction is a slowing down of the cooled-water thermal front. Moreover, it generates a smoother front with a larger transition zone between the injected temperature value and that of the geothermal aquifer.

Higher injection and production rates employed in the operation of a geothermal doublet will provide a faster thermal breakthrough owing to the higher convective heat transfer attributable to the higher Darcy velocity. They will also bring significant caprock and bedrock conduction effects even in the case of short simulation time spans. Furthermore, the higher operational rates will also imply a wider operational pressure range in the geothermal system.

The simulation outputs of Dumux were consistent for the different imposed maximum time step sizes. The validation of the simulation outcome of Dumux against that of ECLIPSE showed that the results of the two simulators are generally in good agreement. However, it was found out that the conduction-controlled temperature trends modeled by both simulators were different. The deviation between the two trends is significant when long simulation time spans are considered. The differences between ECLIPSE and Dumux results increase by increasing the rates. The observed discrepancies are probably due to the different temperature boundary conditions imposed at the caprock and bedrock external surfaces by the two simulators. In any case, such slight deviations do not affect at all the good comparability of the Dumux and ECLIPSE results. Nevertheless, Dumux is much more computationally costly.

APPENDIX

A1. Overview of Basic C++ Concepts and Nomenclature

Before discussing the features of the Dumux code, some basic concepts and nomenclature of the C++ programming language which was used to write the Dumux code should be clarified first. Those basic C++ concepts are listed below.

- C++ data types: they determine the type of data that can be stored in a certain variable where according to Agarwal, (2021), they are subdivided into:
    1. Primitive data types: they include for instance integers (int), boolean values (bool), floating-point numbers (float) and double-precision floating-point numbers (double) which allow for more decimal places than floating-point numbers.
    2. Derived data types: They originate from primitive data types and include for instance functions and arrays.
    3. User-defined data types: They include for instance C++ classes and structs.
- Variable declaration in C++: it means the introduction of the variable before performing further operations on it where this is done by stating the name and the type of the variable (Prabhu, 2019). An illustration of how variable declaration is performed is shown in the code snippet (C1) where type is the data type of the variable which could be (int) or (float) or others and any_var is the variable's name.

```
// Declaring a variable
type any_var;
```

(C1)

- C++ function: it is formed from a group of statements that perform some computational operations on an input and generate an output. The input is given through the values of the function arguments and the output depends on the return type of the function. The code snippet (C2) shows a simple C++ function that compares two input numbers and outputs the larger of the two. The arguments of the function are the two integers a and z and its return type is also of type int.

```
int maximum(int a, int z)
{
  if (a > z)
    return a;
  else
    return z;
}
```

(C2)

- C++ class: it is a user-defined data type that has its own functions and variables inside the class body such that the variables are termed data members and the functions performing operations on those variables are called member functions (Kariya, 2021). It is further explained by Kariya,

(2021) that an object of the class has to be created to be able to access the members of the class and manipulate them. However, accessing the class members is controlled by the type of access specifier which this member belongs to where three types of access specifiers exist according to W3Schools, (n.d.) as follows:

1. Public: it allows class members to be accessed from outside the class.
2. Private: it does not allow class members to be accessed externally.
3. Protected: it does not allow access to the class members from outside the class but they can be accessed from derived classes.

The example in the code snippet (C3) illustrates the class definition while the example in the code snippet (C4) demonstrates how an objected of the class is created and used to access a class member.

```
class class_name {      // The class
  public:               // Access specifier
    int sum;            //  Integer variable
};
```

(C3)

```
int main() {

   // Declaring an object of class class_name
   class_name obj;

   // Accessing data member
   obj.sum = 10;
 }
```

(C4)

- C++ struct: it is similar to a C++ class with main difference that the programming details are not as well protected as in the case of a C++ class because the default setting for the class members is being private while the default setting for the struct members is being public (GeeksforGeeks, 2021c).
- C++ namespace: it is a named scope inside which a variable of a certain name and type can be declared avoiding mixing between this variable and another variable of the same name but of a different data type declared inside another namespace where such C++ feature is useful in case of large codes. A C++ namespace thus offers a narrower scope for the names of variables, functions, classes and structs which allows a more logical organization for those entities (Tiwari, 2019). An example illustrating how a namespace is defined is presented in the code snippet (C5)

```
namespace new_name
{
  int a, b; // variable declarations such that
            // a and b are declared inside
           // new_name's scope
}
```

(C5)

- Inheritance in C++: it means that a class is able to access the members of another class because it has inherited the properties of that other class. This means that code duplication is avoided as the inherited functions and variables do not have to be written again inside the body of the inheriting class (Agarwal, 2021b). It is pointed out by Agarwal, (2021b) that the inheriting class is termed derived class while the class whose characteristics are passed to the derived class is termed base class.
- Templates in C++: It is a very useful feature that is based on the idea of writing a generic code that is compatible with the different data types and thus not having to write the same piece of code for each data type where the feature is implemented in the form of function templates and class templates (GeeksforGeeks, 2021a). The declaration of both the function template or the class template should be preceded by the keyword template then angle brackets which contain the template argument preceded by the keyword **class** or **typename** as shown in the code snippet (C6) for the template function declaration and the code snippet (C7) for the template class declaration. According to the data type that will be passed to the template argument T in the program, the compiler will create another variant of anyFunction() or class_name for that specific data type (Programiz, n.d.)

```
template <class T>
T anyFunction(T arg)
{
        // function body
}
```

(C6)

```
template <class T>
class class_name
{
public:
        T var;
        T anyOperation (T arg)
};
```

(C7)

- Template specialization in C++: it means using a special version of the class template or the function template for a specific data type such that the code is different from the one used for the other data types (GeeksforGeeks, 2021b). The example in the code snippet (C8) shows how a class template for

example may be specialized for a certain data type where the class template called class_name has been specialized for the data type int.

```
template <class T>
class class_name
{

public:
  // Generic code

};

template <>
class class_name <int>
{
public:
  // Special code

};
```

(C8)

- Partial Template Specialization in C++: It differs from the full template specialization as not the same data type is passed to all the template arguments and thus the template is not fully specialized for a specific data type (Sergey & W.F., 2017). This means that some of the template arguments may be fixed. Such a feature allows the template to handle totally diverse datasets, however; it is applicable only to class and struct templates and not function templates (Sergey & W.F., 2017). An example of a struct template provided by Sergey & W.F., (2017). is shown in the code snippet (C9). In this snippet, it is noticeable that even if the same data type is passed to the template arguments T and Z, the struct template will not be fully specialized for that data type as there is a third fixed template argument of the type int.

```
template<typename T, typename Z>
struct A<int, T, Z> {
   // code
};
```

(C9)

It's important to point out that the properties of the system/model are assigned using the Dumux property system which is built on the concepts of inheritance and template specialization in the C++ programming language (Flemisch, 2018; The DuMux developers, 2021). In the framework of Dumux, a property is the body of a C++ class where those properties are attached to the nodes of a hierarchical structure termed type tags such that a lower node (type tag) is inheriting the properties of the upper one (The DuMux developers, 2021).

A2. Dumux Installation, Compiling and Running

To run Dumux on a Linux platform, the following pre-requisites had to be installed:

- CMake version 3.16.3
- gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
- pkg-config version: 0.29.1
- OpenMPI version:4.0.3
- ParaView version 5.7.0
- Python 3.8.5
- git version 2.25.

Dumux v3.3 was installed in January 2021 from the installation page on the Dumux website (https://dumux.org/installation) using the "Installation via script" option. The python script was downloaded and run in the terminal by typing the command shown in the code snippet (C10).

```
python3 installdumux.py
```
(C10)

The steps in the "Getting Started" section on the Dumux website (https://dumux.org/gettingstarted/) were followed to create a new module in the Dumux installation directory inside which two folders were created: one for scenario A and the other one for scenario B.

After having written the Dumux code for each of the two simulation cases, the CMakeLists.txt file in the directory of each simulation case had to be altered in order to be able to compile the code of that problem. The CMakeLists.txt file is adjusted by adding/modifying the dune_add_test(..) command (Scholz et al., 2018) as illustrated in the code snippet (C11) for scenario B.

```
dune_add_test(NAME half_final
        SOURCES main.cc
        COMPILE_DEFINITIONS TYPETAG=OnePNICCMpfa)
```
(C11)

where NAME refers to the name of the executable for that problem that will later be used in the terminal commands to build and run the problem/application and which in this case is half_final. SOURCES refers to the name of the main file (main.cc) which includes the main function of the code. Finally, the name of the type tag of the problem -which contains all the system properties- is entered which in this case is OnePNICCMpfa.

Furthermore, the CMakeLists.txt file will include a shortcut to the parameters/input file and thus the input file name (params.input) has to be entered correctly as shown in the code snippet (C12).

```
dune_symlink_to_source_files(FILES "params.input")
```
(C12)

In addition, the CMakeLists.txt file in the new module's directory also has to be altered by adding the subdirectory for each of the two added scenarios (the names of their folders) as shown in the code snippet (C13).

```
add_subdirectory(convection_final)
```
(C13)

```
add_subdirectory(half_final)
```

After editing the CMakeLists.txt files as previously illustrated, we have to go through the terminal into the directory of our new module which in our case was called "dumux_alpha". by typing the command shown in the code snippet (C14).

```
cd dumux/dumux_alpha
```
(C14)

And then we have to perform a re-configuration of the module by typing in the terminal window the command shown in the code snippet (C15).

```
cmake build-cmake
```
(C15)

Next, to run the problem for scenario B for example, we have to go through the terminal into the folder/directory of that problem located inside the "build-cmake" folder of our module as elaborated in the code snippet (C16). It's worth noting that the "build-cmake" folder will be automatically created inside our new module once we create it.

```
cd build-cmake/half_final
```
(C16)

After that, we should build the code using the command in the code snippet (C17) where the name of the executable is used.

```
make half_final
```
(C17)

Final step is to run the application by typing the command in the code snippet (C18) in the terminal window. The name of the problem executable is used once more.

```
./half_final
```
(C18)

After the simulation run has finished and in order to visualize the simulation outcome on ParaView, the command in the code snippet (C19) has to be typed in the terminal.

```
paraview *pvd
```
(C19)

A3. Dumux Code of Base Case

Problem File (problem.hh)

```
#ifndef DUMUX_1PNI_PROBLEM_HH
#define DUMUX_1PNI_PROBLEM_HH
// Yet Another Structured Parallel Grid
#include <dune/grid/yaspgrid.hh>
// Discretization using MPFA
#include <dumux/discretization/ccmpfa.hh>
// Porous Medium Flow Problem
#include <dumux/porousmediumflow/1p/model.hh>
#include <dumux/porousmediumflow/problem.hh>
// Single component H2O in Liquid Phase
#include <dumux/material/components/h2o.hh>
#include <dumux/material/fluidsystems/1pliquid.hh>
// Spatial Params
#include "spatialparams.hh"

namespace Dumux {

template <class TypeTag>
// Forward Declaration
class OnePNIProblem;

namespace Properties {
// Create new type tags
namespace TTag {
struct OnePNITypeTag { using InheritsFrom = std::tuple<OnePNI>; };
struct OnePNICCMpfa { using InheritsFrom =
std::tuple<OnePNITypeTag, CCMpfaModel>; };
} // end namespace TTag

// Set the grid type
template<class TypeTag>
struct Grid<TypeTag, TTag::OnePNITypeTag> { using type =
Dune::YaspGrid<3, Dune::TensorProductCoordinates<double, 3> >; };

// Set the problem property
template<class TypeTag>
struct Problem<TypeTag, TTag::OnePNITypeTag> { using type =
OnePNIProblem<TypeTag>; };

// Set the fluid system
template<class TypeTag>
struct FluidSystem<TypeTag, TTag::OnePNITypeTag>
{
```

```
 41     using type = FluidSystems::OnePLiquid<GetPropType<TypeTag,
Properties::Scalar>,
 42
Components::H2O<GetPropType<TypeTag, Properties::Scalar>> >;
 43 };
 44
 45 // Set the spatial parameters
 46 template<class TypeTag>
 47 struct SpatialParams<TypeTag, TTag::OnePNITypeTag>
 48 {
 49     using GridGeometry = GetPropType<TypeTag,
Properties::GridGeometry>;
 50     using Scalar = GetPropType<TypeTag, Properties::Scalar>;
 51     using type = OnePNISpatialParams<GridGeometry, Scalar>;
 52 };
 53 }
 54
 55
 56 template <class TypeTag>
 57 class OnePNIProblem : public PorousMediumFlowProblem<TypeTag>
 58 {
 59     using ParentType = PorousMediumFlowProblem<TypeTag>;
 60     using GridView = typename GetPropType<TypeTag,
Properties::GridGeometry>::GridView;
 61     using Scalar = GetPropType<TypeTag, Properties::Scalar>;
 62     using PrimaryVariables = GetPropType<TypeTag,
Properties::PrimaryVariables>;
 63     using FluidSystem = GetPropType<TypeTag,
Properties::FluidSystem>;
 64     using BoundaryTypes = GetPropType<TypeTag,
Properties::BoundaryTypes>;
 65     using NumEqVector = GetPropType<TypeTag,
Properties::NumEqVector>;
 66     using PointSource = GetPropType<TypeTag,
Properties::PointSource>;
 67     using IapwsH2O = Components::H2O<Scalar>;
 68     using ElementVolumeVariables = typename GetPropType<TypeTag,
Properties::GridVolumeVariables>::LocalView;
 69     using FVElementGeometry = typename GetPropType<TypeTag,
Properties::GridGeometry>::LocalView;
 70     using SubControlVolumeFace = typename
FVElementGeometry::SubControlVolumeFace;
 71     using SubControlVolume = typename
FVElementGeometry::SubControlVolume;
 72
 73     enum { dimWorld = GridView::dimensionworld };
 74
```

```
 75         // copy some indices for convenience
 76     using Indices = typename GetPropType<TypeTag,
Properties::ModelTraits>::Indices;
 77     enum {
 78         // indices of the primary variables
 79         pressureIdx = Indices::pressureIdx,
 80         temperatureIdx = Indices::temperatureIdx,
 81         //! Equation indices
 82         contiWEqIdx = Indices::conti0EqIdx,
 83         energyEqIdx = Indices::energyEqIdx,
 84         //! Phase indices (Single Liquid Phase)
 85         LiquidIdx = FluidSystem::comp0Idx
 86
 87     };
 88     using Element = typename GridView::template Codim<0>::Entity;
 89     using GlobalPosition = typename
Element::Geometry::GlobalCoordinate;
 90     using GridGeometry = GetPropType<TypeTag,
Properties::GridGeometry>;
 91
 92 public:
 93     OnePNIProblem(std::shared_ptr<const GridGeometry>
gridGeometry)
 94     : ParentType(gridGeometry)
 95     {
 96                 //initialize fluid system
 97                   FluidSystem::init();
 98         name_ = getParam<std::string>("Problem.Name");
 99
100     }
101
102     /*!
103      * \The problem name.
104      * Setting the prefix for simulation output files.
105      */
106     const std::string& name() const
107     {
108         return name_;
109     }
110     // \}
111
112     /*!
113
114      // #### Boundary conditions
115     // With the following function we define the __type of
boundary conditions__ depending on the location.
```

```
116     // Two types of boundary conditions can be specified:
Dirichlet or Neumann boundary conditions. On
117     // Dirichlet boundaries, the values of the primary variables
need to be fixed. On a Neumann boundaries,
118     // values for derivatives need to be fixed. Mixed boundary
conditions (different types for different
119     // equations on the same boundary) are not accepted for cell-
centered finite volume schemes.
120      */
121     BoundaryTypes boundaryTypesAtPos(const GlobalPosition
&globalPos) const
122     {
123           BoundaryTypes bcTypes;
124         if (globalPos[2] < eps_ || globalPos[2] > this-
>gridGeometry().bBoxMax()[2] - eps_)
125             bcTypes.setAllNeumann();
126         else
127             bcTypes.setAllDirichlet();
128
129         return bcTypes;
130     }
131
132     /*!
133      * \Evaluating the boundary conditions for a Dirichlet
boundary segment
134      *
135      * \param globalPos The position for which the bc type should
be evaluated
136      *
137      */
138     PrimaryVariables dirichletAtPos(const GlobalPosition
&globalPos) const
139     {
140
141         return initialAtPos(globalPos);
142     }
143
144    // On all Neumann boundaries, the boundary flux (whether mass
or energy flux) is zero.
145     NumEqVector neumannAtPos(const GlobalPosition &globalPos)
const
146     {
147         NumEqVector values(0.0);
148
149         return values;
150     }
151
```

```
152
153     /*!
154      * \Evaluating the initial value for a control volume.
155      *
156      * \param globalPos The position for which the initial
condition should be evaluated
157      *
158      * Inside this function, primary variables will be stored in
the parameter "values"
159      */
160     PrimaryVariables initialAtPos(const GlobalPosition &globalPos)
const
161     {
162             PrimaryVariables values(0.0);
163             Scalar densityW = 990; // Kg/m^3
164             Scalar depth = this->gridGeometry().bBoxMax()[2] -
globalPos[2];
165             // Hydrostatic Pressure
166             values[pressureIdx] = 100.0e5 - densityW*this-
>spatialParams().gravity(globalPos)[2]*depth; //Pascal
167             // Geothermal Gradient
168             values[temperatureIdx] = 323.6; //Kelvin = 50.45 degree
celsius
169
170             return values;
171     }
172
173
174    /*
175   * Adding the point source locations
176    */
177    void addPointSources(std::vector<PointSource>& pointSources)
const
178     {
179             // The injection well (source term)
180
181           pointSources.push_back(PointSource({1342, 1402, 15}));
182
183           // The production well (sink term)
184
185           pointSources.push_back(PointSource({1462, 1402, 15}));
186
187     }
188
189    // Using solution-dependent point sources
190   template<class ElementVolumeVariables>
191
```

```
192     void pointSource(PointSource& source,
193
194                      const Element &element,
195
196                      const FVElementGeometry& fvGeometry,
197
198                      const ElementVolumeVariables& elemVolVars,
199
200                      const SubControlVolume &scv) const
201
202     {
203
204         const auto& pos = source.position();
205
206         const auto& volVars = elemVolVars[scv];
207
208
209
210         if (pos[0] < 1350.0) // injection well
211
212         {
213
214           const Scalar volumeSource = 1.157407407e-3; //
injectionRate is positive and in m^3/s = 100 m^3/day
215
216           const Scalar massSource =
volumeSource*IapwsH2O::liquidDensity(293.15, volVars.pressure(0));
217
218           const Scalar energySource =
massSource*IapwsH2O::liquidEnthalpy(293.15, volVars.pressure(0));
219
220           source = NumEqVector({ massSource, energySource });
221
222         }
223
224         else // production well
225
226         {
227
228            const Scalar volumeSource = -1.157407407e-3; //
productionRate is negative and in m^3/s = 100 m^3/day
229
230            const Scalar massSource =
volumeSource*volVars.density(0); // using current water density of the
control volume
231
```

```
232             const Scalar energySource =
massSource*volVars.enthalpy(0); // using current water enthalpy of the
control volumne
233
234             source = NumEqVector({ massSource, energySource });
235
236         }
237
238
239
240
241     }
242
243
244
245 private:
246
247     static constexpr Scalar eps_ = 1e-6;
248     std::string name_;
249
250 };
251
252 } // end namespace Dumux
253
254 #endif // DUMUX_1PNI_PROBLEM_HH
```

Spatial Parameters File (spatialparams.hh)

```
1 #ifndef DUMUX_1PNI_SPATIAL_PARAMS_HH
2 #define DUMUX_1PNI_SPATIAL_PARAMS_HH
3
4 #include <dumux/porousmediumflow/properties.hh>
5 #include <dumux/material/spatialparams/fv1p.hh>
6
7
8 namespace Dumux {
9
10 template<class GridGeometry, class Scalar>
11 class OnePNISpatialParams
12 : public FVSpatialParamsOneP<GridGeometry, Scalar,
13                              OnePNISpatialParams<GridGeometry,
Scalar>>
14
15 {
16     using GridView = typename GridGeometry::GridView;
17     using ParentType = FVSpatialParamsOneP<GridGeometry, Scalar,
```

```
18
OnePNISpatialParams<GridGeometry, Scalar>>;
19     // get the dimensions of the simulation domain from GridView
20     static constexpr int dim = GridView::dimension;
21     static constexpr int dimWorld = GridView::dimensionworld;
22     using Element = typename GridView::template Codim<0>::Entity;
23     using GlobalPosition = typename
Element::Geometry::GlobalCoordinate;
24
25 public:
26     // export permeability type
27     using PermeabilityType = Dune::FieldMatrix<Scalar, dimWorld,
dimWorld>;
28     // The Constructor
29     OnePNISpatialParams(std::shared_ptr<const GridGeometry>
gridGeometry)
30     : ParentType(gridGeometry)
31     , aquiferK_(0)
32
33     {
34       // intrinsic permeabilities
35         aquiferK_[0][0] = 2.46e-13; // Permeability along X =
249.26 mD
36         aquiferK_[1][1] = 2.46e-13; // Permeability along Y =
249.26 mD
37         aquiferK_[2][2] = 9.87e-14; // Permeability along Z = 100
mD
38
39          // porosity
40         aquiferPorosity_ = 0.1;
41
42     }
43
44     /*!
45      * \ Defining the intrinsic permeability
46
47      */
48     PermeabilityType permeabilityAtPos(const GlobalPosition&
globalPos) const
49
50     {
51           return aquiferK_;
52
53     }
54
55    /*!
56      * \Defining the porosity
```

```
57
58       */
59     Scalar porosityAtPos(const GlobalPosition& globalPos) const
60     {
61          return aquiferPorosity_;
62
63     }
64
65
66
67     private:
68
69     static constexpr Scalar eps_ = 1e-6;
70
71     Dune::FieldMatrix<Scalar, dimWorld, dimWorld> aquiferK_;
72
73     Scalar aquiferPorosity_;
74 };
75
76 } // end namespace Dumux
77
78 #endif
```

Parameters/Input File (params.input)

```
1  [TimeLoop]
2  DtInitial = 10000 # [s]
3  TEnd = 311040000 # [s]
4  MaxTimeStepSize = 2592000
5
6  [Grid]
7  Positions0 = 0 400 960 1844 2404 2804
8  Positions1 = 0 400 960 1844 2404 2804
9  Positions2 = 0 30
10 Cells0 = 2 28 221 28 2
11 Cells1 = 2 28 221 28 2
12 Cells2 = 1
13
14 [Problem]
15 Name = 1pnifv # name passed to the output routines
16 EnableGravity = true # enable gravity
17
18 [Newton]
19 MaxRelativeShift = 1e-5
20 MaxTimeStepDivisions = 20
21
22 [Vtk]
```

```
23 AddVelocity = true # enable velocity output
24
25 [Component]
26 SolidDensity = 2750 # [Kg/m^3]
27 SolidThermalConductivity = 2.8 # [Watt/m.°K]
28 SolidHeatCapacity = 800 # [Joule/Kg.°K]
```

Main File (main.cc)

```
1 #include <config.h>
  2
  3 #include <ctime>
  4 #include <iostream>
  5
  6 #include <dune/common/parallel/mpihelper.hh>
  7 #include <dune/common/timer.hh>
  8 #include <dune/grid/io/file/vtk/vtksequencewriter.hh>
  9 #include <dune/grid/io/file/dgfparser/dgfexception.hh>
 10 #include <dune/grid/io/file/vtk.hh>
 11 #include <dune/istl/io.hh>
 12
 13 #include "problem.hh"
 14
 15 #include <dumux/common/properties.hh>
 16 #include <dumux/common/parameters.hh>
 17 #include <dumux/common/valgrind.hh>
 18 #include <dumux/common/dumuxmessage.hh>
 19
 20 #include <dumux/linear/seqsolverbackend.hh>
 21 #include <dumux/linear/linearsolvertraits.hh>
 22 #include <dumux/nonlinear/newtonsolver.hh>
 23
 24 #include <dumux/assembly/fvassembler.hh>
 25 #include <dumux/assembly/diffmethod.hh>
 26
 27 #include <dumux/discretization/method.hh>
 28
 29 #include <dumux/io/vtkoutputmodule.hh>
 30 #include <dumux/io/grid/gridmanager.hh>
 31 #include <dumux/io/loadsolution.hh>
 32
 33
 34
 35
 36 int main(int argc, char** argv) try
 37 {
 38     using namespace Dumux;
```

```
39
40      // we define a convenience alias for the type tag of this
problem. The type
41      // tags contain all the properties that are needed to define
the model and the problem
42      // setup. Throughout the main file, we will obtain types
defined for these type tags
43      // using the property system, i.e. with `GetPropType`.
44      using TypeTag = Properties::TTag::OnePNICCMpfa;
45
46      // initialization of MPI, finalization is automatically
executed on exit
47      const auto& mpiHelper = Dune::MPIHelper::instance(argc, argv);
48
49      // print dumux start message
50      if (mpiHelper.rank() == 0)
51          DumuxMessage::print(/*firstCall=*/true);
52
53      // for parsing command line arguments and input file
54      Parameters::init(argc, argv);
55
56
57      // the `GridManager` class creates the grid from data given in
the input file.
58      GridManager<GetPropType<TypeTag, Properties::Grid>>
gridManager;
59      gridManager.init();
60
61
62      // we compute on the leaf grid view
63      const auto& leafGridView = gridManager.grid().leafGridView();
64
65      // solving the single-phase problem
66      // first, a finite volume grid geometry is constructed from
the grid that was created above.
67      // this builds the sub-control volumes (scv) and sub-control
volume faces (scvf) for each element
68      // of the grid
69      using GridGeometry = GetPropType<TypeTag,
Properties::GridGeometry>;
70      auto gridGeometry =
std::make_shared<GridGeometry>(leafGridView);
71      gridGeometry->update();
72      // we now instantiate the problem, in which we define the
boundary and initial conditions.
73      using Problem = GetPropType<TypeTag, Properties::Problem>;
74      auto problem = std::make_shared<Problem>(gridGeometry);
```

```
 75     // we call the `computePointSourceMap` method to compute the
point sources.
 76     problem->computePointSourceMap();
 77
 78
 79     // get some time loop parameters
 80     using Scalar = GetPropType<TypeTag, Properties::Scalar>;
 81     const auto tEnd = getParam<Scalar>("TimeLoop.TEnd");
 82     const auto maxDt =
getParam<Scalar>("TimeLoop.MaxTimeStepSize");
 83     auto dt = getParam<Scalar>("TimeLoop.DtInitial");
 84
 85
 86     // the solution vector
 87     using SolutionVector = GetPropType<TypeTag,
Properties::SolutionVector>;
 88     SolutionVector x(gridGeometry->numDofs());
 89     problem->applyInitialSolution(x);
 90     auto xOld = x;
 91
 92     // variables of the grid
 93     using GridVariables = GetPropType<TypeTag,
Properties::GridVariables>;
 94     auto gridVariables = std::make_shared<GridVariables>(problem,
gridGeometry);
 95     gridVariables->init(x);
 96
 97     // intialize the vtk output module
 98     using IOFields = GetPropType<TypeTag, Properties::IOFields>;
 99     VtkOutputModule<GridVariables, SolutionVector>
vtkWriter(*gridVariables, x, problem->name());
100     using VelocityOutput = GetPropType<TypeTag,
Properties::VelocityOutput>;
101
vtkWriter.addVelocityOutput(std::make_shared<VelocityOutput>(*gridVari
ables));
102     vtkWriter.addVolumeVariable([] (const auto& v) { return
v.viscosity(); }, "viscosity (Pa s)");
103     IOFields::initOutputModule(vtkWriter);
104
105
106     // write initial solution
107     vtkWriter.write(0.0);
108
109     // instantiate time loop
110     auto timeLoop =
std::make_shared<CheckPointTimeLoop<Scalar>>(0.0, dt, tEnd);
```

```
111     timeLoop->setMaxTimeStepSize(maxDt);
112     timeLoop->setPeriodicCheckPoint(tEnd/30.0);
113
114     // for instationary problems, the assembler has a time loop
115     using Assembler = FVAssembler<TypeTag, DiffMethod::numeric>;
116     auto assembler = std::make_shared<Assembler>(problem,
gridGeometry, gridVariables, timeLoop, xOld);
117
118     // the linear solver
119      using LinearSolver =
AMGBiCGSTABBackend<LinearSolverTraits<GridGeometry>>;
120     auto linearSolver =
std::make_shared<LinearSolver>(leafGridView, gridGeometry-
>dofMapper());
121
122     // the non-linear solver
123     using NewtonSolver = Dumux::NewtonSolver<Assembler,
LinearSolver>;
124     NewtonSolver nonLinearSolver(assembler, linearSolver);
125
126     // time loop
127     timeLoop->start(); do
128     {
129
130         // linearize & solve
131         nonLinearSolver.solve(x, *timeLoop);
132
133         // make the new solution the old solution
134         xOld = x;
135         gridVariables->advanceTimeStep();
136
137         // move to the subsequent time step
138         timeLoop->advanceTimeStep();
139
140          // write the Vtk output on check points.
141         if (timeLoop->isCheckPoint())
142             vtkWriter.write(timeLoop->time());
143
144
145         // report time step stats
146         timeLoop->reportTimeStep();
147
148         // set new dt as suggested by the newton solver
149         timeLoop-
>setTimeStepSize(nonLinearSolver.suggestTimeStepSize(timeLoop-
>timeStepSize()));
150
```

```
151
152     } while (!timeLoop->finished());
153
154     timeLoop->finalize(leafGridView.comm());
155
156
157     // print dumux end message
158     if (mpiHelper.rank() == 0)
159     {
160         Parameters::print();
161         DumuxMessage::print(/*firstCall=*/false);
162     }
163
164     return 0;
165 } // end main
166 catch (Dumux::ParameterException &e)
167 {
168     std::cerr << std::endl << e << " ---> Abort!" << std::endl;
169     return 1;
170 }
171 catch (Dune::DGFException & e)
172 {
173     std::cerr << "DGF exception thrown (" << e <<
174                  "). Most likely, the DGF file name is wrong "
175                  "or the DGF file is corrupted, "
176                  "e.g. missing hash at end of file or wrong number
(dimensions) of entries."
177                  << " ---> Abort!" << std::endl;
178     return 2;
179 }
180 catch (Dune::Exception &e)
181 {
182     std::cerr << "Dune reported error: " << e << " ---> Abort!" <<
std::endl;
183     return 3;
184 }
185 catch (...)
186 {
187     std::cerr << "Unknown exception thrown! ---> Abort!" <<
std::endl;
188     return 4;
189 }
```

Configuration File (CMakeLists.txt)

```
# add a new finite volume 1pni test
dune_add_test(NAME convection_final
              SOURCES main.cc
              COMPILE_DEFINITIONS TYPETAG=OnePNICCMpfa)

# add a symlink for the input file in the build folder
dune_symlink_to_source_files(FILES "params.input")
```

REFERENCES

Aavatsmark, I. (2002). *An Introduction to Multipoint Flux Approximations for Quadrilateral Grids*. 28.

Aboulela, H., Amin, A., Lashin, A., & El Rayes, A. (2020). Contribution of geothermal resources to the future of renewable energy in Egypt: A case study, Gulf of Suez-Egypt. *Renewable Energy*, *167*, 248–265. https://doi.org/10.1016/j.renene.2020.11.079

Agarwal, H. (2021a, April 21). *C++ Data Types—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/c-data-types/

Agarwal, H. (2021b, June 1). *Inheritance in C++—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/inheritance-in-c/

Ahusborde, E., Amaziane, B., & Jurak, M. (2015). Three-dimensional numerical simulation by upscaling of gas migration through engineered and geological barriers for a deep repository for radioactive waste. *Geological Society, London, Special Publications*, *415*(1), 123–141. https://doi.org/10.1144/SP415.2

Ambrus, J., Maliska, C. R., Hurtado, F. S. V., & da Silva, A. F. C. (2010). Finite Volume Methods with Multi-Point Flux Approximation with Unstructured Grids for Diffusion Problems. *Defect and Diffusion Forum*, *297–301*, 670–675. https://doi.org/10.4028/www.scientific.net/DDF.297-301.670

Andrianov, N., & Nick, H. M. (2019). Modeling of waterflood efficiency using outcrop-based fractured models. *Journal of Petroleum Science and Engineering*, *183*, 106350. https://doi.org/10.1016/j.petrol.2019.106350

Bastian, P., Blatt, M., Dedner, A., Engwer, C., Fahlke, J., Gersbacher, C., Gräser, C., Grüninger, D., Kempf, R., Klöfkorn, S., Müthing, M., Nolte, M., Ohlberger, O., & Sander, O. (2021, May 9). *DUNE - DUNE Numerics*. DUNE. https://dune-project.org/

Beaude, L., Brenner, K., Lopez, S., Masson, R., & Smai, F. (2019). Non-isothermal compositional liquid gas Darcy flow: Formulation, soil-atmosphere boundary condition and application to high-energy geothermal simulations. *Computational Geosciences*, *23*(3), 443–470. https://doi.org/10.1007/s10596-018-9794-9

Boyle, J., Mihajlović, M., & Scott, J. (2010). HSL_MI20: An efficient AMG preconditioner for finite element problems in 3D: HSL_MI20 : AN EFFICIENT AMG PRECONDITIONER. *International Journal for Numerical Methods in Engineering*, *82*(1), 64–98. https://doi.org/10.1002/nme.2758

Bringedal, C. (2015). *Modeling of heat transfer in porous media in the context of geothermal energy extraction* [Dissertation for the degree of philosophiae doctor (PhD)]. University of Bergen.

Chen, T., Gewecke, N., Li, Z., Rubiano, A., Shuttleworth, R., Yang, B., & Zhong, X. (2009). *Fast Computational Methods for Reservoir Flow Models*. 20.

Class, H., Ebigbo, A., Helmig, R., Dahle, H. K., Nordbotten, J. M., Celia, M. A., Audigane, P., Darcis, M., Ennis-King, J., Fan, Y., Flemisch, B., Gasda, S. E., Jin, M., Krug, S., Labregere, D., Naderi Beni, A., Pawar, R. J., Sbai, A., Thomas, S. G., … Wei, L. (2009). A benchmark study on problems related to CO2 storage in geologic formations: Summary and discussion of the results. *Computational Geosciences*, *13*(4), 409–434. https://doi.org/10.1007/s10596-009-9146-x

Coltman, N., Lipp, M., Heck, K., Seitz, G., Koch, T., Schollenberger, T., Weinhardt, F., Flemisch, B., Ackermann, S., Schneider, M., Hommel, J., Mohammadi, F., Emmert, S., Veyskarami, M., Winter, R., & Hanchuan. (2021, May 18). *Exercises/exercise-grids · master · dumux-repositories / dumux-course · GitLab*. Dumux-Repositories.Gitlab. https://git.iws.uni-stuttgart.de/dumux-repositories/dumux-course/tree/master/exercises/exercise-grids

Cunningham, A. B., Class, H., Ebigbo, A., Gerlach, R., Phillips, A. J., & Hommel, J. (2019). Field-scale modeling of microbially induced calcite precipitation. *Computational Geosciences*, *23*(2), 399–414. https://doi.org/10.1007/s10596-018-9797-6

Dai, C., & Chen, Y. (2008). *Classification of Shallow and Deep Geothermal Energy*. *32*, 5.

El-Amin, M. F. (2019). Iterative Numerical Scheme for Non-Isothermal Two-Phase Flow in Heterogeneous Porous Media. *Algorithms*, *12*(6), 117. https://doi.org/10.3390/a12060117

Elemér, B. (2014). *Doublet Systems*. Digital Textbook Library. https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011_0059_SCORM_MFKGT5059-EN/sco_11_01.scorm

English, M. J. M. (2001). Physical principles of heat transfer. *Current Anaesthesia & Critical Care*, *12*(2), 66–71. https://doi.org/10.1054/cacc.2001.0331

Erbertseder, K., Reichold, J., Flemisch, B., Jenny, P., & Helmig, R. (2012). A Coupled Discrete/Continuum Model for Describing Cancer-Therapeutic Transport in the Lung. *PLoS ONE*, *7*(3), e31966. https://doi.org/10.1371/journal.pone.0031966

Fetzer, T., Smits, K. M., & Helmig, R. (2016). Effect of Turbulence and Roughness on Coupled Porous-Medium/Free-Flow Exchange Processes. *Transport in Porous Media*, *114*(2), 395–424. https://doi.org/10.1007/s11242-016-0654-6

Flemisch, B. (2013). *Tackling Coupled Problems in Porous Media: Development of Numerical Models and an Open Source Simulator* [Habilitation thesis]. University of Stuttgart.

Flemisch, B. (2018). *The DuMuX Property System*. https://git.iws.uni-stuttgart.de/dumux-repositories/dumux-course/-/blob/master/slides/dumux-course-properties.pdf

Flemisch, B., & Class, H. (2019). *Introduction to DuMux: Overview and Available Models*. https://git.iws.uni-stuttgart.de/dumux-repositories/dumux-course/-/blob/master/slides/dumux-course-intro.pdf

Fontes, E. (2018, November 29). *FEM vs. FVM | COMSOL Blog*. COMSOL - Software for Multiphysics Simulation. https://www.comsol.com/blogs/fem-vs-fvm/

Fourno, A., Ngo, T.-D., Noetinger, B., & La Borderie, C. (2019). FraC: A new conforming mesh method for discrete fracture networks. *Journal of Computational Physics*, *376*, 713–732. https://doi.org/10.1016/j.jcp.2018.10.005

Ganguly, S., Tan, L., Date, A., & Kumar, M. (2017a). Numerical Investigation of Temperature Distribution in a Confined Heterogeneous Geothermal Reservoir Due to Injection-production. *Energy Procedia*, *110*, 143–148. https://doi.org/10.1016/j.egypro.2017.03.119

Ganguly, S., Tan, L., Date, A., & Kumar, M. S. M. (2017b). Effect of Heat Loss in a Geothermal Reservoir. *Energy Procedia*, *110*, 77–82. https://doi.org/10.1016/j.egypro.2017.03.109

GeeksforGeeks. (2021a, April 13). *Templates in C++—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/templates-cpp/

GeeksforGeeks. (2021b, April 26). *Template Specialization in C++—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/template-specialization-c/

GeeksforGeeks. (2021c, June 10). *Structure vs class in C++—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/structure-vs-class-in-cpp/

Gläser, D., Helmig, R., Flemisch, B., & Class, H. (2017). A discrete fracture model for two-phase flow in fractured porous media. *Advances in Water Resources*, *110*, 335–348. https://doi.org/10.1016/j.advwatres.2017.10.031

Goldstein, B., Hiriart, G., Tester, J., Gutierrez-Negrin, L., Bertani, R., Bromley, C., Huenges, E., Ragnarsson, A., Mongillo, M., Lund, J. W., Rybach, L., Zui, V., & Muraoka, H. (2013). Geothermal Energy geothermal energy , Nature geothermal energy nature , Use geothermal energy use , and Expectations geothermal energy expectations. In M. Kaltschmitt, N. J. Themelis, L. Y. Bronicki, L. Söder, & L. A. Vega (Eds.), *Renewable Energy Systems* (pp. 772–782). Springer New York. https://doi.org/10.1007/978-1-4614-5820-3_309

Hagemann, B., Rasoulzadeh, M., Panfilov, M., Ganzer, L., & Reitenbach, V. (2016). Hydrogenization of underground storage of natural gas: Impact of hydrogen on the hydrodynamic and bio-chemical behavior. *Computational Geosciences*, *20*(3), 595–606. https://doi.org/10.1007/s10596-015-9515-6

Heck, K., Coltman, E., Schneider, J., & Helmig, R. (2020). Influence of Radiation on Evaporation Rates: A Numerical Analysis. *Water Resources Research*, *56*(10). https://doi.org/10.1029/2020WR027332

Hommel, J., Lauchnor, E., Phillips, A., Gerlach, R., Cunningham, A. B., Helmig, R., Ebigbo, A., & Class, H. (2015). A revised model for microbially induced calcite precipitation: Improvements and new insights based on recent experiments: A MODEL FOR MICP: IMPROVEMENTS AND NEW INSIGHTS. *Water Resources Research*, *51*(5), 3695–3715. https://doi.org/10.1002/2014WR016503

Kariya, A. (2021, May 17). *C++ Classes and Objects—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/c-classes-and-objects/

Kempka, T., Class, H., Görke, U.-J., Norden, B., Kolditz, O., Kühn, M., Walter, L., Wang, W., & Zehner, B. (2013). A Dynamic Flow Simulation Code Intercomparison based on the Revised Static Model of the Ketzin Pilot Site. *Energy Procedia*, *40*, 418–427. https://doi.org/10.1016/j.egypro.2013.08.048

Kitware. (2021, June). *About—Kitware Inc.* Home - Kitware Inc. https://www.kitware.com/about/

Koch, T., Flemisch, B., Helmig, R., Wiest, R., & Obrist, D. (2020). A multiscale subvoxel perfusion model to estimate diffusive capillary wall conductivity in multiple sclerosis lesions from perfusion MRI data. *International Journal for Numerical Methods in Biomedical Engineering*, *36*(2). https://doi.org/10.1002/cnm.3298

Koch, T., Gläser, D., Weishaupt, K., Ackermann, S., Beck, M., Becker, B., Burbulla, S., Class, H., Coltman, E., Emmert, S., Fetzer, T., Grüninger, C., Heck, K., Hommel, J., Kurz, T., Lipp, M., Mohammadi, F., Scherrer, S., Schneider, M., … Flemisch, B. (2020). DuMux 3 – an open-source simulator for solving flow and transport problems in porous media with a focus on model coupling. *Computers & Mathematics with Applications*, *81*, 423–443. https://doi.org/10.1016/j.camwa.2020.02.012

Koch, T., Heck, K., Schröder, N., Class, H., & Helmig, R. (2018). A New Simulation Framework for Soil-Root Interaction, Evaporation, Root Growth, and Solute Transport. *Vadose Zone Journal*, *17*(1), 170210. https://doi.org/10.2136/vzj2017.12.0210

Koch, T., & Schneider, M. (2015, June 12). *Grid Adaptation with DuMuX*.

Kretzschmar, H.-J., Herrmann, S., Kunick, M., & Posselt, J. (2018, June 18). *IAPWS Educational Resources*. International Association for the Properties of Water and Steam. http://www.iapws.org/edu.html

Limberger, J., Boxem, T., Pluymaekers, M., Bruhn, D., Manzella, A., Calcagno, P., Beekman, F., Cloetingh, S., & van Wees, J.-D. (2018). Geothermal energy in deep aquifers: A global assessment of the resource base for direct heat utilization.

Renewable and Sustainable Energy Reviews, *82*, 961–975. https://doi.org/10.1016/j.rser.2017.09.084

Mahbaz, S. B., Yaghoubi, A., Dehghani-Sanij, A., Sarvaramini, E., Leonenko, Y., & Dusseault, M. B. (2021). Well-Doublets: A First-Order Assessment of Geothermal SedHeat Systems. *Applied Sciences*, *11*(2), 697. https://doi.org/10.3390/app11020697

Mai, T. H., Schnepf, A., Vereecken, H., & Vanderborght, J. (2019). Continuum multiscale model of root water and nutrient uptake from soil with explicit consideration of the 3D root architecture and the rhizosphere gradients. *Plant and Soil*, *439*(1–2), 273–292. https://doi.org/10.1007/s11104-018-3890-4

Mehl, S., Hill, M. C., & Leake, S. A. (2006). Comparison of Local Grid Refinement Methods for MODFLOW. *Ground Water*, *44*(6 Understanding), 792–796. https://doi.org/10.1111/j.1745-6584.2006.00192.x

Menezes Farias, M., Fraxe, T., Bento Cavalcante Neto, J., Marcondes, F., & Sepehrnoori, K. (2019). COMPARISON OF STRUCTURED CORNER POINT AND UNSTRUCTURED GRIDS FOR COMPOSITIONAL RESERVOIR SIMULATION. *Proceedings of the 25th International Congress of Mechanical Engineering*. 25th International Congress of Mechanical Engineering. https://doi.org/10.26678/ABCM.COBEM2019.COB2019-2381

Moog, G. (2013). *Advanced Discretization Methods for Flow Simulation Using Unstructured Grids* [Doctor of Philosophy]. Stanford University.

Moortgat, J. (2017). Adaptive implicit finite element methods for multicomponent compressible flow in heterogeneous and fractured porous media: AIM FOR HETEROGENEOUS FRACTURED MEDIA. *Water Resources Research*, *53*(1), 73–92. https://doi.org/10.1002/2016WR019644

Mosthaf, K., Baber, K., Flemisch, B., Helmig, R., Leijnse, A., Rybak, I., & Wohlmuth, B. (2011). A coupling concept for two-phase compositional porous-medium and single-phase compositional free flow: COUPLING TWO-PHASE COMPOSITIONAL POROUS-MEDIUM AND FREE FLOW. *Water Resources Research*, *47*(10). https://doi.org/10.1029/2011WR010685

Negara, A., Salama, A., & Sun, S. (2014). Density-Driven Flow Simulation in Anisotropic Porous Media: Application to CO2 Geological Sequestration. *All Days*, SPE-172232-MS. https://doi.org/10.2118/172232-MS

Nordbotten, J. M., & Eigestad, G. T. (2005). Discretization on quadrilateral grids with improved monotonicity properties. *Journal of Computational Physics*, *203*(2), 744–760. https://doi.org/10.1016/j.jcp.2004.10.002

Nordbotten, J. M., Flemisch, B., Gasda, S. E., Nilsen, H. M., Fan, Y., Pickup, G. E., Wiese, B., Celia, M. A., Dahle, H. K., Eigestad, G. T., & Pruess, K. (2012). Uncertainties in practical simulation of CO2 storage. *International Journal of Greenhouse Gas Control*, *9*, 234–242. https://doi.org/10.1016/j.ijggc.2012.03.007

Ocłoń, P., Łopata, S., & Nowak, M. (2013). Comparative study of conjugate gradient algorithms performance on the example of steady-state axisymmetric heat transfer problem. *Archives of Thermodynamics*, *34*(3), 15–44. https://doi.org/10.2478/aoter-2013-0013

Ouali, S., Hazmoune, M., & Bouzidi, K. (2015). *LOW TEMPERATURE GEOTHERMAL ENERGY FOR RURAL DEVELOPMENT*. 7.

ParaView. (2021, June). *Overview | ParaView*. ParaView. https://www.paraview.org/overview/

Polyak, B. T. (2007). Newton's method and its use in optimization. *European Journal of Operational Research*, *181*(3), 1086–1096. https://doi.org/10.1016/j.ejor.2005.06.076

Prabhu, R. (2019, November 13). *Variables in C++—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/variables-in-c/

Programiz. (n.d.). *C++ Templates*. Programiz: Learn to Code for Free. Retrieved June 12, 2021, from https://www.programiz.com/cpp-programming/templates

Sahu, R., Borban, C., Bhawsar, K., Arya, R., Makh, S., Chandelkar, V., & Dawande, H. (2018). *Performance and Analysis of Thermal Conductivity of Metal Rod*. *5*, 2.

Sander, O. (2020). *DUNE - The Distributed and Unified Numerics Environment* (Vol. 140). Springer, Cham. https://link.springer.com/chapter/10.1007%2F978-3-030-59702-3_4

Schlumberger. (2017). *ECLIPSE Technical Description*. Schlumberger.

Scholz, S., Heck, K., & Lipp, M. (2018). *Setting up a test problem / application and using the build system (CMake)*. https://git.iws.uni-stuttgart.de/dumux-repositories/dumux-course/-/blob/master/slides/dumux-course-problem.pdf

Schwenck, N., Flemisch, B., Helmig, R., & Wohlmuth, B. I. (2015). Dimensionally reduced flow models in fractured porous media: Crossings and boundaries. *Computational Geosciences*, *19*(6), 1219–1230. https://doi.org/10.1007/s10596-015-9536-1

Sergey, & W.F. (2017, September 8). *C++—Partial template specialization | c++ Tutorial*. Learn Programming Languages with Books and Examples. https://riptutorial.com/cplusplus/example/6253/partial-template-specialization

Stadler, L., Hinkelmann, R., & Helmig, R. (2012). Modeling Macroporous Soils with a Two-Phase Dual-Permeability Model. *Transport in Porous Media*, *95*(3), 585–601. https://doi.org/10.1007/s11242-012-0064-3

Starnoni, M., Berre, I., Keilegavlen, E., & Nordbotten, J. M. (2019). Consistent MPFA Discretization for Flow in the Presence of Gravity. *Water Resources Research*, *55*(12), 10105–10118. https://doi.org/10.1029/2019WR025384

Støverud, K. H., Darcis, M., Helmig, R., & Hassanizadeh, S. M. (2012). Modeling Concentration Distribution and Deformation During Convection-Enhanced Drug Delivery into Brain Tissue. *Transport in Porous Media*, *92*(1), 119–143. https://doi.org/10.1007/s11242-011-9894-7

Tatomir, A., Dimache, A.-N., Iulian, I., & Sauter, M. (2019). Modelling of CO_2 storage in geological formations with DuMu^x, a free-open-source numerical framework. A possible tool to assess geological storage of carbon dioxide in Romania. *E3S Web of Conferences*, *85*, 07002. https://doi.org/10.1051/e3sconf/20198507002

The DuMux developers. (2020a, November). *DuMuX: Cell-centered FV scheme*. https://dumux.org/docs/doxygen/releases/3.3/a01720.html

The DuMux developers. (2020b, November). *DuMuX: Dumux::AMGBiCGSTABBackend< LinearSolverTraits > Class Template Reference*. https://dumux.org/docs/doxygen/releases/3.3/a06314.html

The DuMux developers. (2020c, November). *DuMuX: Dumux::Components::H2O< Scalar > Class Template Reference*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a06662.html

The DuMux developers. (2020d, November). *DuMuX: Dumux::FluidSystems::OnePLiquid< Scalar, ComponentT > Class Template Reference*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a07270.html

The DuMux developers. (2020e, November). *DuMuX: Dumux::Properties::TTag::OnePNI Struct Reference*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a08254.html#details

The DuMux developers. (2020f, November). *DuMuX: Free Flow Models*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a01702.html

The DuMux developers. (2020g, November). *DuMuX: Geomechanics Models*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a01714.html

The DuMux developers. (2020h, November). *DuMuX: Multidomain simulations*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a01755.html

The DuMux developers. (2020i, November). *DuMuX: parameterlist.txt File Reference*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a00005.html

The DuMux developers. (2020j, November). *DuMuX: Porous-Medium Flow Models*. DuMux. https://dumux.org/docs/doxygen/releases/3.3/a01676.html

The DuMux developers. (2021). *Dumux handbook v3.3*. https://dumux.org/docs/handbook/releases/3.3/dumux-handbook.pdf

Tiwari, A. (2019, August 8). *Namespace in C++ | Set 1 (Introduction)—GeeksforGeeks*. GeeksforGeeks | A Computer Science Portal for Geeks. https://www.geeksforgeeks.org/namespace-in-c/

Vinsome, P. K. W. (1976). Orthomin, an Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations. *All Days*, SPE-5729-MS. https://doi.org/10.2118/5729-MS

W3Schools. (n.d.). *C++ Access Specifiers*. W3Schools Online Web Tutorials. Retrieved June 11, 2021, from https://www.w3schools.com/cpp/cpp_access_specifiers.asp

Walter, L., Binning, P. J., Oladyshkin, S., Flemisch, B., & Class, H. (2012). Brine migration resulting from CO2 injection into saline aquifers – An approach to risk estimation including various levels of uncertainty. *International Journal of Greenhouse Gas Control*, *9*, 495–506. https://doi.org/10.1016/j.ijggc.2012.05.004

Weishaupt, K., Bordenave, A., Atteia, O., & Class, H. (2016). Numerical Investigation on the Benefits of Preheating for an Increased Thermal Radius of Influence During

Steam Injection in Saturated Soil. *Transport in Porous Media*, *114*(2), 601–621. https://doi.org/10.1007/s11242-016-0624-z

Zhou, D., Tatomir, A., Tomac, I., & Sauter, M. (2020). Verification benchmark for a single-phase flow hydro—Mechanical model comparison between COMSOL Multiphysics and DuMu ^{X}. *E3S Web of Conferences*, *205*, 02002. https://doi.org/10.1051/e3sconf/202020502002

Zhou, P. (1993). Elements and Shape Functions. In P. Zhou, *Numerical Analysis of Electromagnetic Fields* (pp. 172–211). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-50319-1_6