NTNU – Trondheim
Norwegian University of
Science and Technology

Simulations of CO_2 Migration with a Fully-Integrated VE Model on the GPU

Guro Seternes

Master of Science in Physics and Mathematics
Submission date: Januar 2015
Supervisor: Helge Holden, MATH
Co-supervisor: Knut-Andreas Lie, SINTEF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

The reduction of carbon dioxide emissions is becoming an increasing global priority and is the subject of many current research projects. One of the possible solutions is geological carbon storage, in which CO_2 is captured and injected into geological underground reservoirs for permanent storage. An assessment of the associated CO_2 leakage risks is crucial when evaluating potential storage sites. By simulating the movement of CO_2 during and in the aftermath of the injection we can determine these risks. In this thesis we present a CO_2 migration simulator made for this purpose.

There are often great uncertainties in the available geological data required to make realistic simulations. This means that one has to be able to evaluate multiple scenarios within a relatively short time frame, putting performance requirements on the simulator. Because the temporal and spatial scales in question are very big, the full 3D models commonly applied in the related branch of oil and gas simulators are too computationally demanding. Thus, our simulator is based on a 2D VE model. What sets our simulator apart from other VE-based simulators, is that we include non-linearized CO_2 properties. This means that the vertical integrals which constitute the VE model are no longer trivial expressions, they must be "fully-integrated". Through GPU acceleration we implement a simulator that runs just as fast as other CPU-based VE simulators, in spite of the tedious numerical integrations. The benefits of GPU-acceleration are emphasized when simulating on large domains. Moreover we make a comparison study with a VE-based simulator in which the CO_2 properties are linearized. The optimization potential and different optimization strategies directed at the GPU implementation of the numerical integrations are also elaborated.

Sammendrag

Karbondioksidnivået i atmosfæren vår er i ferd med å nå et kritisk punkt. Å redusere utslippene av denne klimagassen blir en stadig viktigere oppgave for verdensamfunnet og er et høyaktuelt tema innenfor mange forskningsfelter. Et mulig bidrag for å løse dette problemet er karbonfangst og -lagring, hvor man fanger CO_2 og sprøyter det inn i underjordiske geologiske reservoarer. Denne lagringen innebærer en risiko for CO_2-lekkasje, hvilket betyr at det er viktig å gjøre en grundig utredning av potensielle reservoarer. Ved å simulere bevegelsesmønsteret til CO_2 både under og etter injeksjonsfasen kan man kartlegge risikoen for lekkasje. Dette er formålet for CO_2-migrasjonssimulatoren som vi presenterer i denne oppgaven.

Det er generelt mye usikkerhet tilknyttet de geologiske dataene man anvender for å kunne lage realistiske simuleringer. På bakgrunn av dette må man kunne evaluere flere ulike senariorer innenfor en relativt liten tidsramme, noe som setter begrensinger for kjøretiden til simulatoren. Tids- og størrelsesperpektivet for denne typen simuleringer er mye større enn for simuleringer av samme art som brukes i forbindelse med olje- og gassutvinning. Dette betyr at de fulle 3D-modellene som vanligvis brukes der er uegnet for vårt problem, ettersom de krever altfor mye datakraft for å oppfylle tidsbegrensingen. Vår simulator er derfor basert på en 2D VE- modell. Det som skiller vår simulator fra andre tilsvarende VE-simulatorer er at vi inkluderer ikke-lineære egenskaper for å beskrive CO_2-en. Dette innebærer at de vertikale integraltermene som ligger til grunn for VE-modellen ikke lenger er trivielle. Ved hjelp av "GPU acceleration" implementerer vi en simulator som er vel så rask som andre CPU-baserte VE-modeller, til tross for den omstendelige numeriske integrasjonen. Fordelene ved å anvende en GPU blir enda klarere når vi kjører simuleringer på store modeller. Videre presenterer vi en sammenligningsstudie mellom vår simulator og en simulator med lineariserte CO_2-egenskaper. Vi diskuterer også optimaliseringspotensialet for vår simulator, samt ulike optimaliseringsstrategier med hensyn på den numeriske integrasjonen.

Preface

This document is my thesis for the Master's program Industrial Mathematics at the Norwegian University of Science and Technology (NTNU), where my field of specialization has been numerical mathematics. It is based on the specialization project I wrote in Trondheim in the spring of 2014 in cooperation with the independent research organization SINTEF. This cooperation was continued when I moved to Oslo to write my thesis at their offices in the period September 2014- January 2015.

Firstly, I would like to thank my two supervisors at SINTEF, Knut-Andreas Lie and André Brodtkorb, who answered numerous emails in the spring and accepted many office visits in the fall. They also contributed by looking over, commenting and correcting my thesis several times. I would also like to thank Halvor M. Nilsen, my CO_2 modelling expert at SINTEF, for many helpful discussions and white board drawings in his office. The parts of my thesis which Knut-Andreas noted were well-written were based on the notes I wrote in Halvor's office.

My master office companion, Jens, also deserves some credit, helping me with any computer-related problem. Especially when, four weeks before my deadline, my computer crashed. I would also like to thank Tarjei, not only for support during frustrating periods in the lapse of the fall, but also for agreeing to read my thesis numerous times and correcting my notorious comma mistakes. Lastly I would like to mention Agnes and Torbjørn, my faithful study companions through the five years at NTNU. I could not have completed my degree without you.

Guro Seternes

Oslo, January 28, 2015

Contents

Chapter 1

Introduction

The amount of anthropogenic carbon dioxide in our atmosphere is reaching a critical level. There is scientific consensus that the average global temperature on earth has a clear correlation to the concentration of greenhouse gases in the earth's atmosphere. Thus, it is believed that the most abundant greenhouse gas, CO_2, is one of the main causes of global warming and the resulting climate changes. The consequences of the ongoing climate changes are not purely environmental: they include poverty, economic development, human health and agriculture. Reduction of CO_2 has therefore become a high priority for the UN and there are now many research projects dedicated at finding new strategies to resolve this problem. In this thesis, we will look at one such promising strategy, namely CO_2 sequestration.

Simulating Carbon Dioxide Sequestration CO_2 sequestration refers to the capture and permanent storage of CO_2 with the objective of reducing the level of CO_2 in the atmosphere. This approach is considered a good option for handling CO_2 emissions in the transition phase before renewable energy sources hopefully will dominate. The idea is to capture CO_2 during combustion and exhaustion processes and then transport the CO_2 to a suitable storage site where it can be injected. The injection procedure is closely related to enhanced oil and gas recovery, where water and (or) gas are introduced into a reservoir. Hence, the associated technology is familiar and it is compatible with the existing energy infrastructure. From an economical point of view CO_2 storage is also considered realizable [4], especially for petroleum companys, where there is an extra economic incentive due to carbon taxation. Deep saline aquifers are considered to be prime candidates for underground or geological storage due to their large capacities and wide geological spread [15, 16, 32].

Before attempting to go through with the CO_2 sequestration technology, it is crucial to assess the associated risks of leaks. This is the motivation for studying and simulating the behavior of CO_2 during both the injection phase and, even more so, the following migration phase. Different underground formations have different geological properties and their storage potential can be assessed using numerical simulations. Because of large uncertainties in the geology, a fast simulator is required to evaluate multiple scenarios within a short time frame [35].

The system we will study can be described as two-phase flow in heterogeneous porous media, where the two phases are the intruding CO_2 and the water already residing in the formation.

Vertical Equilibrium Model Permeable subsurface formations extend much more in the horizontal direction than in the vertical direction. The lateral extent is often many kilometers, while the vertical extent is in the range of meters to tens of meters. This means that the flow in the horizontal direction, on the time scales of interest for CO_2 sequestration, is usually much greater than the vertical one. In the formation, there is a boundary separating the saline aquifer with higher permeability from the aquitard with lower permeability. By the law of tangent flow, the flow in the higher permeability region tends toward the direction parallel to the interface [44]. Also, due to the character of the fluids in question, there is a strong vertical segregation.

These properties indicate that it may be wise to concentrate on the horizontal flow movement. Thus, we make an assumption of vertical equilibrium (VE) allowing us to reduce the problem down to two dimensions. We will do this by means of vertical integration, where the original three-dimensional equations are integrated along the vertical direction. Through vertical integration, we obtain a new set of equations with new parameter functions that depend on the vertical distribution of the fluids.

The VE model is not a new concept, it has been used for 150 years or so to simplify systems of flow in porous media [37, 3]. With the great development of computer resources in the last decades, this semi-analytical approach has mostly been set aside. However, this model has recently been used as a means to simulate gravity-driven CO_2 migration [44]. The problem with CO_2 simulation, in contrast to oil and gas recovery, is that the scales are very large, both with respect to time and space. As already mentioned, the formation can span many kilometers, possibly hundreds, and the time perspective for these migration processes can be thousands of years. This makes it challenging to make full three-dimensional (3D) simulations that are fast enough. One is often forced to compromise the vertical and lateral resolution for speed. For this reason, the two-dimensional (2D) VE model may be a better candidate. Several studies have shown that the VE models match their 3D counterpart [35, 40, 12]. In fact, in some situations the VE model produces more accurate results due to the overly coarse resolution required for 3D simulations.

Application of the VE model to our two-phase system will leave us with a set of two equations: a pressure equation and a transport equation. There are numerous papers which discuss the details of this system, see for example [23] and [21]. One of the strengths of the VE formulation is that it preserves information about geological heterogeneity. In addition, the resulting equation system has a stronger decoupling than the corresponding 3D system.

Sharp-Interface Models Many VE simulators make an assumption of a sharp interface between the injected CO_2 and the formation water. This further simplifies the equation system, as some of the vertical integration terms in the VE model become trivial, which again increases the computational efficiency. However, we may loose some information. A main objective for this thesis is to explore the limitations of the popular sharp-interface model. We will apply a more extensive framework where we instead make a more realistic, smooth approximation of the transition zone between the two phases, see Figure 1.1. Many of the components in this framework are based on the paper by Nilsen et al. [41]. As we will see, this new framework complicates the mathematical model as the integrated terms are no longer trivial. Consequently, the number of required computations increases substantially. We intend to compensate for this increased computational cost by implementing the simulator on a GPU.

Fully-Integrated Model on a GPU Through the last decade, what is known as GPU-accelerated computing, has become a big phenomenon in the fields of science, engineering and enterprise. The term refers to the coupling between the CPU and the graphics processing unit (GPU) used to accelerate the performance of a computer application. GPU hardware is highly parallel and can perform similar operations on large amounts of data. The key to GPU-accelerated computing is to offload compute-intensive portions of an application to the GPU, such as numerical integration. Thus, through GPU-acceleration, we can implement a simulator for the smooth-approximation

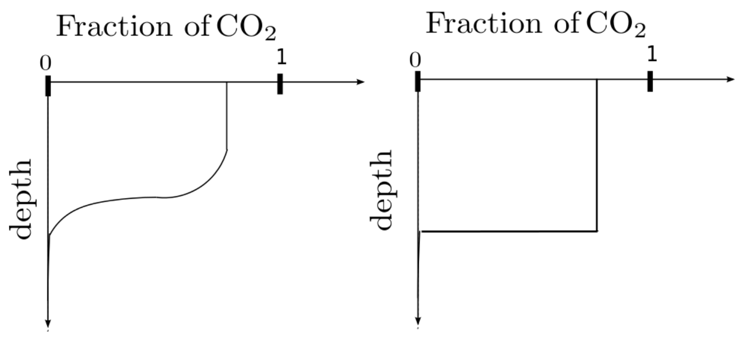


Figure 1.1: Fraction of CO_2 as a function of depth. On the left, the transition zone between the two phases is smooth, while on the right they are separated by a sharp interface.

model, which has more than comparative performance to the sharp-interface CPU simulators. At least our aim will be to implement a GPU simulator with decent performance, not only with respect to outrunning the sharp-interface solvers but perhaps more importantly with respect to utilization of the GPU capacity.

For the comparison study of the two approaches, we will use an explicit sharp-interface simulator from the open-source Matlab Reservoir Simulation Toolbox (MRST) [1].

Chapter 2

Background

2.1 Mathematical Model

To be able to analyze the system of fluids arising in an aquifer when CO_2 is injected, we have to build a mathematical model. This includes developing a valid mathematical description of the physics of the reservoir as well as establishing a set of governing equations. As mentioned in the introduction, our system involves multiple phases, but we will start by looking at a single phase and later advance the analysis to include multiple phases.

2.1.1 Law of Mass Conservation

A starting point for the mathematical model is the law of mass conservation. The principle of conservation of mass is quite simple; for a restricted region of space, the change of mass of a particular substance must equal the amount of mass that either leaves or enters the region. If we denote this restricted region or volume by Ω and the boundary by $\partial\Omega$, the mathematical equation for conservation of mass reads

$$\frac{\partial m}{\partial t} = \int_{\Omega} \frac{\partial \rho}{\partial t} \, \mathrm{d}V = -\oint_{\partial\Omega} \rho v \cdot \mathbf{n} \, \mathrm{d}A + \int_{\Omega} q \, \mathrm{d}V. \tag{2.1}$$

Here, m represents the mass of the substance in the volume unit, v is the mass flow rate, ρ is the fluid density and $\mathbf{n}$ is the unit vector normal to the surface $\partial\Omega$ in the outward direction. The variable q is any source or sink within the volume. If q indicates internal changes to the system, our equation is known as a balance equation or transport equation. Contrary, if the sources and sinks originate from known external sources or do not exist, the mass is locally conserved and thus, Equation (2.1) may be referred to as a conservation law.

Since we have a porous medium, the fluid will not fill the entire volume, and thus, the mass in (2.1) can not be described by integrating over the density alone. To account for this factor in the conservation law, we introduce the porosity function ϕ, which gives the fraction of the sample volume that is occupied by pore space. Keeping in mind that the medium in question may not be homogeneous, the porosity function will depend on spatial location, $\phi = \phi(x)$.

If we apply the divergence theorem to Equation (2.1) and make an assumption of sufficient smoothness, we will arrive at the differential form

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = q, \tag{2.2}$$

where the flux $\rho v \cdot \mathbf{n}$ has been reduced to $\rho\mathbf{u}$.

2.1.2 Darcy's law

Mass conservation alone does not allow for a unique solution for the single-phase system. We must also study the basic forces involved in the movement of fluids in a porous medium. Darcy's law is a key equation for determining the flow characteristics,

$$\mathbf{u} = -\frac{\mathbf{k}}{\mu}\left(\nabla p - \rho\mathbf{g}\right), \tag{2.3}$$

where $\mathbf{u}$ represents the flux, $\mathbf{k}$ is the permeability, p is the pressure, μ is the viscosity and $\mathbf{g}$ indicates the gravitational acceleration. Permeability is a measure of the ability of a porous material to allow fluids to pass through it, and will depend on the type of rock. We will assume that the permeability can be decomposed into a horizontal permeability tensor and a scalar vertical permeability [45]. Thus, $\mathbf{k}$ can be expressed by the block matrix

$$\begin{pmatrix} \mathbf{k}_{\|} & 0 \\ 0 & k_z \end{pmatrix}.$$

By Equations (2.2) and (2.3) we have established a mathematical model for a system of single-phase flow in a porous medium.

2.1.3 Two-phase Flow Equations

We can extend the equation system presented above to describe two-phase flow if we take into consideration the interactions between the two phases. In particular in the areas where the fluids meet and create fluid-fluid interfaces at the pore scale. In such systems, one of the fluids will have a tendency to be more attracted to the solid. This is known as the wetting fluid. In our case this will be the formation water, which is known as *brine* and thus, CO_2 will be referred to as the non-wetting fluid. At the depths in question, the CO_2 will be in a supercritical dense phase.

Darcy and the Mass Conservation Law

Seeing as we now have two phases, we need to adjust the mass term $\rho\phi$ in (2.2), since the volume of the pores is now shared. We include the variable s_α, which is known as the fluid saturation, defined as the fraction of pore space occupied by the phase $\alpha = \{w, n\}$, where $0 \leq s_\alpha \leq 1$. The two fluids fill the pore space completely:

$$s_n + s_w = 1. \tag{2.4}$$

Therefore, the expression ϕs_α now gives the volume fraction occupied by each phase. The mass conservation for the multiphase system becomes

$$\frac{\partial(\rho_\alpha \phi_\alpha s_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \mathbf{u}_\alpha) = q_\alpha, \quad \alpha = \{w, n\}, \tag{2.5}$$

i.e., the mass of each phase is conserved.

In the multiphase system, the fluids will inhibit each other from flowing. We account for this in Darcy's law by introducing the relative permeability function $k_{r,\alpha} = k_{r,\alpha}(s_\alpha)$. This function describes the reduced flow of phase α, caused by the pore-occupation of another phase β. The function is assumed algebraic and to depend only on the saturation and the saturation history. Thus, (2.3) becomes

$$\mathbf{u}_\alpha = -\frac{\mathbf{k} k_{r\alpha}}{\mu_\alpha} \left(\nabla p_\alpha + \rho_\alpha \mathbf{g}\right) = -\mathbf{k}\lambda_\alpha \left(\nabla p_\alpha - \rho_\alpha \mathbf{g}\right), \tag{2.6}$$

where we have introduced the mobility function $\lambda_\alpha(s_\alpha) \equiv \frac{k_{r,\alpha}(s_\alpha)}{\mu_\alpha}$.

Another physical property of the porous medium which needs to be taken into consideration is capillary pressure. This is defined as the pressure difference between the two different phases inside the rock,

$$p_{cap} = p_n - p_w. \tag{2.7}$$

When a non-wetting fluid is injected into a rock filled with a resident wetting fluid, the pores in the rock are affected by this pressure difference and will start deforming, followed by destabilization of the pore interfaces. If the capillary pressure becomes high enough, it will exceed what is known as the capillary entry pressure, $p_{cap,i}$. This is the level at which the non-wetting CO_2 starts displacing the wetting brine, forcing it to leave the region through the bottom boundary, implying a change in saturation. Thus, the saturation depends on the capillary pressure. As a consequence, we can establish a relationship between capillary pressure and saturation. There does not exist an exact analytic expression for this relationship. However, we can create a mapping or an algebraic function, $p_{cap}(s_w)$, based on data from experimental measurements. This function may depend on the saturation history. A typical curve representing measurement data relating CO_2 and brine is shown in Figure 2.1.

The four equations (2.4) - (2.7) form a complete model for two-phase, immiscible flow, provided that explicit parameterizations are given for p_{cap} and $k_{r\alpha}$, and proper initial and boundary conditions are given [45].

2.1.4 Vertical Integration

As mentioned in the introduction, the mathematical model describing CO_2 injection in permeable subsurface formations is well suited for a dimensional reduction. This is because the flow movement in the vertical direction is negligible in comparison to the horizontal one. It should be noted that when making this assumption, we rely on the fact that the complete gravity-driven segregation between the resident brine and the CO_2 occurs "quickly". This is reasonable as the buoyancy forces are strong [2]. For the time scales we are considering, this process can be regarded as instantaneous. Our system of governing equations can thus be reduced to two dimensions by vertical integration over the thickness of the formation. If we were to consider short-term or small-scale processes, the VE assumption might not be appropriate.

Modelling the Aquifer

The multiphase aquifer consisting of brine and CO_2 is trapped between two formations of very low permeability. The top cover of the formation is re-

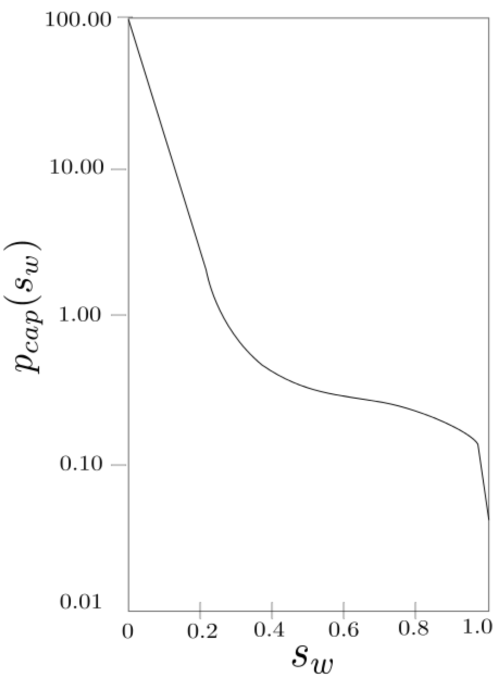


Figure 2.1: Primary drainage capillary pressure as a function of saturation, based on data measurements from Berea sandstone [24], where the saturation of the wetting fluid s_w has been normalized. The capillary pressure is a decreasing function of the wetting saturation.

ferred to as a *caprock*. There may be great variations in the topography of the caprock, but on a large scale we can consider it flat. We have a layered system of two main fluid zones visualized in Figure 2.2. This gravity-driven segregation is due to a high density contrast between the two fluids. The bottom region, consisting of the denser brine, stretches from the bottom, B, defined by $\zeta_B(x, y)$ to the interface I at $\zeta_I(x, y, t)$. In this region we have the saturations $s_b = 1$ and $s_c = 0$, where b and c indicate brine and CO_2, respectively. Above this interface we have a region with a mix of CO_2 and brine.

Capillary Fringe

In the top region, the saturation values will be varying with depth, with $0 < s_c < 1 - s_{b,res}$ and $s_{b,res} < s_b < 1$, where $s_{b,res}$ is the residual saturation of brine. When a fluid, such as CO_2, is injected into a formation of a denser resident fluid, such as brine, some of the brine will remain in the pore space

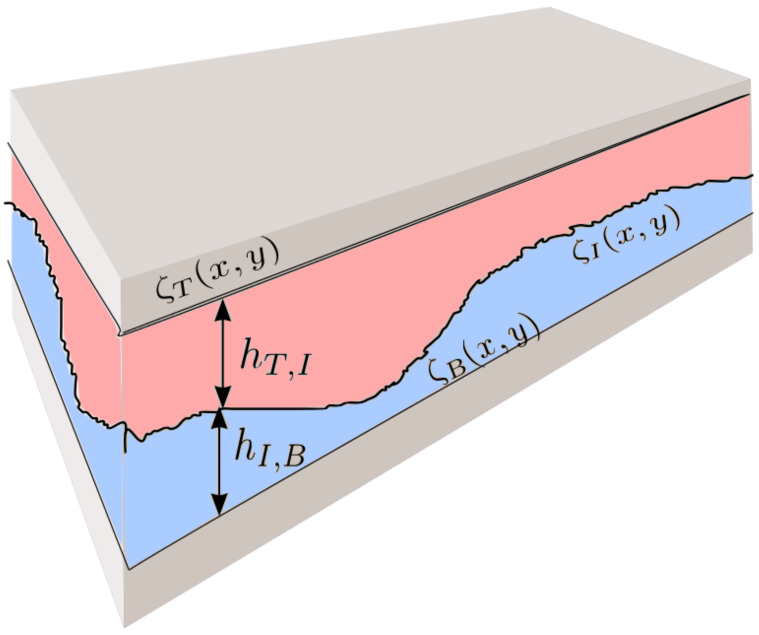


Figure 2.2: Diagram of an aquifer indicating the different interfaces used in vertical integration.

of the newly created CO_2 region. This brine can not be pushed out and is termed residual brine. The corresponding level of saturation is then $s_{b,res}$. One may also think of residual saturation as the fraction of the phase that cannot be recovered. We will use the term *capillary fringe* to describe the part of the mixed top region where the brine saturation ranges from full to residual, see Figure 2.3 (left).

Sharp-Interface Model

The capillary-fringe model may be simplified by making a sharp-interface approximation. The vertical distribution is then re-interpreted as two zones with constant saturation, separated by the interface at $h_{T,I}$. This model is illustrated in the right part of Figure 2.3. As we will see later, this simplifies the vertical integration procedure, resulting in a less computationally demanding simulator. In many cases this approximation will be sufficient, especially considering the limited knowledge we have about large-scale aquifers. In MRST there is a module called `MRST-co2lab`, which is a toolbox for mod-

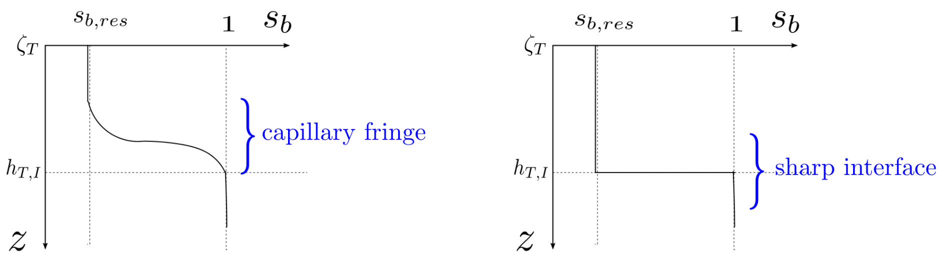


Figure 2.3: Diagram showing how the brine saturation s_b increases with depth z. The residual saturation $s_{b,res}$ is marked on the horizontal axis and the interface depth $h_{T,I}$ (see Figure 2.2) is marked on the vertical axis. In the left figure we have a capillary fringe model, while the right figure depicts a sharp-interface approximation. The capillary fringe represents the smooth transition phase between the two phases, discussed in the introduction.

elling and simulating trapping and isolation of CO_2 [1]. Inside this toolbox there are many simulators based on mathematical models with varying complexity. It includes both implicit and explicit solvers, compressible models and linear and non-linear approximations of the CO_2 properties. We will focus on one of the more basic implementations, namely a sharp-interface simulator with explicit time discretization. By making a comparison between the `MRST-co2lab` sharp-interface simulator and our capillary-fringe based model, we can evaluate the strengths and weaknesses of the two approaches.

Integration Procedure

Before proceeding with the vertical integration, we make some additional simplifications. We assume that fluids and solid are incompressible and decompose the gravitational force; $\mathbf{g} = \mathbf{g}_{\|} + g\mathbf{e_z}$ and $\nabla = \nabla_{\|} + \partial z\mathbf{e}_z$, where the $\|$ indicates the two-dimensional vector composed by the components x and y. The e_z component is pointing downwards, such that z is interpreted as depth. Integrating (2.5) from top to bottom gives

$$\frac{\partial}{\partial t}\left[\int_{\zeta_T}^{\zeta_B} s_\alpha \phi \,\mathrm{d}z\right] + \nabla_{\|}\cdot\left[\int_{\zeta_T}^{\zeta_B} \mathbf{u}_\alpha \,\mathrm{d}z\right] = \left[\int_{\zeta_T}^{\zeta_B} q_\alpha \,\mathrm{d}z\right]. \tag{2.8}$$

The vertically integrated terms in (2.8) are in the literature referred to as *coarse-scale* variables [44]. We take a variable defined in 3D, which is our so-called *fine* scale, and try to find a valid 2D representation, hereby eliminating the vertical component. One alternative is to compress it into some kind of averaged variable, which can by done by vertical integration. Another alternative is to find a reference depth and sample the variable here. Through these transformations we may lose some information, but the reduction in computational requirements is significant. More details about the limits of the vertical equilibrium model can be found in [14] .

From now on, the fine-scale variables will keep lower-casing, while the coarse or upscaled variables will be denoted by upper-case symbols. We start by looking at the integrated flow vector

$$\mathbf{U_c} = \int_{\zeta_T}^{\zeta_B} \mathbf{u_c}\,\mathrm{d}z = \int_{\zeta_T}^{\zeta_I} \mathbf{u_c}\,\mathrm{d}z,$$

where the last equality is due to the absence of CO_2 in the bottom region. We suppose that the pressure at the top of the formation is known, this will be our coarse pressure variable P_T. If we choose to neglect the capillary fringe for a moment, implying a sharp interface located at $h_{T,I}(x,y)$, the pressure distribution for our system in hydrostatic equilibrium is given by

$$p(x,y,z,t) = \begin{cases} P_T(x,y,t) + \rho_c g\left[z - \zeta_T(x,y,t)\right] & \zeta_T \le z \le \zeta_I, \\ P_T(x,y,t) + \rho_c g\left[\zeta_I(x,y) - \zeta_T(x,y)\right] + \rho_b g\left[z - \zeta_I(x,y)\right] & \zeta_I \le z \le \zeta_B. \end{cases}$$

Through this equation we derive the gradient of pressure for the top region:

$$\nabla_{\|} p_c(x,y,\zeta_T \le z \le \zeta_I, t) = \nabla_{\|} P_T - \rho_c g \nabla_{\|}\zeta_T.$$

Recalling the expression for $\mathbf{u}_\alpha$ given in (2.6) leads us to a new expression for the coarse velocity:

$$\mathbf{U_c} = \int_{\zeta_T}^{\zeta_I} \mathbf{u_c}\,\mathrm{d}z = -\left(\nabla_{\|} P_T - \rho_c g \nabla_{\|}\zeta_T - \rho_c \mathbf{g}_{\|}\right) \int_{\zeta_T}^{\zeta_I} \lambda_{\|}(s_c)\mathbf{k}_{\|}\,\mathrm{d}z$$

Similarly, we find the coarse velocity for brine to be

$$\mathbf{U_b} = \int_{\zeta_T}^{\zeta_B} \mathbf{u_b}\,\mathrm{d}z = -\left(\nabla_{\|} P_T - \rho_c g \nabla_{\|}\zeta_T + \Delta_\alpha \rho g \nabla_{\|}\zeta_I - \rho_b \mathbf{g}_{\|}\right) \int_{\zeta_T}^{\zeta_B} \lambda_{\|}(s_b)\mathbf{k}_{\|}\,\mathrm{d}z,$$

where $\Delta_\alpha\rho = \rho_b - \rho_c$. We define the coarse mobilities $\mathbf{\Lambda_c}$ and $\mathbf{\Lambda_b}$:

$$\begin{aligned}
\mathbf{\Lambda_c} &= \int_{\zeta_T}^{\zeta_I} \lambda_{\|}(s_c)\mathbf{k}_{\|}\,\mathrm{d}z\mathbf{K}^{-1}, \\
\mathbf{\Lambda_b} &= \left[\int_{\zeta_T}^{\zeta_I} \lambda_{\|}(s_b)\mathbf{k}_{\|}\,\mathrm{d}z + \int_{\zeta_I}^{\zeta_B} \lambda_{\|}(1)\mathbf{k}_{\|}\,\mathrm{d}z\right]\mathbf{K}^{-1}, \\
\mathbf{K} &= \int_{\zeta_T}^{\zeta_I} \mathbf{k}_{\|}\,\mathrm{d}z.
\end{aligned} \tag{2.9}$$

We wish to find an expression for $\mathbf{U_c}$ on the form

$$\mathbf{U_c} = -\left(\nabla_{\|}P_T - \rho_c g\nabla_{\|}\zeta_T - \rho_b\mathbf{g}_{\|}\right)\mathbf{\Lambda_c}(S_c)\mathbf{K},$$

where $\mathbf{\Lambda_c}$ will be a function of the coarse scale variable S_c defined as

$$S_c = \frac{1}{\Phi}\int_{\zeta_I}^{\zeta_T} \phi(z)s_c(z)\,\mathrm{d}z. \tag{2.10}$$

We recognize (2.10) as the first term in (2.8) divided by the coarse-scale porosity:

$$\Phi = \int_{\zeta_B}^{\zeta_T} \phi(z)\,\mathrm{d}z.$$

Fine-Scale Saturation

As stated in the preceding section, we want to reconstruct the vertical distribution $\mathbf{\Lambda}_c$ from the coarse variable S_c. From the definition made in (2.9), we identify s_c as the only unknown parameter since the expression for the mobility function $\lambda_{\|}$ is known. To find the relation between s_c and S_c, we need to the apply the function $p_{cap}(s_b) = p_{cap}(1 - s_c)$, relating capillary pressure and saturation, introduced in Section 2.1.3. By the definition of capillary pressure in (2.7) we have

$$p_{cap} = p_c - p_b,$$

for our aquifer. We now define z' as a new vertical component, representing depth below the top surface so that $z' = 0$ at $\zeta_T(x, y)$. If we let H denote the full height of the sample so that $H = \zeta_B - \zeta_T$, and denote the pressure at the bottom of the aquifer by p_B, we have

$$p_{cap} = p_c - p_b = P_T + g\rho_c z' - \left[p_B - g\rho_b(H - z')\right]. \tag{2.11}$$

A diagram illustrating this new coordinate system is depicted in Figure 2.4.

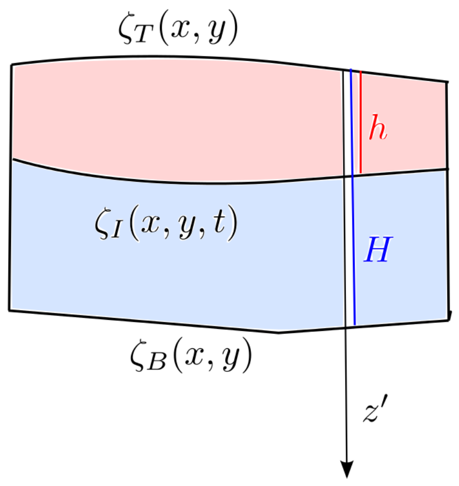


Figure 2.4: Diagram of the aquifer illustrating the new coordinate system.

We define h as the depth of the interface separating the two regions, i.e., $h = \zeta_I - \zeta_T$. Thus, h is the maximum depth with non-zero CO_2 saturation. The interface is known as the capillary interface, and here the pressure is equal to the entry pressure $p_{cap,i}$, introduced in Section 2.1.3. Writing $p_{cap,i}$ in terms of Equation (2.11), we get

$$p_{cap,i} = P_T - p_B + hg(\rho_c - \rho_b) + g\rho_b H.$$

Through this relation we may now express p_{cap} in terms of the known value $p_{cap,i}$. Thus, we arrive at an expression for fine-scale capillary pressure

$$p_{cap} = p_{cap,i} + g(\rho_c - \rho_b)(z' - h).$$

The capillary pressure curve, $p_{cap}(s_b)$, is a monotonically decreasing function of water saturation. Consequently, it has an inverse mapping or function known as the capillary-saturation function $s_b = s_{b,cap}(p_{cap})$. Thus, we now have an expression for the fine-scale saturation

$$1 - s_c(z') = s_b(z') = s_{b,cap}(p_{cap}) = s_{b,cap}\left(p_{cap,i} + g(\rho_c - \rho_b)(z' - h)\right),$$

in which the only unknown is the variable h. Hence we are able to express $\mathbf{\Lambda_c}$ as a function of h.

Relating h and S

We have still not answered our initial question of finding a relation between $\mathbf{\Lambda_c}$ and S_c. If we go back to the definition made in (2.10), we can make the following approximation

$$S_c = \frac{1}{\Phi}\int_{\zeta_T}^{\zeta_I} \phi(z)s_c(z)\,\mathrm{d}z \approx \frac{\phi}{\phi H}\left[\int_0^h s_c(z')\mathrm{d}z'\right] = f(h), \tag{2.12}$$

where we have assumed constant porosity, implying $\Phi = \phi H$. This means we have a direct relation between S_c and h. We can now choose to express $\mathbf{\Lambda_c}$ as a function of h or S_c. As a consequence, Equation (2.8) is left with only two unknowns, the top pressure P_T and S_c or h:

$$\Phi\frac{\partial S_c}{\partial t} + \nabla\cdot\mathbf{U_c}(S_c, P_T) = Q_c \quad \mathit{or} \quad \frac{\partial f(h)}{\partial t} + \nabla\cdot\mathbf{U_c}(h, P_T) = Q_c, \tag{2.13}$$

where $Q_c = \int_{\zeta_T}^{\zeta_I} q_c\,\mathrm{d}z$. Likewise, for brine we have

$$\Phi\frac{\partial S_b}{\partial t} + \nabla\cdot\mathbf{U_b}(S_b, P_T) = Q_b. \tag{2.14}$$

By our definition of the coarse saturation in (2.10), S_α will be bounded between 0 and 1 and sum to one, i.e.,

$$\sum_\alpha S_\alpha = 1. \tag{2.15}$$

Consequently, we may replace the saturation S_b in Equation 2.14 with $1-S_c$, and we are left with an upscaled system of two equations with two unknowns.

The fine-scale and coarse-scale relations have been resolved for our VE model. An overview of all the dependencies is depicted by the dependence tree in Figure 2.5. In the common sharp-interface approximation, the transition between fine-scale and coarse-scale becomes fairly simple. The coarse mobility integrals in (2.9) are simple to evaluate as the saturation-dependent fine-scale mobilities, $\lambda_\alpha(s)$, are constant and $\mathbf{k}$ is often approximated by an average. This is also the case for the relation between saturation and interface height, described by Equation (2.12). In contrast, our capillary-fringe approximation requires complete numerical integration of the fine-scale quantities, where all the fine-scale dependencies take part. A characterization of the different fine-scale functions will be given in Section 2.1.6.

2.1.5 Solving the Vertically Integrated System

There are many ways to solve the final system of vertically integrated two-phase flow equations derived in the previous section. We saw in Equation

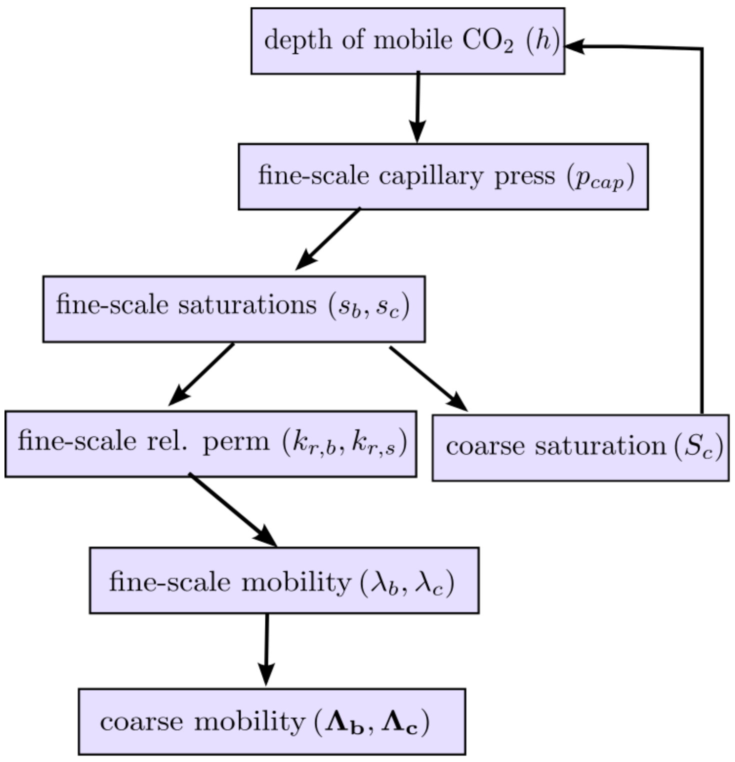

depth of mobile CO_2 (h)
fine-scale capillary press (p_{cap})
fine-scale saturations (s_b, s_c)
fine-scale rel. perm ($k_{r,b}, k_{r,s}$)
coarse saturation (S_c)
fine-scale mobility (λ_b, λ_c)
coarse mobility ($\mathbf{\Lambda_b}, \mathbf{\Lambda_c}$)

Figure 2.5: Dependence tree for the various coarse and fine-scale variables in our VE model.

(2.13) that we could choose between h and S as our primary variable. This is also true for other variables. Instead of the top pressure P_T, we could solve for the average pressure. See [44] for a discussion of different methods and simplifications for manipulating these two-phase flow equations.

We will choose S_c and P_T as our primary variables. For our governing equations, we start by transforming Equations (2.13) and (2.14) into an equation system consisting of one pressure equation and one transport equation. This is done by summing the equations, using (2.15) to eliminate the temporal derivative:

$$\nabla \cdot \mathbf{U_b}(S_b, P_T) + \nabla \cdot \mathbf{U_c}(S_b, P_T) = Q_b + Q_c.$$

If we now introduce the following new variables

$$\mathbf{U} = \mathbf{U_b} + \mathbf{U_c}, \quad Q = Q_b + Q_c, \quad \mathbf{\Lambda} = \mathbf{\Lambda_b} + \mathbf{\Lambda_c}, \quad \mathbf{F}_\alpha = \mathbf{\Lambda}_\alpha \mathbf{\Lambda}^{-1}, \tag{2.16}$$

we can rewrite our pressure equation as

$$\begin{aligned}\nabla\cdot\mathbf{U} &= Q\\ \mathbf{U} &= -\,\mathbf{K\Lambda}\big[\nabla_{\|}p_T-\mathbf{F_c}\rho_c g\nabla_{\|}\zeta_T-(\mathbf{F_c}\rho_c+\mathbf{F_b}\rho_b)\mathbf{g}_{\|}\\ &\qquad -\mathbf{F_b}\Delta_\alpha\rho g\nabla_{\|}h-\mathbf{F_b}\rho_b g\nabla_{\|}\zeta_T\big].\end{aligned}\tag{2.17}$$

$\mathbf{U_b}$ and $\mathbf{U_c}$ can be expressed in terms of $\mathbf{U}$ as follows

$$\begin{aligned}\mathbf{U_c} &= \mathbf{F_c}\big[\mathbf{U}+\mathbf{K\Lambda_b}\Delta_\alpha\rho\,\big[g(\nabla_{\|}h+\nabla_{\|}\zeta_T)+\mathbf{g}_{\|}\big]\,\big]\\ \mathbf{U_b} &= \mathbf{F_b}\big[\mathbf{U}-\mathbf{K\Lambda_c}\Delta_\alpha\rho\,\big[g(\nabla_{\|}h+\nabla_{\|}\zeta_T)+\mathbf{g}_{\|}\big]\,\big],\end{aligned}\tag{2.18}$$

which will prove useful when evaluating the fluxes in Section 2.2.3. Incorporating the total velocity $\mathbf{U}$ into (2.13), we get the transport equation:

$$\Phi\frac{\partial S_c}{\partial t}+\nabla\cdot\big[\mathbf{F_c}\mathbf{U}+\mathbf{K\Lambda_b F_c}\Delta_\alpha\rho\,\big[g(\nabla_{\|}h+\nabla_{\|}\zeta_T)+\mathbf{g}_{\|}\big]\,\big]=Q_c.\tag{2.19}$$

The final system of governing equations given by (2.17) and (2.19) is known as a fractional-flow formulation.

2.1.6 Fine-Scale Model Properties

Relative Permebility

For the fine-scale relative permeability function $k_{r,\alpha}(s_\alpha)$, we use a Corey-type approximation [29]. The Corey correlation is a power law in the wetting saturation s_b. If we define the effective saturation s_e as

$$s_e=\frac{s_b-s_{b,res}}{1-s_{b,res}},\tag{2.20}$$

the Corey correlations read

$$k_{r,c}(s_c)=\hat{k}_{r,c}(s_e)=k_c^e(1-s_e)^{N_c},\quad k_{r,b}(s_b)=\hat{k}_{r,b}(s_e)=k_b^e(s_e)^{N_b},\tag{2.21}$$

where k_b^e and k_c^e are end-point coefficients. The exponents N_c and N_B depend on the heterogeneity in the distribution of pore sizes. Large values indicate a relatively heterogeneous media, while small values correspond to homogeneous media. The Corey model has the favorable properties

$$\begin{aligned}&k_{r,c}(s_{c,res})=0,\quad k_{r,c}(1-s_{b,res})=k_c^e,\\ &k_{r,b}(s_{b,res})=0\quad\text{and}\quad k_{r,b}(1)=k_b^e,\end{aligned}$$

implying that the phases become immobile when they reach a level of residual saturation. A schematic of one set of Corey curves is given in Figure 2.6.

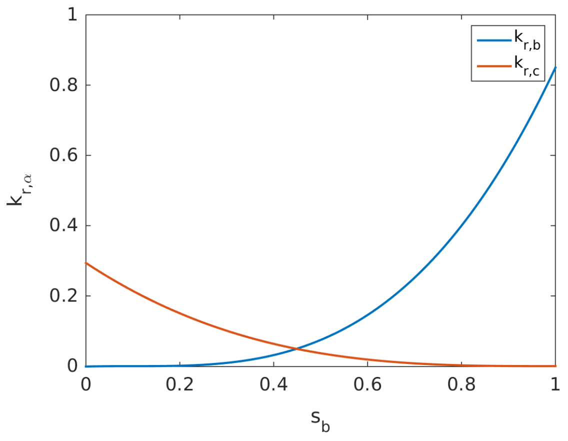


Figure 2.6: Example of a Corey approximation with end-point coefficients, $k_b^e = 0.85$, $k_c^e = 0.2142$ and $N_c = N_b = 3.$. The residual brine saturation $s_{b,res}$, is set to 0.1. The mobility of brine is increasing with the saturation and when we reach the maximum brine saturation at $s_b = 1$, the brine phase has reached its peak mobility. We find the same relationship between $s_c = 1 - s_b$ and $k_{r,c}$, only here the highest level of mobility is much lower because of the smaller value pf the end-point coefficient.

Relationship Between Saturation and Capillary Pressure

As mentioned earlier, the existing dependence between saturation and capillary pressure, $p_{cap}(s_w)$, is a part of our model. The shape of the $p_{cap}(s_w)$ curve will take a different form depending on whether we have a *drainage* or an *imbibition* process. Drainage is the displacement process in which an invading non-wetting fluid displaces a wetting fluid, whereas imbibition occurs when a wetting fluid displaces a non-wetting fluid. In our model, the injected non-wetting CO_2 displaces the wetting brine, corresponding to a drainage process. In 1966, Brooks and Corey [7] presented the capillary pressure function for the drainage case as follows,

$$p_{cap}(s_w) = C' s_e^{-1/\gamma}, \tag{2.22}$$

where s_e is the normalized saturation given in (2.20). The constant C' is a scaling parameter and γ is known as the pore size distribution index. We will use $\gamma = 2$. This approximation assumes that the initial saturation of CO_2 is equal to zero.

The Brooks-Corey model has been shown to hold for many different rock types and has been widely accepted in the petroleum industry. Theoretical development in later years has shown that this model, once considered as empirical, has a solid theoretical basis as well [34]. From (2.22), we derive the following inverse function

$$s_{w,cap}(p_{cap}) = \max\left[\left(\frac{C}{C+p_{cap}}\right)^2, s_{w,res}\right], \tag{2.23}$$

where C is on the form $\beta g \Delta_\alpha \rho H$, for $0 \leq \beta < 1$. Figure 2.7 shows the saturation as a function of depth z', with different values of β. We can see that for $\beta = 0$ the saturation distribution resembles a sharp-interface model.

Leverett-J Approximation

The Brooks and Corey approximation in (2.22) disregards the dependency between capillary pressure and permeability. We may include this effect by applying the dimensionless Leverett J-function:

$$J(s_w) = \frac{p_{cap}(s_w)\sqrt{\mathbf{K}/\phi}}{\sigma cos(\theta)}, \tag{2.24}$$

where θ is the contact angle and σ is the surface tension. The purpose of this function is to use known capillary-pressure data from one rock to approximate the data of comparable rocks with different permeability, porosity and wetting properties [33]. This is achieved through extrapolation. By rearranging (2.24) we get a new expression for the fine-scale capillary pressure

$$p_{cap}(s_w) = J(s_w)\sqrt{\frac{\phi}{\mathbf{K}}}\sigma cos(\theta) = C_p^{-1} J(s_w).$$

The inverse mapping between capillary pressure and saturation for the Levrett J approximation takes the form

$$\begin{aligned} s_{w,cap}(p_{cap}) &= J^{-1}(C_p\, p_{cap}) \\ J^{-1}(C_p\, p_{cap}) &= \max\left[\left(\frac{1}{1+C_p\, p_{cap}}\right)^2, s_{w,res}\right]. \end{aligned} \tag{2.25}$$

The purple curve in Figure 2.7 shows the saturation distribution in terms of a Leverett J approximation.

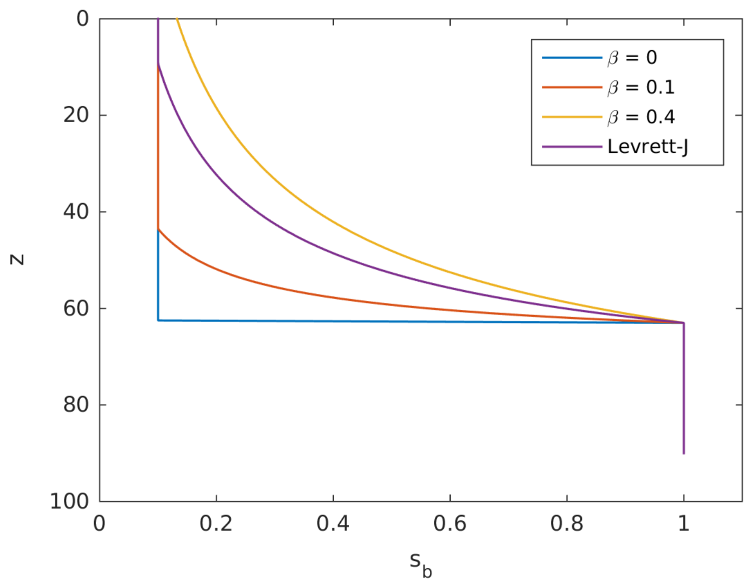


Figure 2.7: Examples of saturation distributions for $h = 0.6H$, $H = 85$. The saturation values have been computed using the inverse capillary pressure functions in (2.25) and (2.23). The scaling parameter in the latter equation, $C = \beta g \Delta_\alpha \rho H$, is varying in β. We can see that the height of the capillary fringe scales with β and when $\beta = 0$ we have a sharp-interface approximation. The parameters in Equation 2.25 have been set to $\sigma cos(\theta) = 30$ mPa, $\rho = 0.1$ and $\mathbf{K} = 100$ mD.

Hysteresis

Due to what is known as *hysteresis* or, in other terms, the effect of irreversibility, both the capillary pressure curve and the relative permeability curve will depend on the saturation history. The hysteresis effect takes place after the injection phase is completed. During injection, CO_2 migrates upwards, away from the injection wells, towards the top of the formation. This is caused by buoyancy forces. When the injection stops, CO_2 will continue migrating upwards and the leading edge of the CO_2 will continue to displace brine. Conversely, the trailing edge will be subject to an imbibition process, where brine is displacing CO_2, leaving a fraction of the CO_2 disconnected in the form of blobs or bubbles. This is referred to as the snap-off effect, and the CO_2 becomes effectively immobile or trapped [30]. As the plume migrates towards the top, a trail of residual, immobile CO_2 is left behind. An illustration of this effect is given in Figure 2.8. Thus, in the post-injection phase, the expression for fine-scale permeability (2.21) and fine-scale capillary pressure (2.22) must be reconstructed to incorporate this effect.

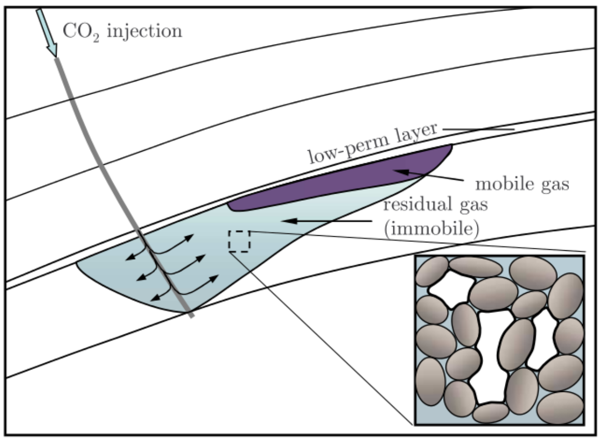


Figure 2.8: Schematic of the trail of residual CO_2 that is left behind because of snap-off as the plume migrates upward during the postinjection period. Figure taken from [30].

When making a sharp-interface approximation, the incorporation of hysteresis is quite simple. The original model of two different saturation zones only has to be extended to include three zones: maximum CO_2 concentration, residual CO_2 concentration and no CO_2. For capillary-fringe models it is more complex. The fine-scale capillary pressure functions have to be reconstructed frequently, as they will depend on the saturation history. Of course, the saturation history will vary across the formation, meaning that the reconstructions must be made locally. One way of handling this is to use tabulated values as described in [41]. The inclusion of hysteresis is outside the scope of this thesis, but it is an interesting topic for future research.

2.2 Numerical Method

We have established the primary variables and the corresponding equation set for our simulator in the previous sections. There is now the choice of method; whether to use a numerical or analytical approach, or a combination of these. For a numerical approach, which is our choice, there is the question of implicit versus explicit methods. The most common discretization practice in the field of reservoir simulation is the fully implicit method with phase-based upstream-mobility weighting with a two-point flux approximation [42]. However, as we aim at taking full advantage of the parallel nature of the GPU, an explicit scheme may be a better choice because they typically map better to the GPU architecture. We will thus turn to a sequential splitting method, which has been shown to produce satisfactory results for incompressible sharp-interface models [43]. This method is an extension of the implicit pressure, explicit saturation (IMPES) method, which has been very popular within research on CO_2 migration [42].

2.2.1 Sequential Splitting and IMPES

In the IMPES method, the parabolic pressure equation is solved by an implicit method, where we treat all the other variables in the equation explicitly, eliminating the nonlinearities. This means that the current pressure of the reservoir is computed using either the initial data or the results from the previous time step. The transport or saturation equation is then solved by an explicit solver, where the pressure from the previous step is considered a constant, rather than a variable. Due to its explicit nature, the IMPES method is conditionally stable and therefore requires relatively small time steps. In particular, if the rock permeability is very heterogeneous, the capillary pressure will affect the path of fluid flow substantially, putting very strong restrictions on the time steps for the transport solver [31].

Because of the physical character of our system, the implicit pressure computation is far more time consuming than the explicit saturation computation. Furthermore, the pressure changes less rapidly in time than the saturation in a two-phase flow system [10]. Based on these two properties, it seems reasonable to take larger time steps for the pressure equation, while keeping the smaller stability-preserving time step for the saturation equation. This reduces the computational cost, which is especially beneficial for problems with a large temporal scale, such as ours. This procedure is known as a sequential splitting method or an improved IMPES method, and is discussed in [10]. The time step restriction for the saturation equation will be adaptive and further details on this control strategy will be given in Section 2.2.4.

2.2.2 CPU and GPU Division

As already mentioned, the pressure equation requires an implicit method. Implicit methods are difficult to parallelize because the solution of one cell depends on the solution of all the other cells in the domain. Mathematically speaking this means solving a system of linear equations. Algorithms traditionally used for solving matrix systems are sequential in nature, and thus, not very suitable for the GPU. In recent years there has been a lot of effort focused on creating fast implicit solvers on the GPU. These approaches include application of the parallel cyclic reduction algorithm [53] and application of new generation CUDA tools such as CUBLAS [9]. However, this is outside the field of interest for this thesis, and we resort to MRST's routine for solving the pressure equation. Our focus will be on the GPU implementation of the explicit saturation computation. For the spatial discretization, we will apply a finite volume method on a structured Cartesian grid. The basis for this choice is the GPU hardware, which works very well with these types of structured grids.

2.2.3 Finite-Volume Method

For our system of partial differential equations in (2.17) and (2.19) we apply a finite-volume method. This means that we divide our domain Ω into smaller sub-domains or cells, hereby creating a meshed geometry. Instead of looking at point-wise values, as is customary in finite-difference methods, we look at cell averages. The average value $S_{i,j}(t)$ of $S(x,y,t)$ in the rectangle $x \in \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right]$, $y \in \left[y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}\right]$ is defined as

$$S_{i,j}(t) = \frac{1}{\Delta x \Delta y} \int_{y_{i-\frac{1}{2}}}^{y_{i+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} S(x,y,t)\,\mathrm{d}x\,\mathrm{d}y.$$

We rewrite (2.19) to fit the standard form of the transport equation

$$\Phi S_t + f(S)_x + g(S)_y = Q, \tag{2.26}$$

where f and g represent the fluxes in the x- and y-directions, respectively. We have cut the subscript c to simplify notation. Imposing the conservation law on integral form to (2.26), we get

$$\begin{aligned}
&\frac{d}{dt} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Phi(x,y) S(x,y,t)\,\mathrm{d}x\,\mathrm{d}y = \\
&- \int_{y_{j+\frac{1}{2}}}^{y_{j-\frac{1}{2}}} \left[f(S(x_{i+\frac{1}{2}}, y, t)) - f(S(x_{i-\frac{1}{2}}, y, t)) \right] \mathrm{d}y \\
&- \int_{x_{i+\frac{1}{2}}}^{x_{i-\frac{1}{2}}} \left[g(S(x, y_{j+\frac{1}{2}}, t)) - g(S(x, y_{j-\frac{1}{2}}, t)) \right] \mathrm{d}x \\
&+ \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} Q(x,y,t)\,\mathrm{d}x\,\mathrm{d}y
\end{aligned}$$

We define the numerical approximation of the fluxes over the cell edges

$$\begin{aligned}
F_{i\pm\frac{1}{2},j}(t) &\approx \frac{1}{\Delta y}\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f(S(x_{i\pm\frac{1}{2}},y,t))\,\mathrm{d}y \\
G_{i,j\pm\frac{1}{2}}(t) &\approx \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} g(S(x,y_{j\pm\frac{1}{2}},t))\,\mathrm{d}x,
\end{aligned}$$

and end up with the following expression

$$\begin{aligned}
\frac{d}{dt}S_{i,j}(t)V^p_{i,j} = &-\Delta y\left[F_{i+\frac{1}{2},j}(t)-F_{i-\frac{1}{2},j}(t)\right] \\
&-\Delta x\left[G_{i,j+\frac{1}{2}}(t)-G_{i,j-\frac{1}{2}}(t)\right]+(\Delta x\Delta y)Q_{i,j}(t). \qquad (2.27)
\end{aligned}$$

Here $V^p_{i,j}=\Phi_{i,j}\Delta x\Delta y$ is the pore volume of cell (i,j). To simplify notation we use the term $L_{i,j}(S(t))$ to represent the flux terms and $\tilde{Q}_{i,j}$ for the source term:

$$\begin{aligned}
L_{i,j}(S(t)) &= \Delta y\left[F_{i+\frac{1}{2},j}(t)-F_{i-\frac{1}{2},j}(t)\right]+\Delta x\left[G_{i,j+\frac{1}{2}}(t)-G_{i,j-\frac{1}{2}}(t)\right] \\
\tilde{Q}_{i,j} &= (\Delta x\Delta y)Q_{i,j}(t).
\end{aligned}$$

Discretizing (2.27) in time from t to $t+\Delta t$ by applying the first-order Euler method, results in the scheme

$$S^{n+1}_{i,j}=S^n_{i,j}-\frac{\Delta t}{V^p_{i,j}}\left[L_{i,j}(S^n)+\tilde{Q}_{i,j}\right], \qquad (2.28)$$

where $S^n_{i,j}$ represents the numerical approximation of $S_{i,j}(n\Delta t)$.

Upstream Mobility Weighting

Reservoir simulation schemes typically employ single-point upstream mobility weighting when approximating convective fluxes for multi-phase flow. In a single-point upstream method, a one-sided approximation is used for the derivative, where the direction of the local flow determines from which side we make the approximation. As an example, we consider the x-direction and look at cell (i,j). If the velocity in the x-direction is positive, meaning that the flow is going from left to right, we say that the left cell is the upstream cell.

For the numerical approximation of the fluxes in (2.19), we will apply the upstream principle to the coarse phase mobilities Λ_α. The direction of the flow of CO_2 and brine determines the upwind cell. In our system, the upstream direction of the two phases must be determined independently. As we can see from (2.18), the flow velocities depend on the mobility and the total velocity $\mathbf{U}$. If we assume that the total velocity $\mathbf{U}$ is known, the direction of the velocity of one phase depends solely on the mobility of the other phase. Therefore, if we know the upwind direction of one phase, we can determine the other. We look at the one-dimensional case and study the cells or columns in the x-direction, see Figure 2.9. We want to find the flux across the interface between cell (i, j) and cell $(i, j + 1)$. Defining

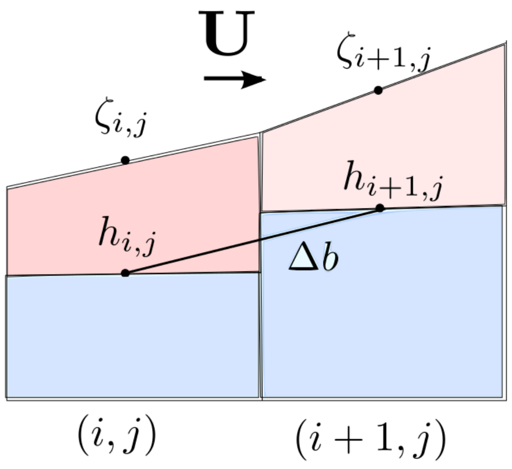


Figure 2.9: Diagram showing two adjacent columns in the vertical equilibrium model, where $\zeta_{i,j}$ is equal to the top surface ζ_T evaluated at the centroid of cell (i, j). The value of Δb helps determine the upstream cell for the flux evaluations.

$\Delta b = (\zeta_{i,j} + h_{i,j}) - (\zeta_{i+1,j} + h_{i+1,j})$, where $\zeta_{i,j}$ is the depth of the top surface ζ_T evaluated at the centroid of cell (i, j). Since the total velocity is known, we can find the direction of one phase by evaluating $\Delta b \cdot (\mathbf{U} \cdot \mathbf{x})$. In our example (Figure 2.9), the x-component of the velocity $\mathbf{U}$ is positive, meaning the total flow is going from left to right and $\Delta b > 0$. From the diagram it is clear that the CO_2 will move from left to right as it is the lighter fluid. A complete summary of the evaluation procedure is described in pseudocode in Algorithm 1.

Algorithm 1: Pseudocode for upwind flux evaluation

```
if $\Delta b \cdot \mathbf{U} \geq 0$ then
    if $\mathbf{U} \geq 0$ then
        $\mathbf{\Lambda}^c_{i+\frac{1}{2},j} = \mathbf{\Lambda}^c_{i,j}$
    else
        $\mathbf{\Lambda}^c_{i+\frac{1}{2},j} = \mathbf{\Lambda}^c_{i+1,j}$
    end
    if $\mathbf{U} + \mathbf{K}\mathbf{\Lambda}^c_{i+\frac{1}{2},j}\Delta_\alpha\rho\left[g(\nabla_\parallel h + \nabla_\parallel \zeta_T) + \mathbf{g}_\parallel\right] > 0$ then
        $\mathbf{\Lambda}^b_{i+\frac{1}{2},j} = \mathbf{\Lambda}^b_{i,j}$
    else
        $\mathbf{\Lambda}^b_{i+\frac{1}{2},j} = \mathbf{\Lambda}^b_{i+1,j}$
    end
else
    if $\mathbf{U} \geq 0$ then
        $\mathbf{\Lambda}^b_{i+\frac{1}{2},j} = \mathbf{\Lambda}^b_{i,j}$
    else
        $\mathbf{\Lambda}^b_{i+\frac{1}{2},j} = \mathbf{\Lambda}^b_{i+1,j}$
    end
    if $\mathbf{U} + \mathbf{K}\mathbf{\Lambda}^c_{i+\frac{1}{2},j}\Delta_\alpha\rho\left[g(\nabla_\parallel h + \nabla_\parallel \zeta_T) + \mathbf{g}_\parallel\right] > 0$ then
        $\mathbf{\Lambda}^c_{i+\frac{1}{2},j} = \mathbf{\Lambda}^c_{i,j}$
    else
        $\mathbf{\Lambda}^c_{i+\frac{1}{2},j} = \mathbf{\Lambda}^c_{i+1,j}$
    end
end
```

Source Term

The source term can be discretized as follows,

$$\tilde{Q}_{i,j} = \max(q, 0) + \mathbf{F_c} \min(q, 0),$$

where $\mathbf{F_c}$ is known as the fractional flow function which we recognize from (2.16). Hence if we have a source, i.e., q is positive, then the amount of CO_2 in the cell is increasing. If q is negative, we have a sink, and the amount of CO_2 that leaves the cell has to be scaled by the fractional flow.

2.2.4 CFL Condition

To guarantee stability in explicit time integration schemes, it is important that the numerical domain of dependence contains the physical domain of dependence of the PDE. This restriction is known as the Courant-Friedrichs-Levy (CFL) condition, and is applicable for any explicit finite volume or finite difference method. In general, this condition ensures that information propagates according to the physical speeds of the problem. These speeds are directly correlated with the fluxes. It must be stressed that the CFL condition only provides a necessary, and not sufficient, condition for stability.

Coats Time Stepping

Coats [13] derives a stability criterion for the IMPES method. The criterion is suitable for multiphase flow in multiple dimensions and can be applied to structured and unstructured grids. The time-step selector depends on gravitational, capillary and viscous forces and takes into consideration all possibilities of cocurrent and countercurrent flow configurations.

For the explicit part of IMPES, the conditional stability criteria for each cell (i,j) takes the form

$$\frac{R_{i,j}\Delta t}{V^p_{i,j}} < 1,$$

where Δt is the maximum stable time step and $R_{i,j}$ is a function composed by the reservoir and fluid properties. The global time step will be taken as the minimum stable time step of the cells.

For simplicity, we will study the one-dimensional case. In 1D, Equation (2.28) becomes

$$\frac{V^p_i}{\Delta t}(S^{n+1}_i - S^n_i) = -\left[F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}\right]. \tag{2.29}$$

Since we are using an upwind method to approximate the fluxes F, we may rewrite (2.29),

$$\frac{V^p_i}{\Delta t}(S^{n+1}_i - S^n_i) = -\left[F_{i+\frac{1}{2}}(S^n_{i+1}, S^n_i) - F_{i-\frac{1}{2}}(S^n_i, S^n_{i-1})\right],$$

emphasizing that the flux function depends only on the saturation values of the two adjacent cells. For stability analysis, a constant-coefficient, linear difference equation is needed. We introduce the error at step n as $\epsilon^n_i =$

$S_i^n - S_i^{*n}$, where S_i^{*n} represents the exact solution. The evolution of the error in time can then be expressed:

$$\frac{V_i^p}{\Delta t}(\epsilon_i^{n+1} - \epsilon_i^n) = \left[F_{i+\frac{1}{2}}(S_{i+1}^{*n}, S_i^{*n}) - F_{i+\frac{1}{2}}(S_{i+1}^n, S_i^n)\right] - \left[F_{i-\frac{1}{2}}(S_i^{*n}, S_{i-1}^{*n}) - F_{i-\frac{1}{2}}(S_i^n, S_{i-1}^n)\right]. \tag{2.30}$$

Recognizing that $F_{i+\frac{1}{2}}(S_{i+1}^{*n}, S_i^{*n}) = F_{i+\frac{1}{2}}(S_{i+1}^n - \epsilon_{i+1}^n, S_i^n - \epsilon_i^n)$, we can derive the first terms of two Taylor series:

$$F_{i+\frac{1}{2}}(S_{i+1}^n - \epsilon_{i+1}^n, S_i^n - \epsilon_i^n) = F_{i+\frac{1}{2}}(S_{i+1}^n, S_i^n) - \epsilon_{i+1}^n \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}} - \epsilon_i^n \frac{\partial F_{i+\frac{1}{2}}}{\partial S_i}$$

$$F_{i-\frac{1}{2}}(S_i^n - \epsilon_i^n, S_{i-1}^n - \epsilon_{i-1}^n) = F_{i-\frac{1}{2}}(S_i^n, S_{i-1}^n) - \epsilon_i^n \frac{\partial F_{i-\frac{1}{2}}}{\partial S_i} - \epsilon_{i-1}^n \frac{\partial F_{i-\frac{1}{2}}}{\partial S_{i-1}}.$$

Inserting these two expression into (2.30) gives

$$\begin{aligned}\frac{V_i^p}{\Delta t}(\epsilon_i^{n+1} - \epsilon_i^n) &= -\epsilon_{i+1}^n \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}} - \epsilon_i^n \left[\frac{\partial F_{i+\frac{1}{2}}}{\partial S_i} - \frac{\partial F_{i-\frac{1}{2}}}{\partial S_i}\right] + \epsilon_{i-1}^n \frac{\partial F_{i-\frac{1}{2}}}{\partial S_{i-1}} \\ &= a_i \epsilon_{i+1}^n - b_i \epsilon_i^n + c_i \epsilon_{i-1}^n.\end{aligned}$$

By assuming locally constant derivatives, that is,

$$\frac{\partial F_{i-\frac{1}{2}}}{\partial S_{i-1}} = \frac{\partial F_{i+\frac{1}{2}}}{\partial S_i}, \quad \frac{\partial F_{i-\frac{1}{2}}}{\partial S_i} = \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}},$$

we have $a_i + c_i = b_i$. We may also conclude that $a_i, b_i, c_i \geq 0$, seeing as $\frac{\partial F_{i+\frac{1}{2}}}{\partial S_i} \leq 0$, $\frac{\partial F_{i-\frac{1}{2}}}{\partial S_i} \geq 0$ for all cases of cocurrent and countercurrent flow [13]. Applying von Neumann stability analysis [38], results in the stability condition

$$\frac{\Delta t}{V_i^p}(a_i + b_i + c_i) \leq 2$$

$$a_i + c_i \leq b_i.$$

Expressing this criteria in terms of the partial derivatives we get

$$\frac{\Delta t}{V_i^p}\left[\frac{\partial F_{i+\frac{1}{2}}}{\partial S_i} - \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}}\right] \leq 1.$$

We recall that for our equation, the flux function F is given in (2.18). Thus, to calculate the term

$$\frac{\partial F_{i+\frac{1}{2}}}{\partial S_i} - \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}},$$

we must differentiate this equation. The partial derivative of F_i will be taken with respect to the upwind cell. Consequently, one may suspect that the final expression may differ for the four possibilities of cocurrent and countercurrent flow. However, the final result holds for all four cases:

$$\frac{\partial F_{i+\frac{1}{2}}}{\partial S_i} - \frac{\partial F_{i+\frac{1}{2}}}{\partial S_{i+1}} = \frac{\Lambda_c}{\Lambda_b \Lambda}|\mathbf{U}_{b,i}|\Lambda_b' + \frac{\Lambda_b}{\Lambda_c \Lambda}|\mathbf{U}_{c,i}|\Lambda_c' + \mathbf{K}\Delta\rho g \Lambda_b \frac{\Lambda_c}{\Lambda}\frac{2}{\Delta x}. \tag{2.31}$$

The same analysis can be applied for multidimensional flow, resulting in additional contributions to R_i on the same form as (2.31).

2.3 GPU Programming

The explicit numerical scheme derived in the previous section has a highly parallel character and is therefore well suited for a GPU implementation. Implementations of explicit schemes for conservation and balance laws, such as ours, have been found to benefit significantly from GPU acceleration [25, 39, 6, 8].

When the GPU was created, its main purpose was to improve the graphic experience in game consoles. Within this wealthy industry there was a great demand for powerful and inexpensive hardware that was highly parallel and could offer high data throughput. The GPU architecture was built upon these two properties to optimize graphics rendering. In the beginning, the functionality was limited to accelerating the memory-intensive work of texture mapping and rendering polygons. In the early 2000's, the programmable shader was introduced, giving the user more control and abilities to manipulate data. This trend continued, and as the GPU obtained more and more CPU-like capabilities, in particular in the field of matrix and vector operations, scientists and engineers gained interest [48]. These new capabilities made it possible to program algorithms for non-graphical applications on the GPU, as long as they were suitable for parallelization. Since then the GPU has been a key tool for accelerating scientific computations which require a high number of computations.

2.3.1 Solving the Saturation Equation on the GPU

As outlined in the introduction, we intend to use the GPU to accelarate our simulator. Two completed master projects have shown that sharp-interface

CO_2 simulators can be accelerated by means of a GPU [49, 19]. Thus, there is strong reason to believe that this is the case for our simulator as well.

The meshes used for aquifer modelling are large and each time step requires many computations per cell. This is especially true for the capillary-fringe approximation, which involves multiple numerical integrations. Thus, we have a very compute-intensive algorithm, which is ideal for GPU acceleration if it can implemented in parallel. The numerical method we have considered is based upon explicit temporal discretization. This means that each unknown can be computed independently. In particular, each cell can compute its coarse or vertically integrated values independent of its neighboring cells. Consequently, the cells can be solved in arbitrary order, making an ideal case for parallel computing. Since the solution is evolving in time, high throughput is desirable and thus, GPUs are very well suited for the job.

2.3.2 Understanding the NVIDIA GPU

For our GPU implementation we target NVIDIA GPUs with its programming platform named CUDA (Compute Unified Device Architecture) [46]. Starting with the basic terminology, the CPU and its memory are referred to as the *host* and the GPU and its memory constitute the *device*. When we make a program that is both serial (host) and parallel (device), we are dealing with what is referred to as *heterogeneous computing*. This term refers to a system that uses more than one kind of processor. A function running on the device is a *kernel*. The kernel executes the same code on a large batch of parallel threads, and each thread has its own ID so it can make control decisions and access the right addresses. The key to CUDA's efficiency is that it can launch thousands of threads at the same time and thereby hide latency. The platform is designed to give transparent scalability, meaning that code written for systems existing today should be equally compatible with the accelerated systems of the future. Whether CUDA lives up to this ideal in practice is debatable.

Device Architechture

A GPU consists of several streaming multiprocessors (SM), each of these contain a number of scalar cores or thread blocks. The individual thread blocks either execute identical instruction sets or "sleeps". The instructions are executed in groups of 32 threads called a *warp*, and there is a limit to the number of warps that can be active on each SM. This limit depends on the device properties.

Memory Hierarchy

To be able to set up, and especially optimize, a GPU code, it is essential to understand the CUDA memory model. The device memory has several layers illustrated in Figure 2.10. On the bottom level, we have the thread specific or local memory, which has the lifetime of the thread. On the next level, we have the on-chip shared memory which is shared among many threads. The group sharing this memory unit belong to the same thread block. This feature allows cooperation between threads on the same block, which is advantageous as the threads can share results, hence we can avoid computational redundancy. The block-level cooperation also gives us the ability to synchronize threads. On the top level, we have global or device memory which is accessible by all the threads. The shared memory is faster than the global memory by an order of magnitude, and has about 20-30 times lower latency and ten times higher bandwidth. This property is important to keep in mind when building a CUDA application.

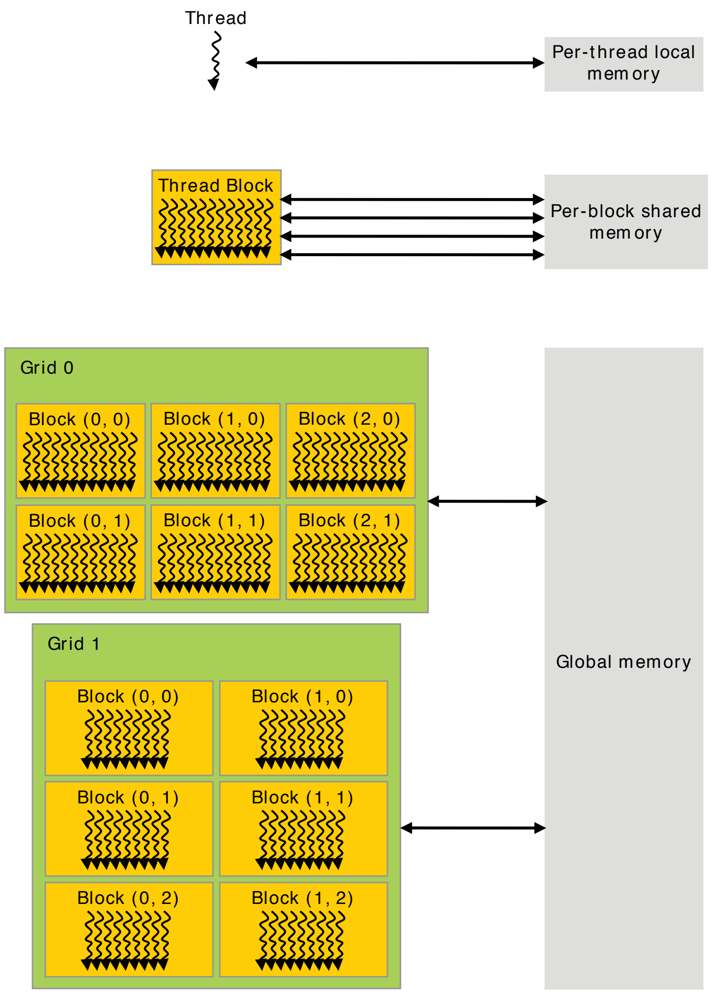


Figure 2.10: Illustration of the CUDA memory model taken from NVIDIA [46].

Blurred Image Example

For a better understanding of how to utilize shared memory and to give a pointer on how our code can benefit from it, we give a simple example. To create a simple blurring effect on a black and white image, each pixel is replaced by an average of the eight neighboring pixels. This is a very parallel process and thus a perfect task for the GPU, where each thread is assigned one pixel. A naive approach would be for each thread to import the neighboring pixel values from global memory into its local memory and compute the average. This means that each pixel value would be copied from global to local memory nine times, which clearly is not very efficient. If we instead make a domain decomposition where we divide the image into small square tiles that fit into shared memory, then each thread within the *tile* only has to copy a single pixel value from global memory. All the pixels on the edge of the tile will depend on pixels outside the tile, thus we add a *halo* of pixels of length one outside the tile, see illustration in Figure 2.11. The halos will be overlapping. Before the threads can start the average computation, the threads must be synchronized.

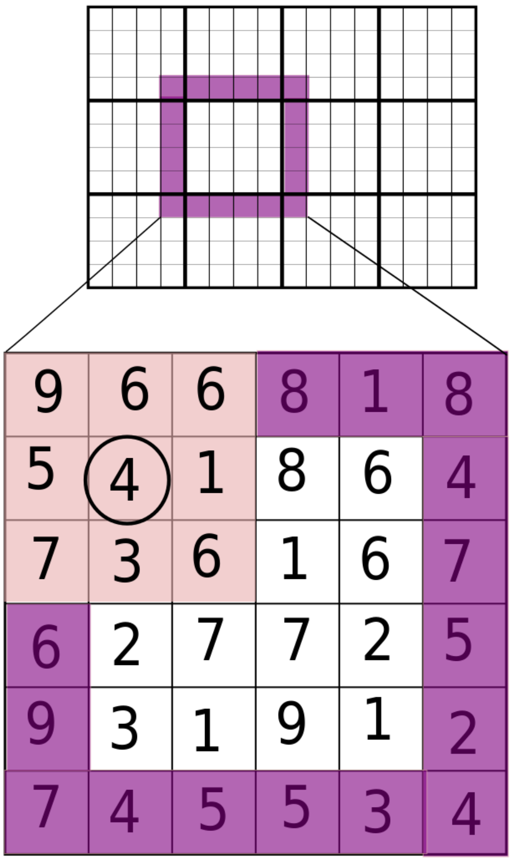


Figure 2.11: Domain decomposition for blurred image computation on the GPU. The image block in the shared memory is enlarged. The halo is marked in purple and the pixel values required for blurring the encircled pixel are shown in pink. The entire image is represented by the grid in the background.

2.3.3 Single versus Double Precision

From the beginning, the NVIDIA GPUs have emphasized single-precision arithmetics. While most of the newer GPUs support both single- and double-precision, the double-precision comes with a big performance reduction. The great speed tradeoff when using double-precision motivates the use of single-precision whenever it is found to be sufficient.

Chapter 3

A Fully-Integrated VE Simulator

We have established the mathematical framework and given an insight into GPU programming in the previous chapter, and we can now begin to understand the implementation. A presentation of the implementation will be the starting point of this chapter. We will also test the accuracy of our implementation by running the simulator on a problem for which there is an existing numerical solution. Next, we will approach the main question of this thesis, namely, to what degree do a sharp-interface and a capillary-fringe based simulation differ? Lastly, we will evaluate the performance of our code; how much are we benefitting from the GPU hardware? How large can the simulation domains be before we exhaust the GPU? Does our implementation have an optimization potential?

3.1 Implementation

Keeping in mind that the capillary-fringe model, with full numerical integration, was unknown territory in terms of implementation, our initial focus was simply to write a code that worked. Thus, some of the initial implementation choices concerning the numerical integrations might not be ideal. The improvements that were made later, at a more mature stage, will be elaborated in the optimization section.

3.1.1 Outline of the Algorithm

The fully-integrated VE simulator can be divided into two main parts, the `MRST-co2lab` pressure solver performed on the CPU and our saturation solver, which is mainly comprised by a set of CUDA kernels running on the GPU. Although the saturation solver is GPU based, it requires a C++ interface for data initialization, allocation and some transport which is run on the CPU. Figure 3.1 shows the program flow, where the steps performed on the CPU are marked in blue. As the yellow box indicates, the pressure is not updated on every iteration, in agreement with the sequential splitting method discussed in Section 2.2.1. The GPU sub steps, marked in purple, represent the four kernels that form the GPU program.

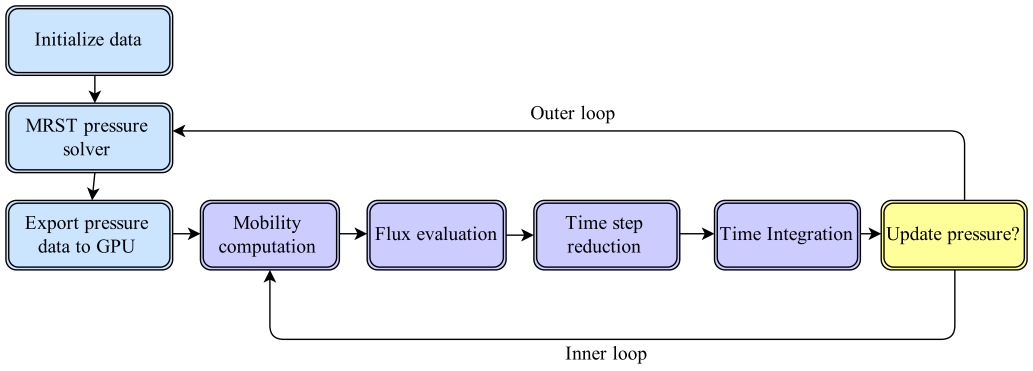


Figure 3.1: Flow chart showing the steps of the simulator. The CPU part of the implementation is marked with blue, while the part running on the GPU is marked with purple. Each purple box represents a kernel on the GPU.

3.1.2 The GPU Program

The GPU implementation is inspired by the shallow water simulator made by Brodtkorb et al. [6], which also aims at solving a PDE by implementing an explicit finite-volume scheme on the GPU. Although the shallow water simulator is more advanced due to a higher order scheme, it has many similar components. In particular, the implementation of the flux and time-step reduction kernels can be adapted to fit our scheme. On the GPU we will always work on a rectangular domain, regardless of the shape of the physical domain. This facilitates the domain decomposition required for the flux kernel, see Figure 3.3. The program is implemented in single-point precision to achieve higher performance. Because of the uncertainties in the underlying mathematical model and in the adaptation of the model to fit a numerical scheme, the use of single precision is justified. We will revisit this topic in Section 3.2.2, where the justification will become more clear.

Coarse Mobility Kernel

The first step of the saturation solver is the computation of the coarse mobilities $\mathbf{\Lambda_c}$ and $\mathbf{\Lambda_b}$ for each cell. These are found by approximating the definite integrals given in Equation (2.9). The numerical approximation is made by applying the trapezoidal rule, i.e.,

$$\int_0^h s_c(z')k(z')\,\mathrm{d}z \approx \frac{1}{2}\sum_{k=1}^{N}(z'_{k+1} - z'_k)\big[s_c(z'_{k+1})k(z'_{k+1}) + s_c(z'_k)k(z'_k)\big].$$

The geological 3D data sets that we apply, which include the permeability values k, have a very coarse resolution in the vertical direction. Typically there are only 5 or 6 cells. Since the mobility integral includes a saturation evaluation as well, we refine the cells in the initial 3D grid so that we can use shorter subintervals $(z_{k+1} - z_k)$, allowing us to capture the variations in saturation $s_c(z)$. We make the simple implementation choice of assigning one thread to each cell. For each cell, the value of h is different. Thus, we have two options for the choice of interval lengths in the trapezoidal method. We can either have an equal number of intervals N, or equal interval lengths $(z_{k+1} - z_k)$, for all cells. We choose the latter approach, preserving the numerical precision over the cells. However, this choice makes it difficult to get a good computational load balance on the GPU. The threads on the GPU work in a Single Instruction Multiple Data (SIMD) fashion, within the aforementioned warps of 32 threads. Consequently, if there is divergent branching within a warp, this will serialize the code. Thus, one should avoid different execution paths inside a warp. In this kernel, the number of evaluations per thread will be varying as N is varying, leaving many threads idle for long periods of time. An illustration of this problem is shown in Figure 3.2.

3.1.3 Flux Kernel

In this kernel the flux $\mathbf{U_c}$ is computed. Because of the upwind-scheme, the computation of the flux of one cell depends on the values of the adjacent cells. For example, for the east face of the cell (i, j), we need information about the cell $(i+1, j)$ to determine which cell is upwind. This should be exploited.

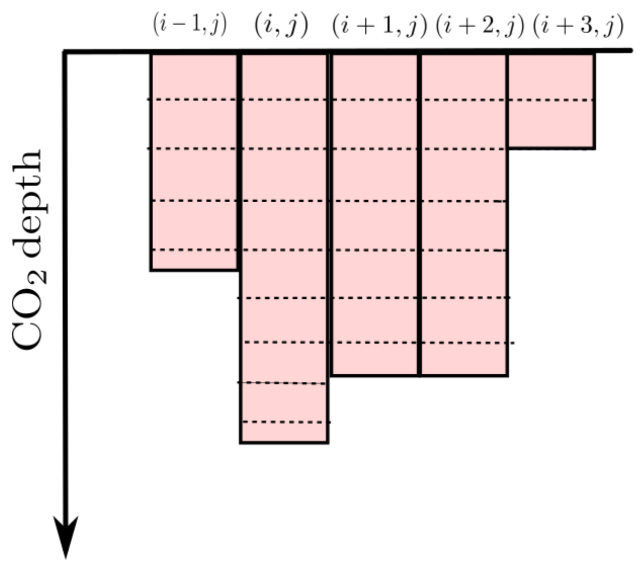


Figure 3.2: One-dimensional representation of the integration intervals for cells within a warp. As we can see, the number of intervals and thus, the number of computations for each thread, is varying. All threads within this warp have to wait for thread (i, j) to finish.

Along the same lines as the blurred image example given in Section 2.3.2, we make a domain decomposition of our global domain into smaller overlapping rectangular subdomains, where the pixels have been replaced with cells. A visualization is given in Figure 3.3. Since we have a stencil with a radius of one, each subdomain needs a local halo of one cell. Each subdomain is assigned to a group of threads belonging to the same block, where each thread is assigned one cell. For each cell, there are four fluxes to compute; north, south, east and west. However, since the cells share faces, we choose to only compute the northern and eastern fluxes, saving computational effort.

To better understand the implementation we divide our local grid into two layers: *block* and *tile*, see explanation in Figure 3.3. Each thread in the same block loads the data of its cell from the global memory and stores it in the shared memory, so that these values are available to all the threads in the block. Based on the data now residing in the shared memory, each thread in the inner region can determine the upwind cell. Once the upwind cell is found, the flux and the maximum time step for this cells' face are computed. By Coats [13], the time-step computation also depends on the upstream cell. We reuse the shared memory previously used to store the total flux $\mathbf{U}$ to minimize the shared memory use per block. After the fluxes have been computed, the next step is for each cell to perform the spatial discretization,

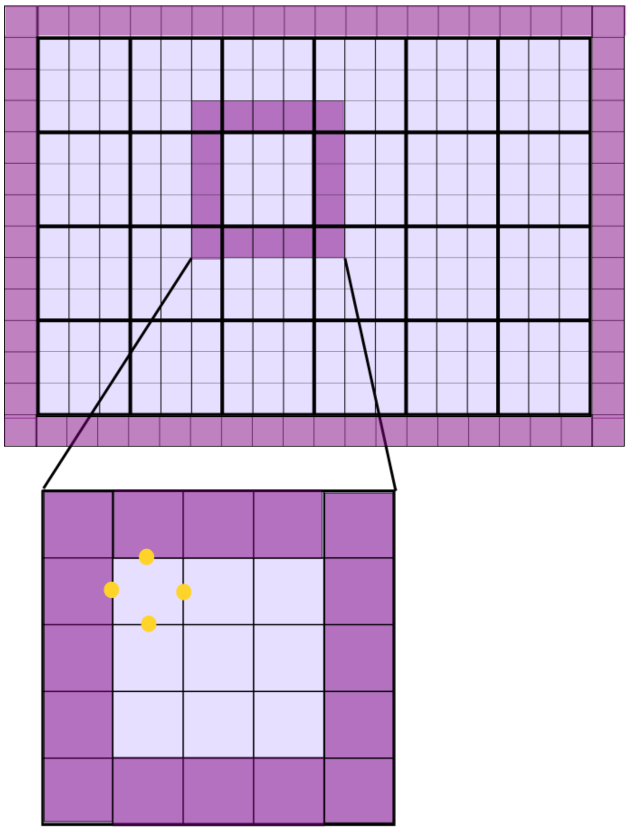


Figure 3.3: Global domain with halo and subdomain with local halo represented graphically. The yellow dots show the flux computations required for one cell. The subdomain is divided into two layers; the *tile*, which is the inner region, marked in light purple, and the *block*, which is the entire subdomain. All cells in the tile region are able to compute their fluxes based on the information of the cells in the block.

that is, compute the value $L_{i,j}$ for insertion into the time integration scheme (2.28). The final step of this kernel is a local time-step reduction. The maximum time step of each cell is collected in a shared memory block and we perform a local reduction. The resulting single time step value is then stored in an array residing in global memory.

3.1.4 Time-Step Reduction Kernel

This kernel performs a reduction on the global array consisting of the local time-step values collected from each of the thread blocks, as described in the previous section. The reduction algorithm outlined in the NVIDIA guide [27], was implemented with small modifications.

3.1.5 Time Integration Kernel

In this kernel we evolve the time step. The computed $L_{i,j}$ values and the coarse saturation values S_c^n of the preceding time step, are imported from global memory. We then perform the Euler time integration using the time step computed in the reduction kernel, to obtain the new coarse saturation, S_c^{n+1}. Before proceeding to the next iteration, we need to compute the new interface height h^{n+1}. We will consider three different algorithms for the computation of this quantity: a "brute force" method, Newton's method and the bisection method .

Brute Force

In Section 2.1.4, we saw that the coarse saturation and the interface height are related by the integration equation (2.12). Our unknown h is in the upper integration limit and also a part of the integrand $s_c(z)$,

$$s_c(z) = 1 - s_{b,cap}\left(p_{cap,i} + g(\rho_c - \rho_b)(z' - h)\right)\right).$$

To make the integrand independent of h, we make a change of variables $\tilde{z} = z' - h$, which gives the following integral

$$S_c = \frac{1}{H}\int_0^h 1 - s_{b,cap}\left(p_{cap,i} - g\tilde{z}(\rho_c - \rho_b)\right)\,\mathrm{d}\tilde{z}. \tag{3.1}$$

To compute h, we once again apply the trapezoidal rule. We keep integrating until the trapezoidal sum reaches the value of S_c^{n+1}, starting with coarse intervals and refining when we get close. The idea behind this procedure is described in pseudocode in Algorithm 2, where we have left out the refinement step, which is constructed in a similar manner. This kernel is also subject to a performance penalty caused by bad load balancing between the threads.

Algorithm 2: Pseudocode for computation of the new interface height h^{n+1}.

```
$sum = 0$
$\tilde{z} = 0$
$dz = interval\,length$
$previous\,s_c = computeSaturation(\tilde{z})$
while $sum < H \cdot S_c^{n+1}$ do
    $\tilde{z} = \tilde{z} + dz$
    $current\,s_c = computeSaturation(\tilde{z})$
    $sum = sum + 0.5 \cdot dz \cdot (current\,s_c + previous\,s_c)$
    $previous\,s_c = current\,s_c$
end
$h^{n+1} = \tilde{z}$
```

Newton's Method

Another alternative is to apply Newton's method, which is a derivative-based root-finding algorithm. We can reformulate our problem as a root-finding problem, $F(h) = 0$,

$$F(h) = S_c - \frac{1}{H}\int_0^h s_c(\tilde{z})\,\mathrm{d}\tilde{z}, \tag{3.2}$$
$$F'(h) = -\frac{s_c(h)}{H},$$

where the integral in $F(h)$ is evaluated using the trapezoidal rule. Newton's method is then defined as follows,

$$h_{i+1} = h_i - \frac{F(h)}{F'(h)}. \tag{3.3}$$

The convergence rate of Newton's method is highly dependent on the starting point h_0. Thus the initial guess must be chosen carefully. One strategy is to use the h from the previous time step h^n as the first guess, as for most cells the interface height h will not vary to much between consecutive time steps.

Bisection Method

The bisection or interval halving method is a reliable, but relatively slow, root-finding method for a function $F(h)$. It is the simplest and slowest member of the group of *bracketing* methods. The idea is that one starts with an interval $[a, b]$ for which $F(a)$ and $F(b)$ have opposite signs. By the intermediate value theorem, there must be at least one root between a and b. The bisection method then evaluates the function F at the midpoint $c = \frac{(a+b)}{2}$. Depending on the sign of $F(c)$, the search interval is cut in half. If $F(c)$ and $F(b)$ have opposite signs, the root belongs to the subinterval $[c, b]$, while if $F(c)$ and $F(a)$ have opposite signs, the root belongs to the subinterval $[a, c]$. This procedure is repeated until one reaches the desired accuracy. If the initial interval $[a, b]$ is big, the method requires many iterations to reach convergence as it only gains one "bit" of accuracy per iteration.

For the initial implementation, we choose the method outlined in Algorithm 2 to solve for the updated interface height h^{n+1}, but we will revisit both Newton's method and the bisection method in the optimization process described in Section 3.3.2.

When h exceeds the cell height H

If CO_2 starts filling the whole cell, the height of the capillary interface h will exceed the full height of the formation H, meaning that the capillary fringe is disappearing. Thus, the value h no longer has the same physical interpretation as before. When this happens, the numerical integrations performed in the coarse mobility kernel and in the time integration kernel need to be adjusted. For the former kernel, the adjustment is simply setting the upper integration limit to $\max(h, H)$, for the latter kernel it is more complex because of the change of variables made in (3.1). With the new variable $\tilde{z}$, the integration path starts with the capillary fringe, i.e., at the point where $s_c = 0$. For the brute-force approach, we need to shift the integral when h exceeds H, see Figure 3.4. Following the figure, we see that when we reach $\tilde{z} = H$, we need to cut off some of the initially integrated values shown in red, while continuing to integrate the blue part. This way, the "while" condition in Algorithm 2 is evaluated to be true for the correct $h^{n+1} = \tilde{z}$ value. For Newton's method and the bisection method, we keep the expression given in (3.2) for $h_i \leq H$, while for $h_i > H$ we get

$$F(h) = S_c - \frac{1}{H} \int_{h-H}^{h} s_c(\tilde{z})\, \mathrm{d}\tilde{z},$$
$$F'(h) = -\frac{1}{H} \left(s_c(h) - s_c(h - H) \right).$$

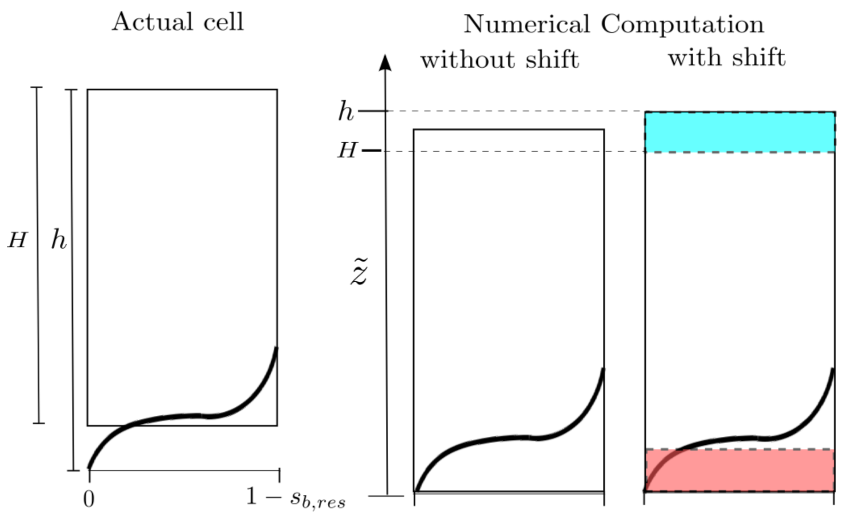


Figure 3.4: Illustration of the necessary shift so that Algorithm 2 finds the correct $\tilde{z} = h^{n+1}$ corresponding to the updated saturation ${S_c}^{n+1}$. The rectangles show the integration area. In the left figure we have the actual CO_2 distribution with the correct h. In the middle we see that when there is no shift, the numerically computed h values will be incorrect. On the far right we have the shift solution, where the red area is cut off and the blue area is added. The thick black line represents the capillary fringe, see Figure 2.3.

Cells Outside the Domain and Boundary Cells

Since we are bound to a rectangular domain, there will usually be many cells outside the actual domain, see the illustration in Figure 3.5. These cells will be marked by a zero height H. In the initialization step we identify all sub-domains in which none of the cells belong to the actual domain. By creating a grid mask, we make sure that no thread block is assigned to these dead sub-domains. We will use a no-flow boundary condition, which means that no fluid is allowed to leave the domain. This means that for all cell faces on the boundary, the fluxes are set to zero.

Data Movement and Communication Between the CPU and the GPU

The transfer speed or bandwidth between the GPU and the CPU is, by far, the slowest and is a potential bottleneck for the overall system performance. Therefore we wish to minimize this data transport as much as possible by keeping all intermediate data on the GPU. This means that parts of the code

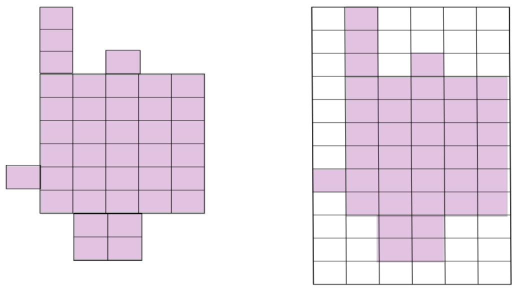


Figure 3.5: The real aquifer domain versus the rectangular GPU domain.

that exhibit low parallelism, and would run just as fast on the CPU, are run through kernels on the GPU to avoid data transfer. With this in mind, the geological reservoir data and the variables are first initialized on the host and then copied over to the device, where they remain throughout the entire computation. There are two repeated data movements between the host and the device through the course of a simulation: the time step, taking place at every iteration, and the interchange of the pressure p and the interface height h between the pressure and saturation solvers, taking place at every "outer" iteration, see Figure 3.1.

3.2 Numerical Results

For the main part of our results we use the Johansen formation as our test case. The Johansen formation is located offshore the south-west coast of Norway and has been considered a potential site for large-scale CO_2 storage. The reason for selecting this test case is that there exists a geological model based on available seismic and well data. This model has been developed as part of the MatMoRA project, and has been published online by SINTEF [50]. There have been several simulation projects revolved around this model, see for example [18, 20, 22].

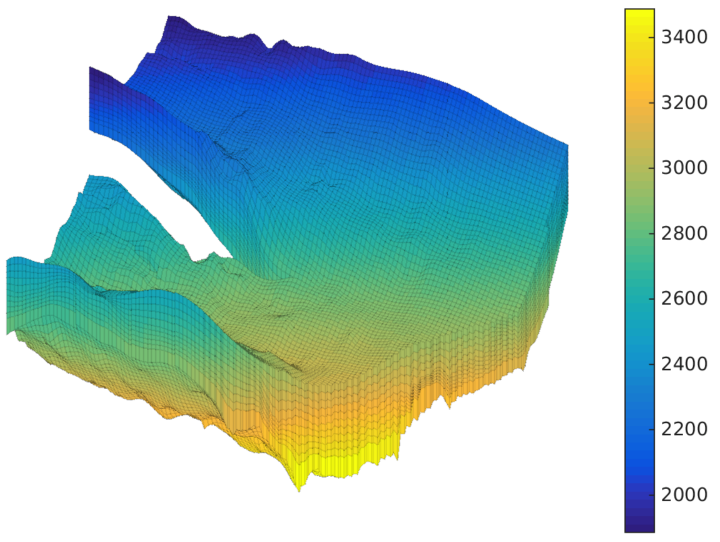
The top part of Figure 3.6 depicts the shape of the Johansen formation. We can observe that there is a large fault in the left part of the formation. This means that there will be no flow across the fault line, which must be taken into consideration in the implementation. The bottom part shows the permeabilty distribution. In Figure 3.7, the thickness of the formation is shown as a surface plot in 2D. Note the orientation of the formation in this plot. We will keep this orientation in all subsequent 2D plots. This plot also demonstrates the Cartesian grid which we are using, with 81×100 cells.

3.2.1 General Description of Simulation Set-Up

We will simulate the early migration phase of a CO_2 plume with a single injection point. The CO_2 injection well will be placed in the top layer of the formation, close to the fault, at the coordinate $(51, 51)$ indicated by a red line and a red square in Figures 3.6 (bottom) and 3.7, respectively. The injection period will last for $T_I = 100$ years, unless specified otherwise. The rate of injection of supercritical CO_2 is set to $1.4 \cdot 10^4\text{m}^3$ per day. The migration period, T_M, will be varying. We let the totalt time be denoted by $T_T = T_I + T_M$. The pressure is updated at even intervals by the MRST pressure solver. Values for some of the constant parameters are given in Table 3.1.

Table 3.1: Values for various constant parameters applied in the simulation.

μ_b	μ_c	ρ_b	ρ_c	$s_{b,res}$	$s_{c,res}$	k_b^e	k_c^e
(cP)	(cP)	(kg/m^3)	(kg/m^3)	-	-	-	-
0.3086	0.057	975.86	686.54	0.1	0.2	0.85	0.2142



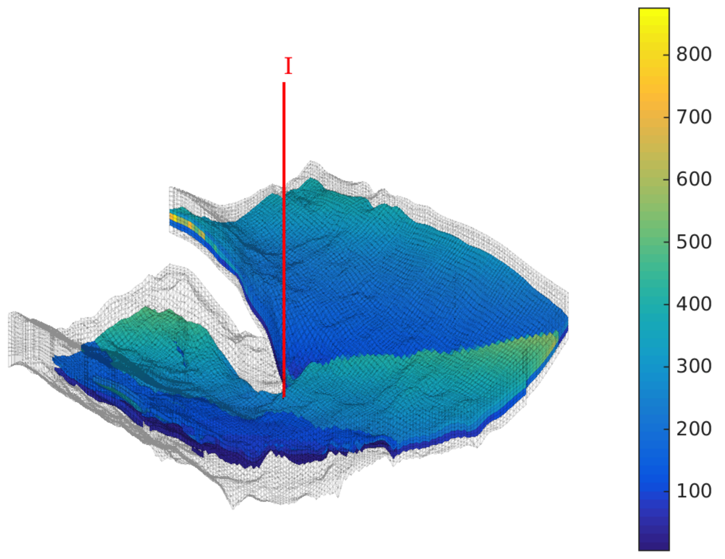


Figure 3.6: The Johansen formation. In the top figure, the shape of the formation is plotted. The values on the colorbar indicate the depth in meters. On the bottom, the permeability distribution is plotted in the unit millidarcy. The injection point is indicated by the red line.

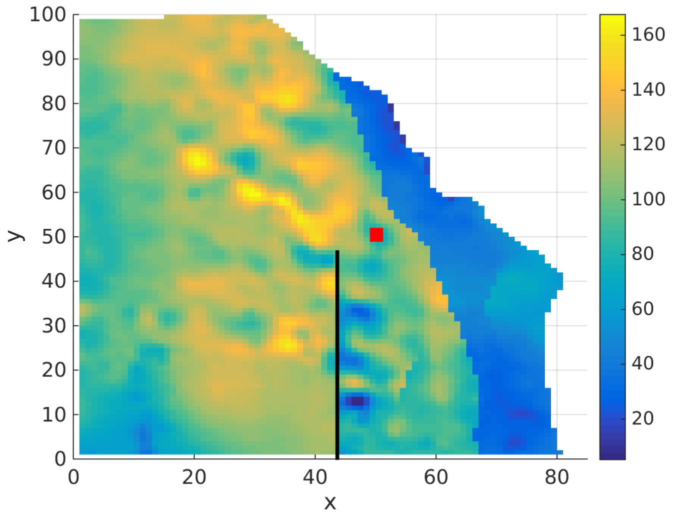


Figure 3.7: Illustration of the thickness distribution of the Johansen formation. The values on the colorbar show the thickness in meters. The fault in the reservoir is marked by the long black line, while the injection point is indicated by the red coordinate.

3.2.2 Verification of Simulator

To verify that the fully integrated GPU simulator is working correctly, we modify it so that it represents a sharp-interface model. We then compare the results with the `MRST-co2lab` sharp-interface simulator, where we have disabled the hysteresis. This particular simulator has been applied to the Johansen formation in an article by Ligaarden and Nilsen [35]. In this paper, a similar study is presented, where they make a comparison between a sharp-interface model and a full 3D model.

To convert the capillary-fringe model into a sharp-interface model, we make the following modifications

- Set the constant C in the expression for the relative permebility (2.23), equal to zero. We refer to Figure 2.7 illustrating that this conforms with a sharp-interface model.
- Set the permeability $k_{\|}$ along the z-direction to a constant value, $k_{\|}(z) = \frac{\mathbf{K}}{H}$, i.e., the average permeabilty.

To provide a solid verification, we let the MRST simulator run for $T_I = 100$ years with injection followed by two years of migration, $T_M = 2$ years. This assures that we have many cells filled with CO_2 at varying levels. The data is then used to initiate the GPU simulator and we let the two simulators run up to $T_T = 130$ years with identical time steps. The time-step computation is based on Coats [13] and is performed in MRST, where the time step has been scaled by a CFL coefficient of 0.5. The pressure is updated every two years.

From the plots in Figure 3.8 we see that before starting iterations on the GPU, the lower floating-point precision of the GPU solver is evident. In the right plot we have an absolute error $e \sim 10^{-1}$, and on the left we have plotted the volume of the CO_2 columns $\sim 10^6$. Thus, the relative error is of order $e_r \sim 10^{-7}$, which is reasonable as the MRST solver is using double precision, while the GPU is limited to single precision. In Figure 3.9 we see that the error increases slowly with time, and after 43 time steps the relative error is around $e_r \sim 10^{-6}$, which is still acceptable.

Time Step Sensitivity

When running the two simulators with the sharp-interface approximation from time $T_T = 0$ to $T_T = 100$, i.e., the full injection period, the relative

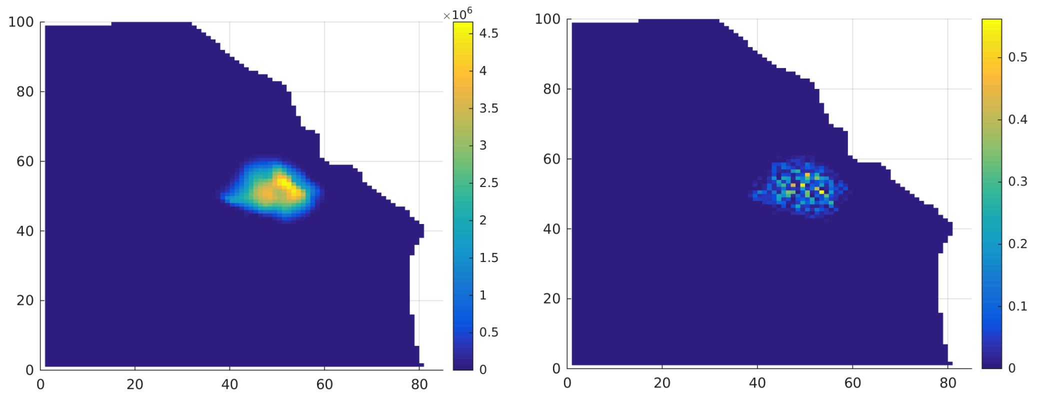


Figure 3.8: Surface plot showing the volume of the CO_2 columns at $T_T = 102$ years on the left. On the right, absolute difference between the volume of CO_2 in the MRST solver and the GPU solver at $T_T = 102$ years, before any iterations on the GPU.

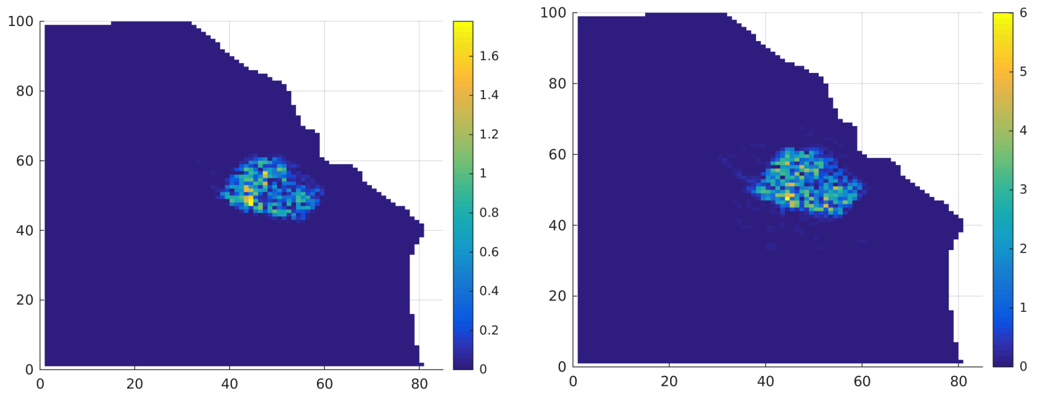


Figure 3.9: Absolute difference between the volume of CO_2 in the MRST solver and the GPU solver. On the left, after three timesteps, $T_T = 104$, and on the right, after 43 time steps, $T_T = 130$.

error turned out to be quite large. In this test the time steps were computed independently by Coats formula in the two simulators. In the top left plot in Figure 3.10 we see that the absolute error or discrepancy is $e \sim 400$ in some cells. In these cells the volume of $CO_2 \sim 10^5$, thus we have a relative error (discrepancy) $e_r \sim 10^{-3}$, which is quite large. The error distribution also has a special geometric shape, as opposed to the "salt-and-pepper" distribution in the preceding plots. On the right, we have plotted the real error. We see that there is an inner circle around the injection well where the GPU simulator overestimates and an outer circle where it underestimates. After further investigation, we find that this tendency starts at $T_T = 60$ years, after 96 time steps, see the bottom plot in Figure 3.10. This coincides with the point at which the size of the time steps start differing. Before we reach $T_T = 58$ years, the relative difference between the time steps in the two simulators is $\sim 10^{-6}$. After this point the GPU simulator computes larger time steps, which differ from the MRST time steps by an order of $e_r \sim 10^{-3}$.

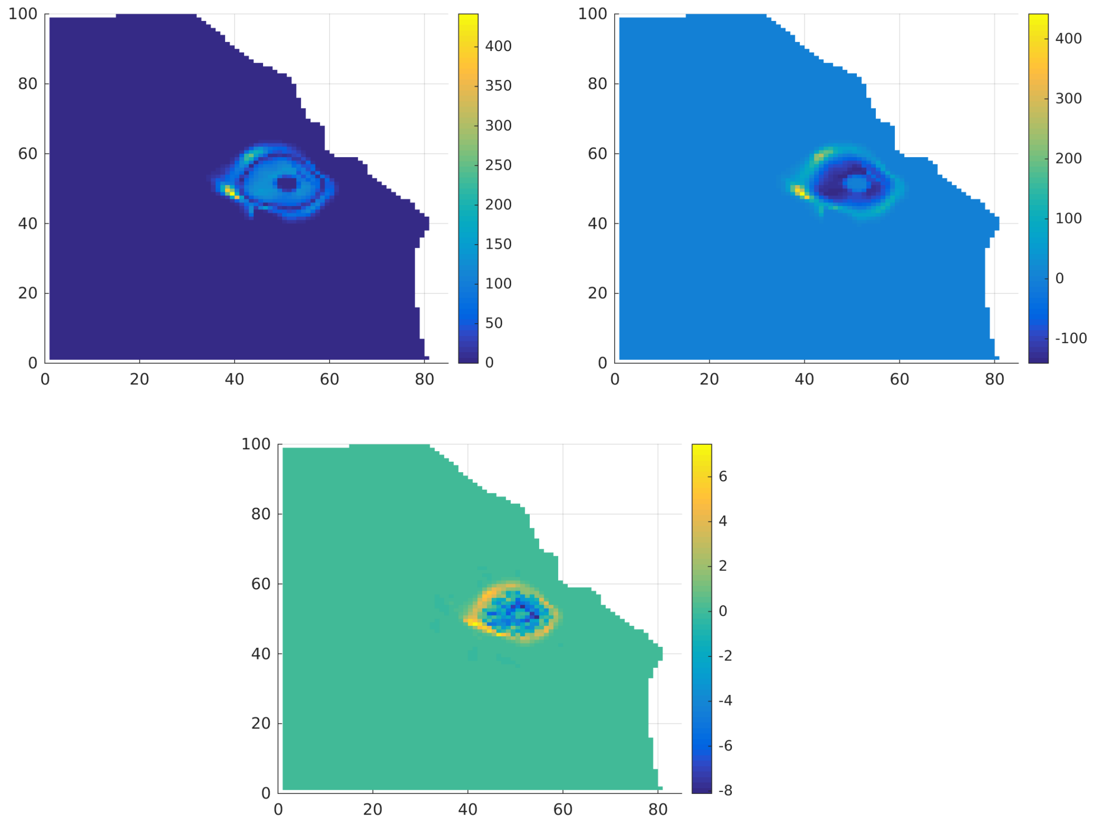


Figure 3.10: On the top, absolute difference (left) and real difference (right) between the volume of CO_2 in the MRST solver and the GPU solver after $T_T = 100$ years. On the bottom an equivalent plot at $T_T = 60$ years.

Even though the total integration time T_T is the same, the size of the time step seems to have an impact on the result. We try to confirm this hypothesis in two ways. First, we run the same simulation with identical time steps, resulting in an error of $\sim 10^{-6}$. The results are shown in Figure 3.11. Second, we run the MRST simulator with varying CFL-coefficient and investigate the effect. Referring to Figure 3.12 (left), where we have visualized the result of adjusting the CFL-coefficient from the initial 0.50 to 0.49, we observe the exact same geometrical pattern as in Figure 3.10. We have also plotted the maximum relative error as a function of the CFL coefficient in 3.12 (right). Clearly the simulator is sensitive to variations in the time step size. An important question then is "why do the two simulators compute different time steps in the first place?" At the critical point, $T_T = 58$ years, MRST finds the minimum time step in a cell with $\mathbf{\Lambda}_c = 10^{-15}$, that is, a cell with a very small amount of CO_2. Because of the reduced precision, no CO_2 has reached this cell in the GPU simulator and by Coats formula [13] it will not contribute to the time step. Hence, the GPU simulator will find a minimum time step in a different cell, with a value that is larger than this.

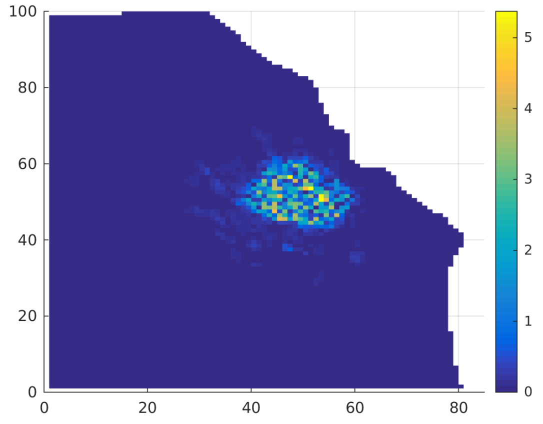


Figure 3.11: Absolute difference between the volume of CO_2 in the MRST solver and the GPU solver starting from $T_T = 0$ to $T_T = 100$, run with identical timesteps.

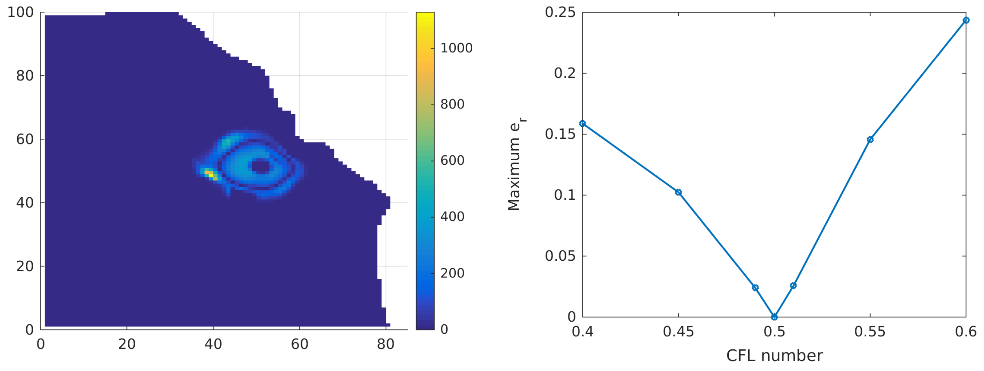


Figure 3.12: On the left, absolute difference between the volume of CO_2 when MRST is run with two different CFL coefficients, 0.50 and 0.49. The total simulation time is $T_T = 100$. On the right, the maximum relative error is plotted for varying CFL numbers for the same simulation set-up. The two plots confirm that variations in the time step has an impact on the simulation results.

Comments on the Verification Procedure

The verification made above gives a good indication that the numerical scheme is implemented correctly on the GPU, which is not a given, and also demonstrates the impact of using single-precision. However, it does not provide a solid verification with respect to the accuracy of the problem the simulator is intended to solve. As an example, consider the accuracy of the numerical integrations. When we evaluate the coarse mobilities and solve for the updated h by performing numerical integration on a sharp-interface model, the achieved accuracy will be independent on the the number of subintervals or evaluation points. The only thing that matters, in terms of accuracy, is that the last subinterval has the right length Δz, see Figure 3.13. Therefore, increasing or decreasing the number of evaluations or applying more accurate numerical quadratures will have no effect. Thus, to evaluate the accuracy of the numerical integration procedures, a different type of comparison must be made. We will leave this analysis for future work.

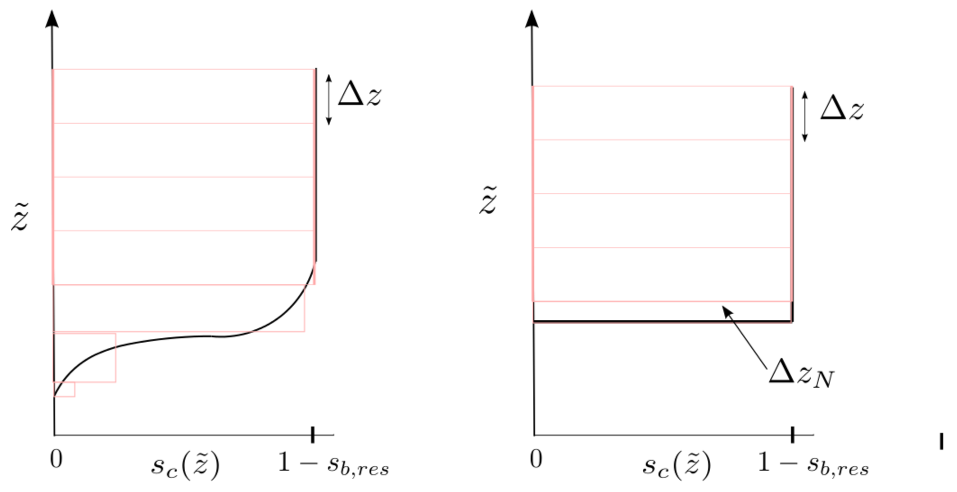


Figure 3.13: Illustration of the trapezoidal rule for integral approximation applied to a capillary-fringe model (left) and a sharp-interface model (right). For the capillary-fringe model, the size of Δz affects the numerical accuracy, while for the sharp-interface model, the only thing that will affect the accuracy is the length of the last interval Δz_N.

3.2.3 Sharp Interface and Capillary Fringe Comparison

Now that we have verified our implementation, we can start investigating the differences between a sharp-interface and fully-integrated or capillary-fringe model. From now on all sharp-interface simulations will be run by `MRST-co2lab`, while the capillary-fringe simulations will be performed by our GPU simulator.

In our tests we let the migration period last for $T_M = 400$ years. For the presentation of the results we have chosen to plot the saturation distribution of CO_2, which lies in the range $[0, 1 - s_{b,res}] = [0, 0.9]$, as this range is more intuitive and gives a better indication of the spread. In Figure 3.14 we see the result of running this simulation with a sharp-interface model with constant or averaged permeability. The black stapled box indicates the area of the domain which we will focus on in the comparisons. We will run many different versions of the capillary-fringe model where we vary different parameters, such as the permeabilty and the fine-scale functions, to highlight the strengths and weaknesses of the sharp-interface model.

The tests will be divided into two main groups. In the first one, the mobility approximation is a linear function of saturation, while in the second group it is cubic. The different test scenarios are given in Tables 3.2 and 3.3.

Capillary Fringe with Linear Mobility

For our linear mobility simulations, we use the Corey relative permeability approximation introduced in Section 2.1.6, where the exponents N_c and N_b are set to 1, such that $\lambda_c \propto s_c$. The different simulation runs with linear mobility are summarized in Table 3.2 and the corresponding plots are given in Figures 3.15 and 3.19.

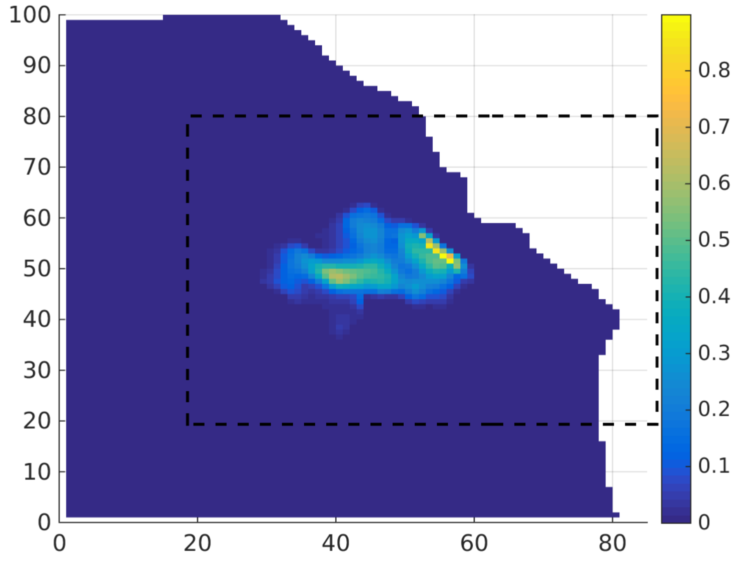


Figure 3.14: CO_2 saturation distribution for the sharp-interface simulator run for $T_T = 500$ years.

Table 3.2: An overview of the different simulator test cases run with linear mobility.

Simulation	Permeability	Fine-scale Capillary Pressure	
		Model	Description
1	constant	Brooks-Corey	Eq. (2.23) with $C = 0.1g\Delta_\alpha\rho H$
2	constant	Brooks-Corey	Eq. (2.23) with $C = 0.4g\Delta_\alpha\rho H$
3	constant	Leverett J	Eq. (2.25) with $\sigma\cos(\theta) = 50\text{mPa}$
4	varying	Brooks-Corey	Eq. (2.23) with $C = 0.1g\Delta_\alpha\rho H$
5	varying	Brooks-Corey	Eq. (2.23) with $C = 0.4g\Delta_\alpha\rho H$

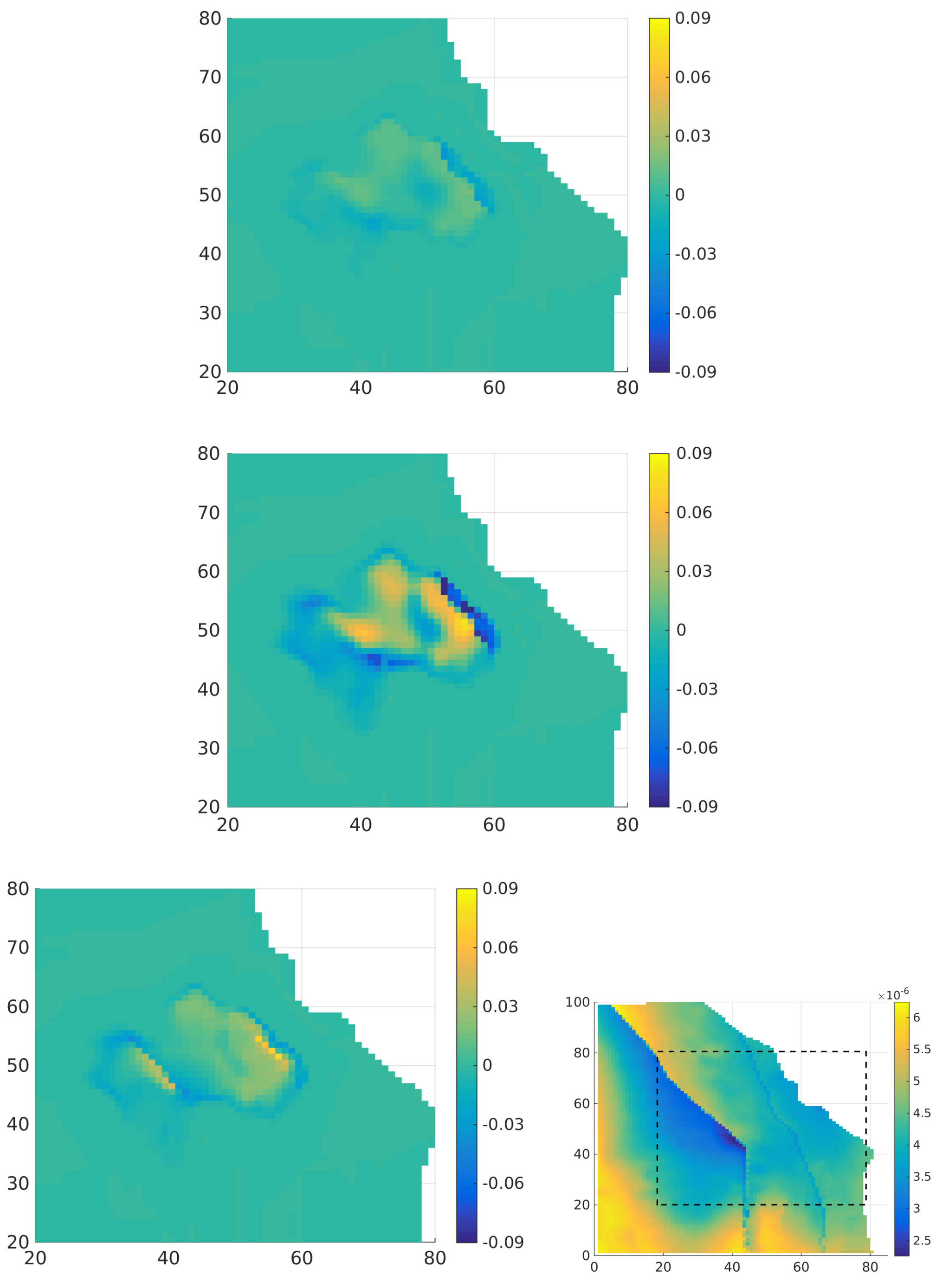


Figure 3.15: Sharp-interface simulator versus the capillary-fringe simulator with linear mobility, when running for $T_T = 500$ years. The plots show the difference in CO_2 saturation between the two simulators, i.e., $S_c^{\mathrm{SI}} - S_c^{\mathrm{CF}}$. From top to bottom we have simulations 1-3 whose descriptions are given in Table 3.2. The right plot on the bottom shows the distribution of $\frac{\mathbf{K}}{\phi}$.

Capillary Fringe Effect For the simulations run with a constant or averaged permeability, we see that the capillary fringe has an impact. The results are shown in Figure 3.15, where the difference between the sharp-interface (SI) model and the capillary-fringe (CF) model is plotted, i.e., $S_c^{\mathrm{SI}} - S_c^{\mathrm{CF}}$. In the two top plots we see that the effect of the capillary fringe increases with the parameter C. This is expected as C is proportional to the length of the capillary fringe. In the bottom plot, the Leverett J approximation (left) is plotted alongside a surface plot of $\frac{\mathbf{K}}{\phi}$ (right). The areas where the variation in $\frac{\mathbf{K}}{\phi}$ is strong, such as the dark blue area, are reflected in the saturation distribution. Thus, the Leverett J relative permeability approximation captures the rock properties of the formation, as intended. Also, if we look at the magnitude of the discrepancy, our capillary-fringe hypothesis fits well with the Levrett-J result. In Figure 2.7 we see that the Levrett-J capillary fringe lies right between the other two.

With regards to the shape of the plots, the capillary fringe model seems to spread the CO_2 more out. The difference plots in 3.15 have a negative blue contour indicating that in the periphery there is a higher level of S_c^{CF}. How can we explain this difference in the migration pattern? If we study two identical cells with equal coarse saturation S_c, the height h will differ between the two models. This is depicted by the curves in Figure 3.16. The interface height h of the capillary fringe model will always be higher, meaning that the CO_2 in this cell will be mobile at lower depths. Depending on the topography of the formation, this may affect the results. To better understand the influence of the topography, we recall Figure 2.9 from the theory on upstream mobility weighting in Chapter 2. The flow between two adjacent cells was dependent on the length $\Delta b = (\zeta_{i,j} + h_{i,j}) - (\zeta_{i+1,j} + h_{i+1,j})$, where $\zeta_{i,j}$ was the depth of the top surface ζ_T evaluated at the centroid of cell (i, j). If the depth $\zeta_{i,j} + h_{i,j}$ in a CO_2-filled cell is lower than the depth of the top surface in the adjacent cell, the CO_2 will not flow there.

To demonstrate that this effect is present in our case, we study a 1D snapshot of simulation 2 at an early stage, see Figure 3.17 (top). Because of the "parabolic" shape of the top surface caused by a fold in the formation, the sharp-interface CO_2 becomes trapped, limited by its short h-value. This type of geological structure is referred to as a *structural* trap. In Figure 3.17 (bottom) we show an extension of the same 1D intersection at a later point in time, where the same situation is occurring on the left. We also observe that on the right side, the capillary-fringe model has migrated further as a result of the structural trapping observed in the first snapshot. Thus, every time the CO_2 comes across this type of top-surface curvature, the sharp-interface model will decelerate migration. In conclusion, for simulation scenarios with undulating topographies, in which the mobility is estimated to be linear, the sharp-interface model will underestimate the CO_2 migration radius.

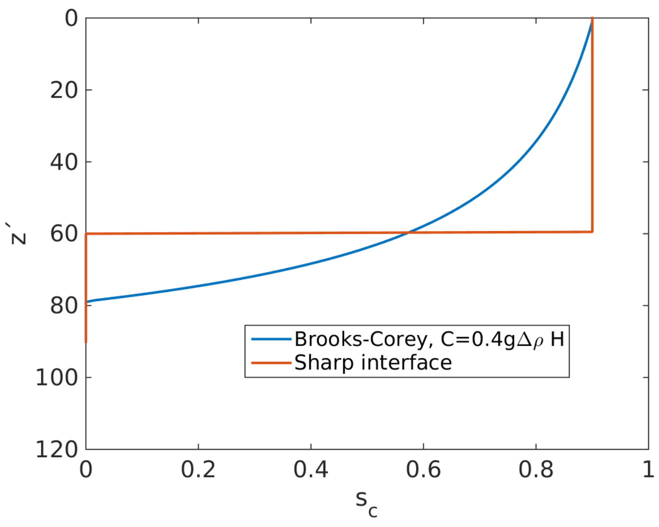
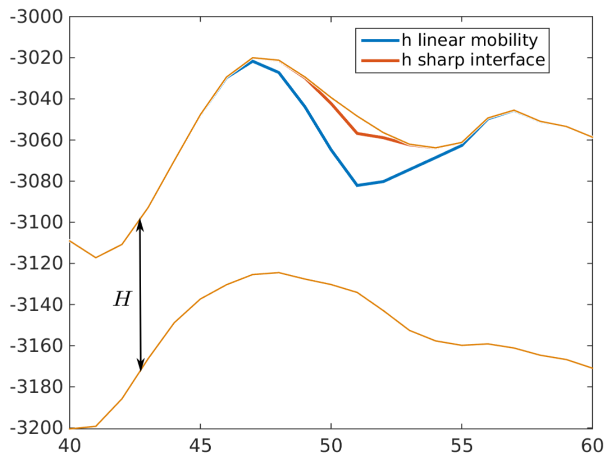


Figure 3.16: Saturation distribution for the capillary-fringe model and the sharp-interface model with equal coarse saturation, $S_c^{CF} = S_c^{SI} = 0.6$. For the former model, the height $h \approx 80$, while for the latter $h \approx 60$.



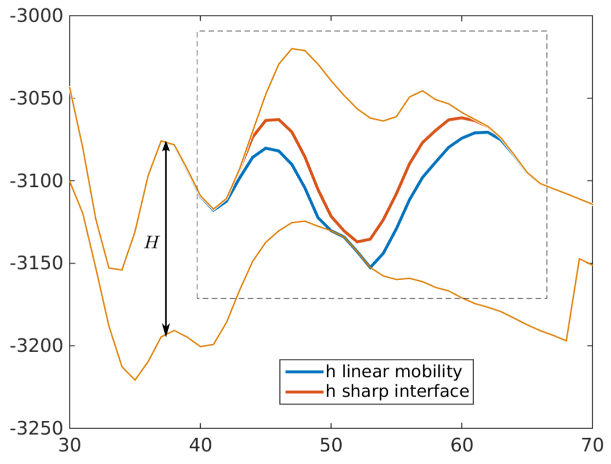


Figure 3.17: 1D intersection of the formation at $x = 48$, which is very close to the injection point, $(51, 51)$. The vertical axis shows the actual depth below the surface in meters. The two beige lines indicate the top and bottom of the formation. Here we study simulation 2 where the snapshots are taken at $T_T = 6$ years (top) and $T_T = 80$ years (bottom). The stapled frame indicates the location of the top plot. In both images, the smaller sharp-interface h inhibits migration past folds in the top surface.

Varying Permeability versus Constant Permeability The `MRST-co2lab` sharp-interface simulator also supports permeability integration. This feature is designed to capture vertical permeability variations, while still ignoring the vertical saturation variations. We can evaluate this feature by running this version of the sharp-interface simulator alongside the capillary-fringe simulator with varying permeability. In Figure 3.18, we see the saturation distribution of the sharp-interface simulator with permeability integration. This plot is not distinguishable from Figure 3.14 with the naked eye, but there is a difference. The comparison plots given in Figure 3.19 imply that the sharp-interface simulator succeeds in capturing the permeability variations. In the left part of the figure we see the difference between the two models when varying permeability is taken into account. These plots resemble the plots in Figure 3.15. On the right, we see the difference when the permeability integration is disabled in the sharp-interface model. Here the discrepancy between the two models is much greater. This weakness of the permeability-averaged sharp-interface model was also found in the study by Ligaarden and Nilsen [35].

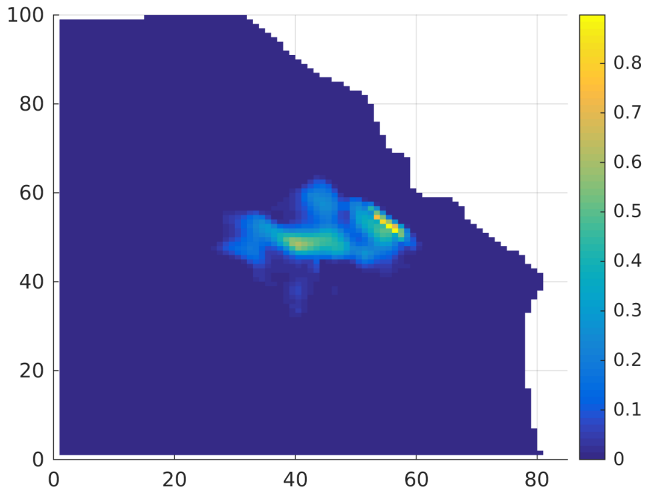


Figure 3.18: CO_2 saturation distribution for the sharp-interface simulator with permebility integration. The simulation is run for $T_T = 500$ years.

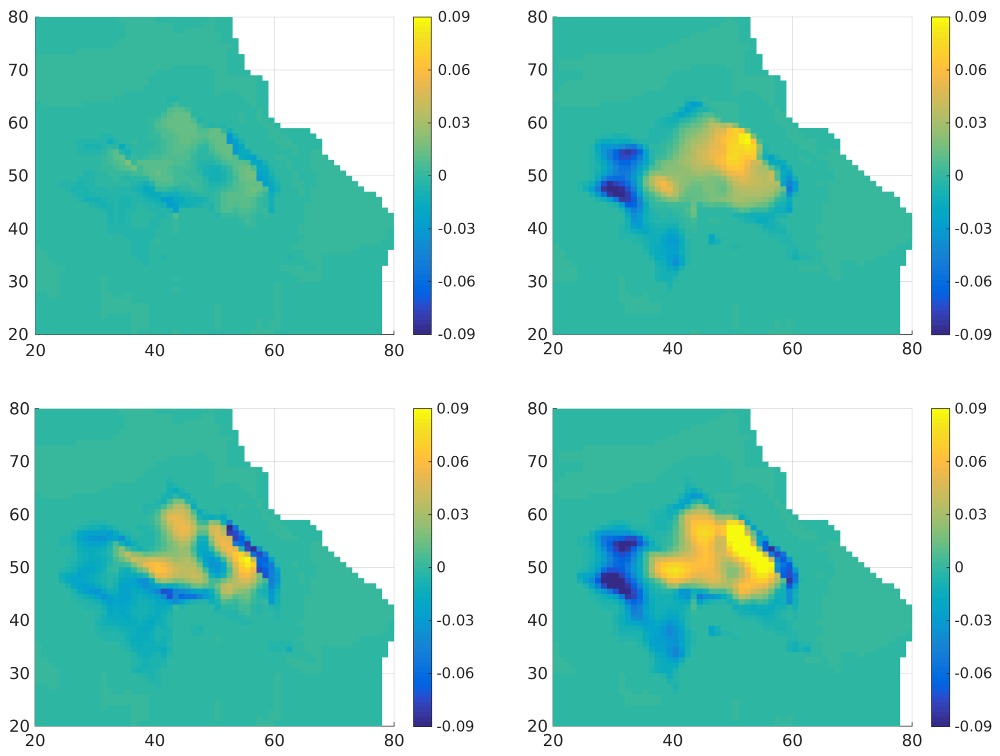


Figure 3.19: Sharp-interface simulator versus the capillary-fringe simulator with heterogeneous vertical permeability distribution. The plots show the difference in CO_2 saturation between the two simulators $S_c^{SI} - S_c^{CF}$, after running for $T_T = 500$ years. The top and bottom figures show simulations 4 and 5, respectively, whose descriptions are given in Table 3.2. On the left, both simulators are run with varying permeabililty. On the right, only the capillary-fringe model is run with varying permeability.

Capillary Fringe with Cubic Mobility

We continue with the Corey approximation for relative permeability, but now we apply the cubic form, thus $\lambda_c \propto s_c^3$. The various simulation runs are summarized in Table 3.3. For this case, the dissemblance between the two models is a lot greater and the main differences can be identified without plotting the difference, see for example Figure 3.20 where we have plotted simulation 7. From this plot we see that the coarse saturation distribution is more smudged and the spread has a slightly smaller radius. For a more quantitative inspection, the difference between the two models $S_c^{SI} - S_c^{CF}$, is given in Figure 3.21. On the left, two cases of Brooks-Corey with constant permeability (simulations 6 and 7). The small frames on the right show the corresponding simulations with varying permeability (simulations 9 and 10). These frames are almost identical, confirming our earlier observation with regards to the validity of the permeability integration option in MRST. This figure also shows that not only the pattern of the cubic mobility simulation is different, but also the magnitude. Here the difference is quite large, around 0.25 at the most (outside the colorbar), compared to 0.09 for the linear mobility in Figure 3.15.

In contrast to the linear-mobility simulations, the CO_2 in the cubic-mobilty simulation migrates less than the sharp-interface CO_2. Near the injection point there is a dark blue negative area indicating that the S_c^{CF} level is much higher here. We will attempt to explain this phenomenon in the following paragraph.

Table 3.3: An overview of the different simulation test cases run with cubic mobility.

Simulation	Perm.	Fine-scale Capillary Pressure	
		Model	Description
6	constant	Brooks-Corey	Eq. (2.23) with $C = 0.1g\Delta_\alpha\rho H$
7	constant	Brooks-Corey	Eq. (2.23)with $C = 0.4g\Delta_\alpha\rho H$
8	constant	Levrett J	Eq. (2.25) with $\sigma\cos(\theta) = 50\text{mPa}$
9	varying	Brooks-Corey	Eq. (2.23) with $C = 0.1g\Delta_\alpha\rho H$
10	varying	Brooks-Corey	Eq. (2.23)with $C = 0.4g\Delta_\alpha\rho H$

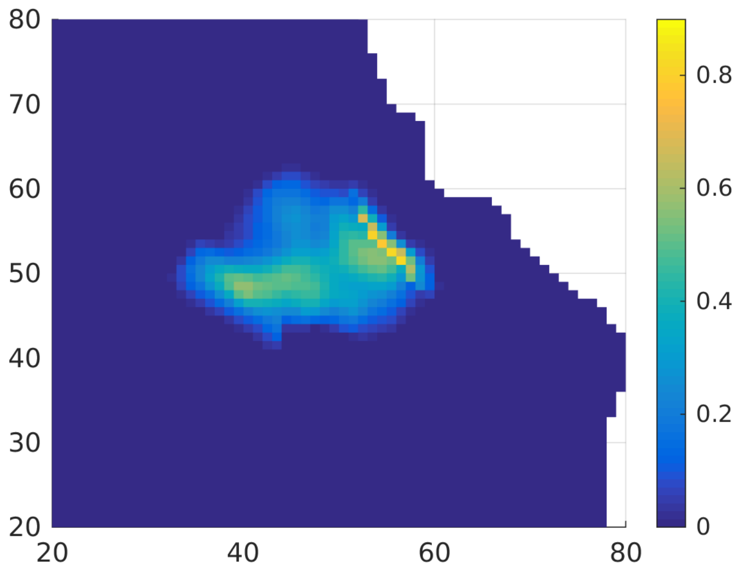


Figure 3.20: Capillary-fringe simulator run with cubic mobility. Here we show the coarse saturation distribution for simulation 7 after $T_T = 500$ years. More details on this simulations is given in Table 3.3.

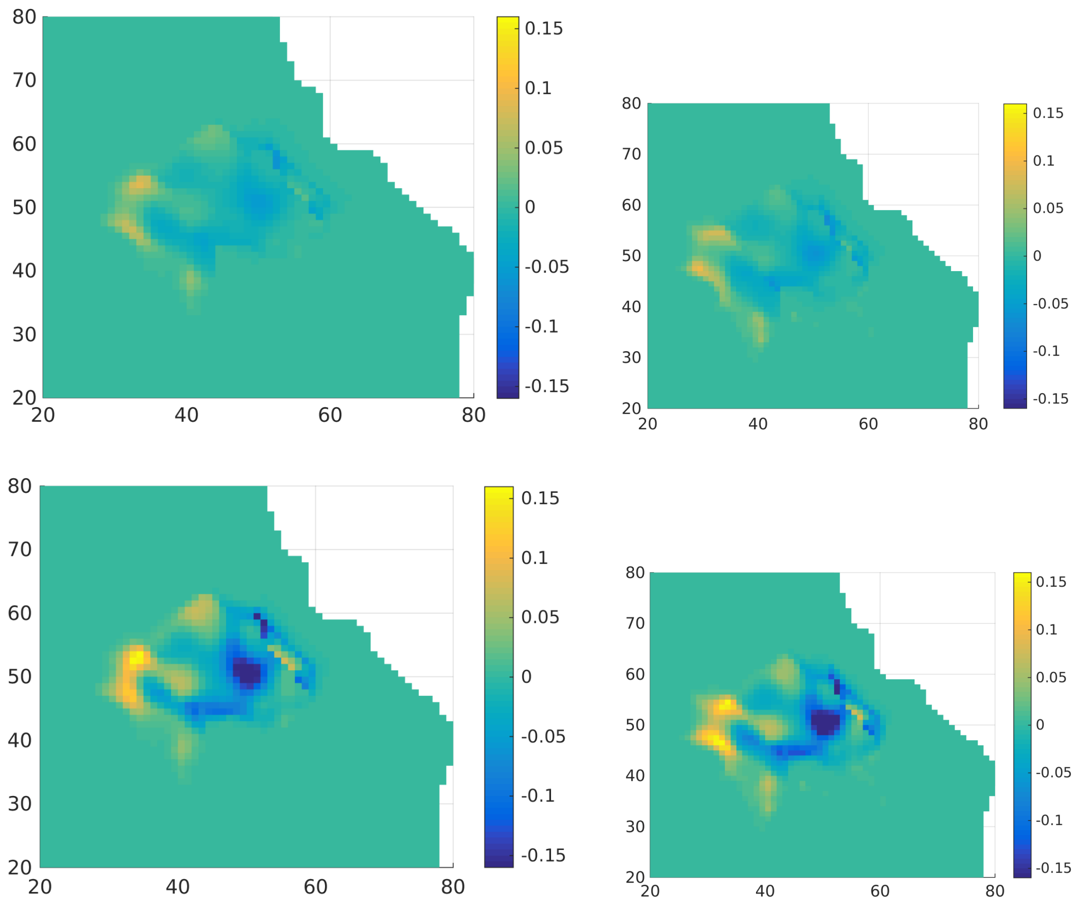


Figure 3.21: Sharp-interface simulator versus the capillary-fringe simulator with cubic mobility. The plots show the difference in CO_2 saturation between the two simulators, that is, $S_c^{\mathrm{SI}} - S_c^{\mathrm{CF}}$, after running for $T_T = 500$ years. The left frames show simulations 6 and 7, where the permeability is kept constant. The right frames show their variational permeability counterpart (simulations 9 and 10). More detailed descriptions of these simulations are given in Table 3.3.

Linear Mobility versus Cubic Mobility From the results above we can draw some important conclusions. The extent of the capillary fringe has a big influence on the results. It appears that the magnitude of any discrepancy between the two models scales with the height of the capillary fringe. Thus, when the fringe is high, the sharp-interface approximation is quite poor. Another interesting observation is that the cubic and linear mobilities produce different results. This can be explained by looking at the relative permeability as a function of the two different saturation distributions: Brooks-Corey and sharp interface. We recall Figure 3.16 where we plotted the saturation distribution for the two different simulators with equal S_c. In Figure 3.22, these distributions serve as input to cubic and linear permeability functions. For the sharp-interface distribution, the relative permeability curve will be the same, regardless of the exponent of the relative permeability function. For the Brooks-Corey saturation distribution, on the other hand, the exponent has an impact. The rate of motion for the CO_2 is controlled by the upscaled mobility $\mathbf{\Lambda}_c$. The upscaled mobility is proportional to the vertical integral of the curves in this figure, see Equation (2.9). Clearly, if we integrate curve 1 along z', this quantity will be smaller than the reciprocal integrals for curves 2 and 3. Based on the plots on the left in Figure 3.21, the effect of this reduced mobility seems to be critical for the cubic-mobility simulation. It leads to a reduced migration distance in some regions, particularly on the left half of the plots, where there is a positive area. Contrary, in the upper right corner of the plots, close to the coordinate position $(52, 58)$, there is a small area where it seems the capillary-fringe CO_2 has migrated further. For further insight we inspect the topography in this region. In Figure 3.23 we have plotted an intersection of the final result at $x = 52$. The situation presented in this plot is familiar, underneath the fold the coarse saturation is approximately equal for the two models. However, because of the greater h-value corresponding to the capillary-fringe model, the CO_2 is free to advance further. Thus, when the mobility is estimated to be on a cubic form, it is difficult to determine whether the sharp-interface model will overestimate or underestimate the spread of CO_2. The only thing we can say for sure is that if we have a flat surface, then sharp-interface modelled CO_2 will drift further away from the injection point.

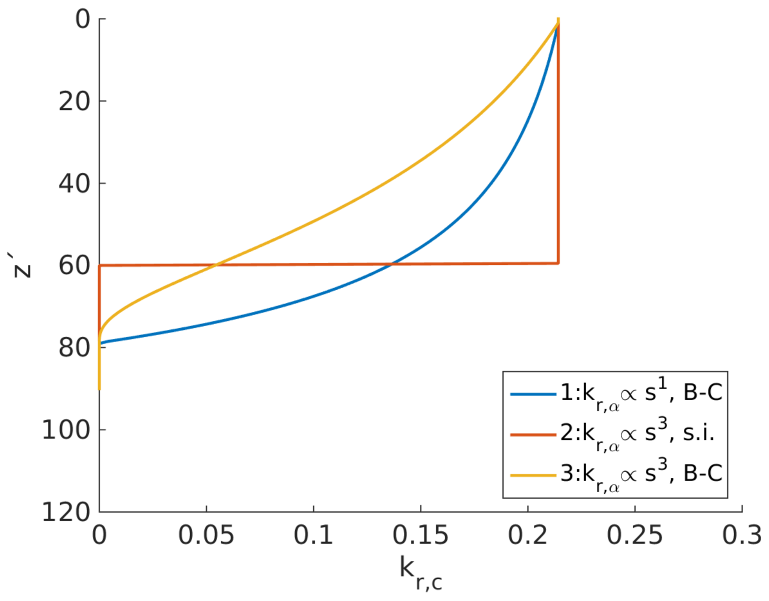


Figure 3.22: The sharp-interface (s.i.) and capillary-fringe (B-C) saturation distributions plotted in Figure 3.16 serve as input to linear and cubic relative permeability functions. For the expressions of these functions, we refer to Equation (2.21), derived in the mathematical background chapter, where the exponents are set to 1 and 3. If curves 1 and 2 are integrated along the z' axis, the result is the same. For curve 3 the result will be smaller.

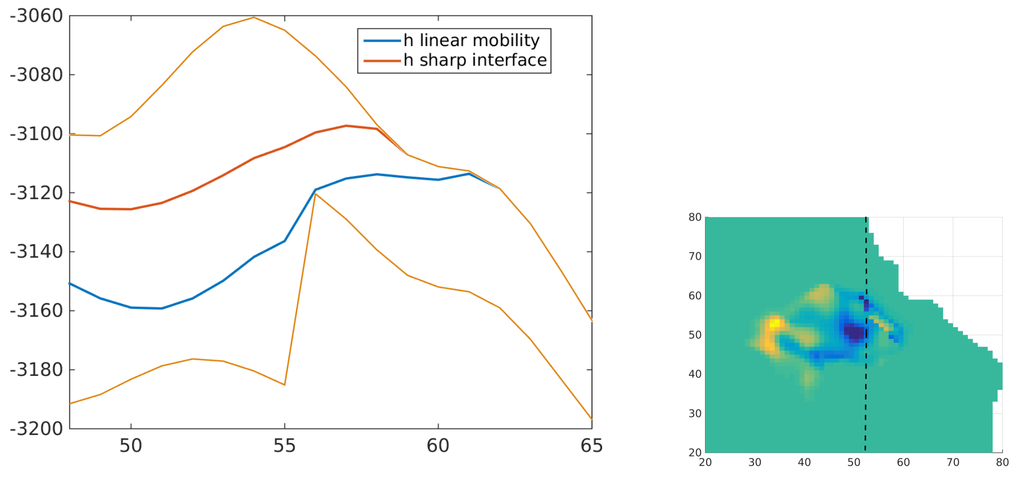


Figure 3.23: On the left, a 1D intersection of the formation at $x = 52$. The location is indicated by the stapled black line in the right plot, which is a replicate of the coarse-saturation plot in 3.21. The vertical axis shows the actual depth below the surface in meters. The two beige lines indicate the top and bottom of the formation. Here we study simulation 7 at the end of the simulation when $T_T = 500$. The shorter sharp-interface h inhibits migration when the top surface is undulating.

3.3 Performance Evaluation

A direct comparison of the runtimes of the MRST solver and the GPU solver is a bit like comparing apples and oranges, with respect to performance. First of all, the two solvers do not solve the exact same problem as they apply different models and use different grids. The MRST uses an unstructured grid as opposed to the structured Cartesian grid applied in the GPU implementation. Secondly, they are implemented in two quite diverse programming languages. CUDA/C++ is a low-level language that generates relatively optimized machine code, while MATLAB is a high-level language which, in general, comes with a performance penalty. Lastly, they apply different levels of floating-point precision. However, for practical purposes, a rough runtime comparison is useful. If we run a simulation of 250 time steps and isolate the execution time of the saturation solvers, the times are 0.60s for the unoptimized GPU solver and 4.4s for the MRST solver. Thus, with the help of GPU-acceleration we can simulate a more advanced model within a smaller time frame. The Johansen grid, with the domain size of 81×100, is a relatively small grid in terms of the GPU capacity and we expect even greater performance gains for larger grids. We will look deeper into this in the next section, where we will discuss the absolute performance of the GPU.

3.3.1 Absolute Performance

To fully exploit the GPU's computational capabilities, the domain needs to be of a certain size. By studying the absolute performance for different domain sizes in terms of cells, we can determine this size. For the absolute performance and optimization testing we will use the Utsira formation, which has been a CO_2 injection site since 1996. A visualization of the formation is given in Figure 3.24. For more details on the geological properties of the formation and the ongoing CO_2 storage project, we refer the reader to [36, 5]. The data set we are using is taken from the CO_2 storage atlas distributed by the Norwegian Petroleum Directorate [17, 26]. The formations in this atlas are typically much larger than the Johansen formation.

The Utsira allows for large domains in terms of cells, which is ideal for performance testing. We place 15 wells at different locations and let them inject CO_2 into the formation for $T_I = 200$ years followed by $T_M = 800$ years of migration, before running the tests. This scenario with 15 wells placed at random is perhaps not very realistic, but it ensures that the GPU has a sufficient workload. In Figure 3.25 we see the saturation distribution of the Utsira formation at $T_T = 1000$ years. As we can see, at this point, the majority of the cells contain CO_2, which is our intention.

The maximum resolution supported by the Utsira data set is 219×840. To obtain models with higher resolution, we interpolated the data from the biggest Utsira data set in MATLAB. However, when we reached a domain size that exceeded 418×1680 MATLAB ran out of memory. Our solution to this problem was to tile replications of the Utsira domain. An example of the replicated grid is given in Figure 3.26. This domain is clearly not realistic, but it serves for our purpose of absolute performance testing, where our main concern is the number of cells that the GPU can process.

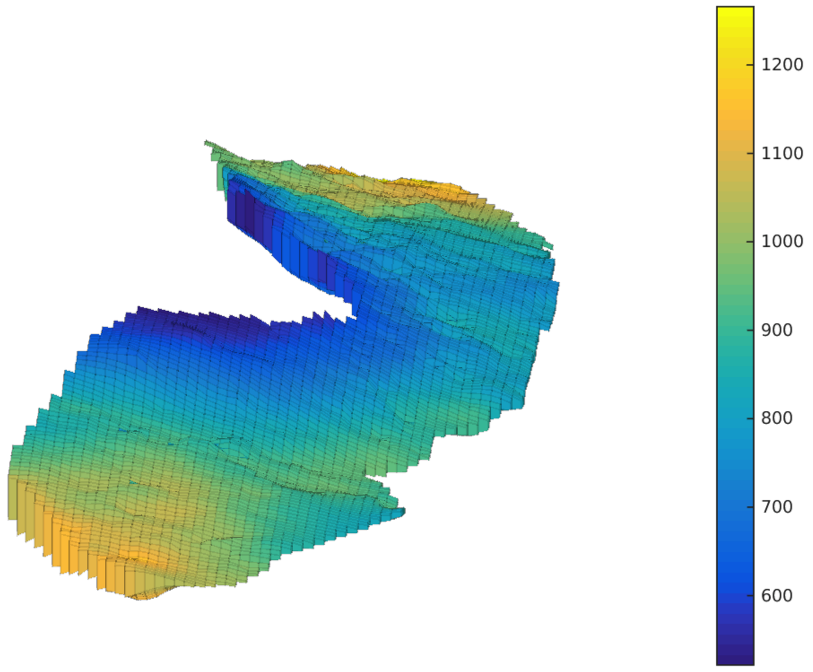


Figure 3.24: Plot depicting the shape of the Utsira formation. The colorbar indicates the depth below the surface.

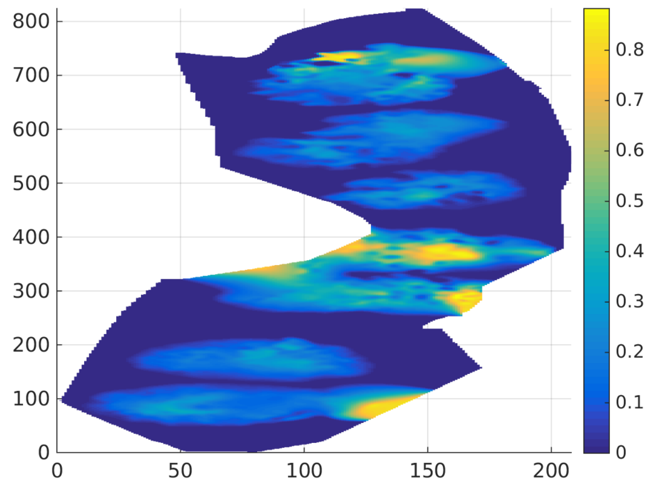


Figure 3.25: CO_2 distribution for the Utsira formation after injecting CO_2 for 200 years followed by 800 years of migration. There are 15 different injection points. This will be the basis case for the performance testing.

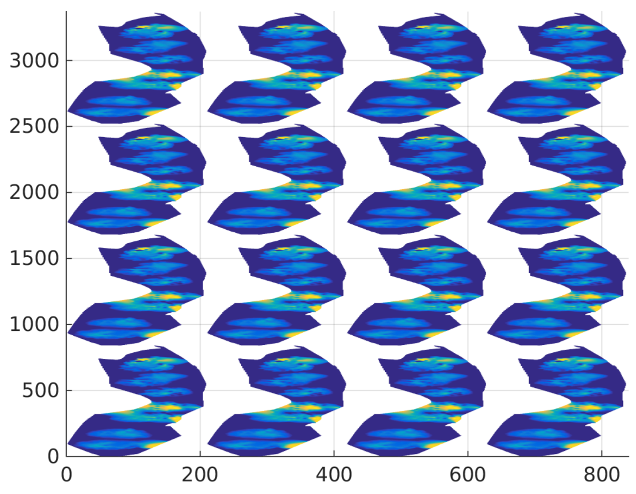


Figure 3.26: An example of a replicated Utsira grid used to create large domains required for the absolute performance testing.

In Figure 3.27 we have plotted the absolute performance for increasing resolution of the Utsira formation. It is clear that for small domains the GPU's processing capabilities are not exploited. When we hit a domain size of $2400 \times 600 \approx 1.6$ million cells, the graph stabilizes and we have hit maximum performance. At this point we manage to occupy the GPU hardware fully and overheads become insignificant. The absolute performance results prove that with the GPU we can simulate on large aquifers with acceptable resolution. Note that although the rectangular grid size is 2400×600, the actual number of active cells is lower since, as we can see in Figure 3.25, the physical domain is not rectangular.

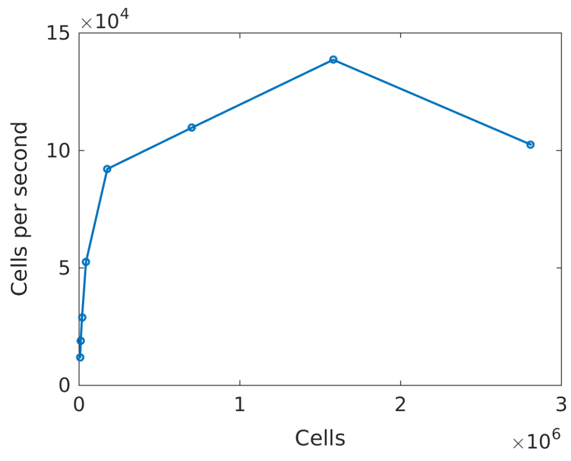


Figure 3.27: Absolute performance for the Utsira test case with fifteen wells shown in Figure 3.25. We run the simulator for 200 time steps and report the GPU time. We see that at 1.6 million cells, we hit the peak performance.

Hardware Specifications

For the testing we ran our simulator on a NVIDIA Tesla k20 graphics card. This GPU has a peak single precision floating point performance of 3.52 Tflops and the core clock runs at 706 MHz. The memory bandwidth is 208 GB per second and the total number of CUDA cores is 2496 divided among 13 streaming multiprocessors. It has a reasonable sized global memory of 4.7 GiB.

3.3.2 Optimizing Performance

The objective of this thesis was first of all to make a GPU simulator that could perform the complete vertical integration, but secondly, it was also a priority to achieve good performance. Although our simulator outperforms the `MRST-co2` simulator, this does not mean that our code is optimal. Even an elementary GPU implementation of a parallel problem will likely increase performance. In this section we will discuss the optimization potential and the optimzation effort made on our GPU implementaion. Central points will be how we identified the performance bottlenecks and how we adressed the associated problems. Optimizing a parallel program is not a trivial task, and the intricate GPU architecture does not make it easier. It is important to ensure that the parallel algorithm is efficient before turning to GPU-specific optimizations. If say, the algorithms we apply are not well balanced with respect to parallelism, then tuning the GPU will not make much difference.

When making GPU-specific optimization we need to think about memory with respect to location and access patterns as well as transfers. At the same time we have to expose as much parallelism as possible, not only within the GPU hardware, where both thread level and multiprocessor level need to be taken into account, but also between the CPU and GPU. For the kernels, there are three main factors that affect the performance: instruction throughput, memory throughput and latencies. NVIDIA provides a performance profiling tool called Visual Profiler. This tool analyzes the code and provides many performance metrics, giving a great insight to the application and helps identify performance bottlenecks.

Kernel Work Division

The first step for code optimization is to identify which kernels take up the majority of the computational time. There is no sense in spendig great optimization effort on a kernel which is responsible for 1% of the execution time. The results provided by Visual Profiler, given in Table 3.4, show that the time integration kernel is dominating. If we study the work division of comparable codes, this is quite surprising. For the shallow water simulator [6], which solves a related problem, the flux kernel is, without question, the most time consuming, with 87.5%. In this simulator, the time integration kernel only takes up 12% of the exectution time. The same trend is presented in the master thesis on the sharp-interface GPU simulator [49]. The cause of this inconsistency lies in the added complexity in our time integration kernel. In our implementation, the time integration kernel has to perform an additional task, namely the numerical computation of h^{n+1}, described in Section 3.1.5.

Table 3.4: Runtime percentage for the four different kernels.

Coarse mobility	Flux	Time-step red.	Time integration
25.4%	11.6%	0.6%	62.3%

Optimizing the Time Integration Kernel

From the results in Table 3.4, the time integration kernel is a clear optimization priority. The time-consuming part of this kernel is undoubtedly the computation of h^{n+1}. Our first approach was to investigate the corresponding brute force algorithm, which is quite slow as it requires very many evaluations due to the refinement step. However, the biggest problem is that it does not map well to the GPU architecture. As suspected, there is a lot of branch divergence, especially for the while-statement in Algorithm 2, where 60% of the executions are divergent. The top chart in Figure 3.28 shows the consequences of the varying loop size; the threads are inactive about 50% of the time. This is a poor result if we compare it to a kernel with a more "ideal" execution distribution, shown in the bottom chart. Not only is there a bad load balance in the warps but this also affects the blocks. The top chart in Figure 3.29 shows that there is substantial variation in the execution time of the different multiprocessors. Optimally, the work distribution should resemble the bottom chart, where an optimized code example is presented.

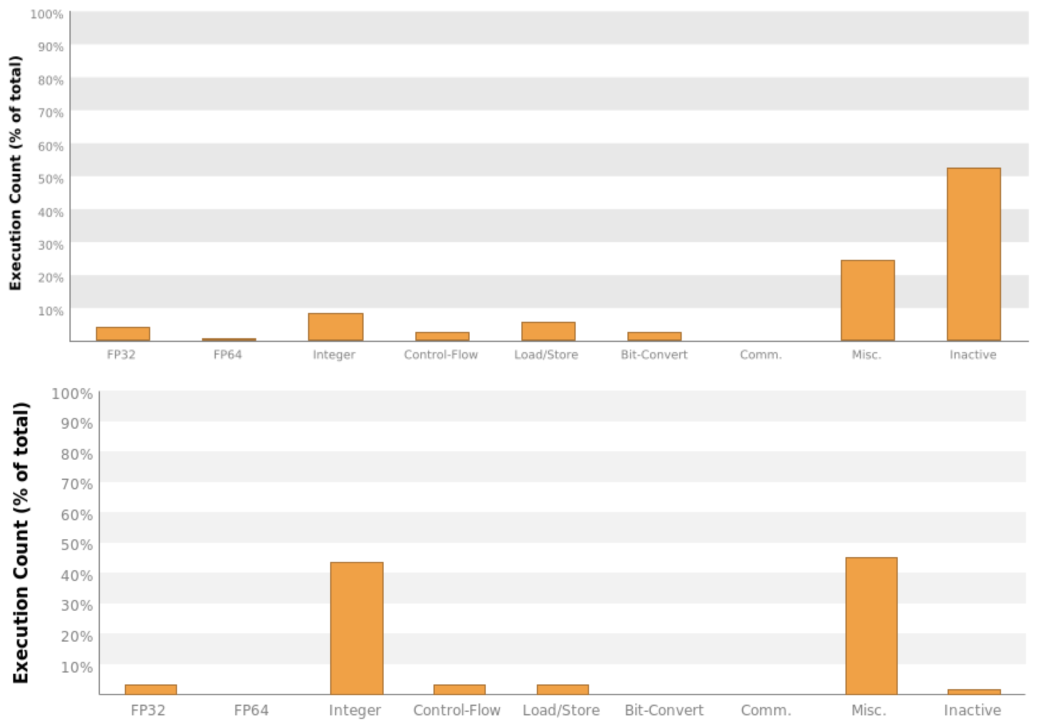
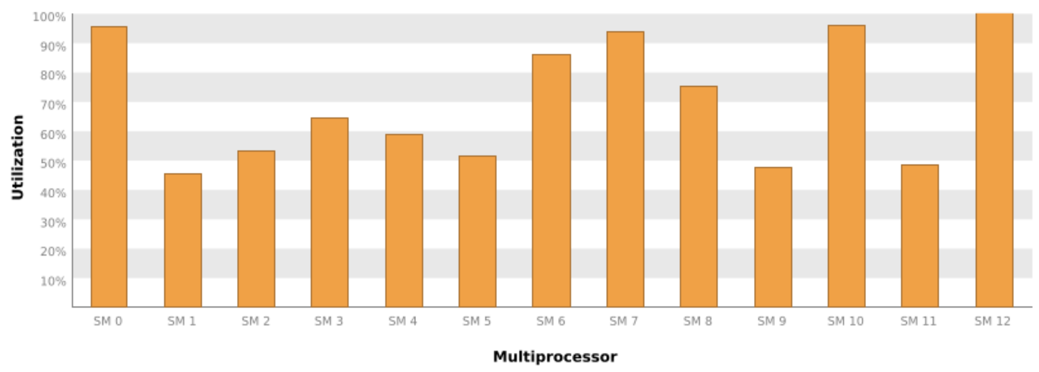


Figure 3.28: Percentage of thread execution cycles devoted to executing instructions in the indicated class. On the top, the time integration kernel in our simulator. On the bottom, an optimized matrix multiplication implementation taken from the CUDA toolkit samples [47]. Note the great difference in the percentage value of the inactive threads bar on the right.



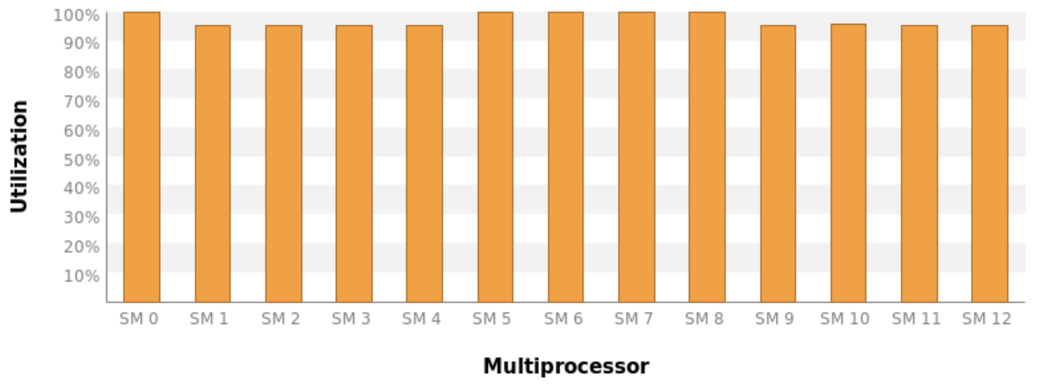


Figure 3.29: Work distribution among the streaming multiprocessors (SMs). On the top, the time integration kernel in our simulator. On the bottom, an optimized matrix multiplication implementation taken from the CUDA toolkit samples [47]. We see that our simulator has an uneven work-load distribution which puts constraints on the performance.

Newton Optimization

An alternative algorithm is Newton's method derived in Equation (3.3). The convergence of Newton's method is highly dependent on the starting point h_0. When testing this method with the proposed initial starting point $h_0 = h^n$, the convergence is good for most cells. However problems arise when the saturation gap between two consecutive time steps is big. When this happens, Newton can overshoot h, such that we may end up in an area where the denominator $F'(h)$ in Newton's method evaluates to zero, that is

$$F'(h) = \frac{1}{H}(s_c(h-H) - s_c(h)) = 0 \iff s_c(h-H) = s_c(h). \tag{3.4}$$

A visualization of this problem is given in Figure 3.30.

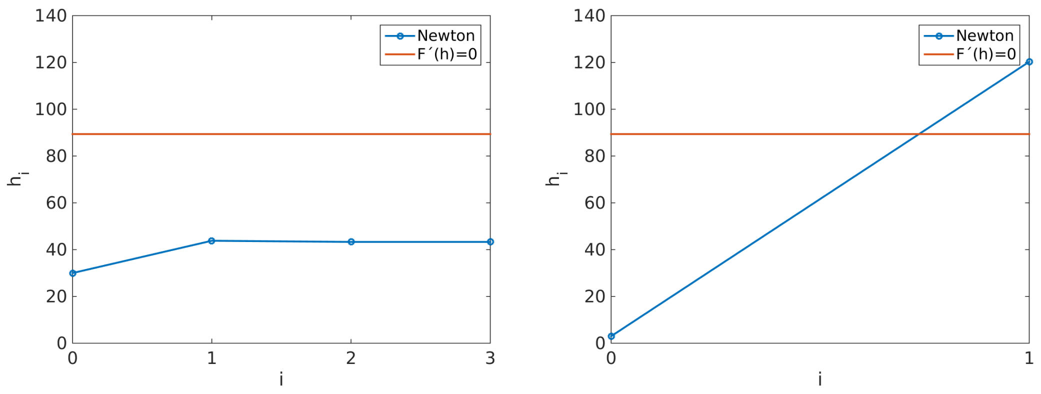


Figure 3.30: Behavior of the Newton method when solving for h^{n+1} for one of the cells in the Utsira simulation with $S_c = 0.63$ and $H = 47.9$. On the left $h_0 = h^n = 30$ and we see that within three iterations we reach $h^{n+1} = 46.6$. On the right we have a bad initial guess with $h_0 = h^n = 3$ and we see that Newton overshoots h on the first iteration so that h_1 is above the red line, where Newton's method terminates.

We can avoid this stationary point by putting a restriction on h_i, such that we do not end up in this "undefined" area. Based on (3.4), we need to find out when $s_c(h-H) = s_c(h)$. If we study the $s_c(\tilde{z})$ curve illustrated in Figure 3.31, this occurs when $s_c(h-H) = 1 - s_{b,res}$. Hence, we need to solve $s_c(l_{cap}) = 1 - s_{b,res}$, where the quantity l_{cap} can be recognized as the height of the capillary fringe. Once we have found the value of l_{cap}, then we can for example set $h_i = \min(h_i, H + l_{cap} - 0.1)$. Although with this approach we avoid the zero division, Newton now encounters a different form of instability,

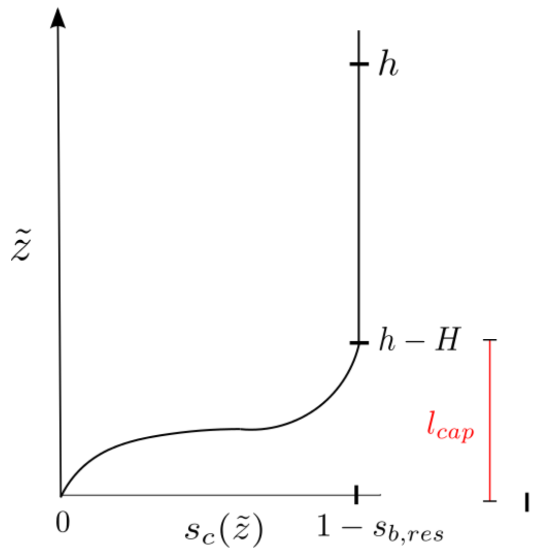


Figure 3.31: The saturation curve $s_c(\tilde{z})$. When $h - H = l_{cap}$, then $s_c(h - H) = s_c(h) = 1 - s_{b,res}$ and the derivative in Newton's method will evaluate to zero.

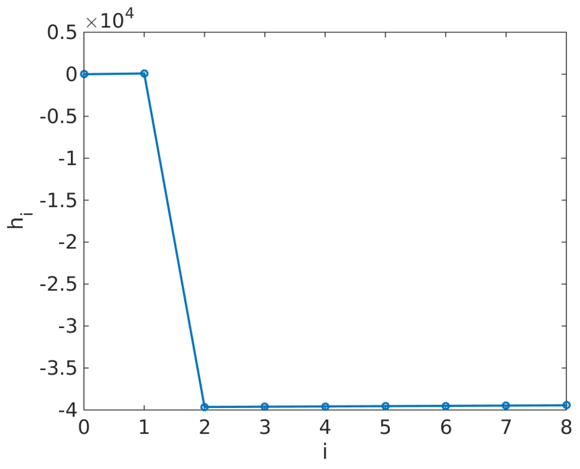


Figure 3.32: Behavior of the Newton method with the restriction $h_i = \max(h_i, H + l_{cap} - 0.1)$. Here we solve for h^{n+1} for a cell in the Utsira simulation with $S_c = 0.63$ and $H = 47.9$. We start with a bad initial guess; $h_0 = h_n = 3$ and we see that Newton becomes very unstable.

see Figure 3.32. Since the derivative $F'(h)$ is a very small number, Newton makes a big jump.

Based on the analysis above, clearly Newton struggles when we have an overshoot caused by a bad initial guess. However, the root of the problem seems to be related to the area around $H + l_{cap}$. This phenomenon can be explained by studying the function $F(h)$, which we for simplicity restate here,

$$\begin{aligned} F(h) &= S_c - \frac{1}{H}\int_0^h s_c(\tilde{z})\,\mathrm{d}\tilde{z} \approx S_c - S^*_{c,h} \quad h \leq H, \\ F(h) &= S_c - \frac{1}{H}\int_{h-H}^h s_c(\tilde{z})\,\mathrm{d}\tilde{z} \approx S_c - S^*_{c,h} \quad h > H, \end{aligned} \tag{3.5}$$

where we have introduced the variable $S^*_{c,h}$ to denote the numerical approximation for the integral term. From the plot of $F(h)$ given in Figure 3.33, we see that for large values of h the gradient is very small, which makes Newton unstable. We may also observe that the gradient is quite small for very small h, which can explain why small initial estimates of h_0 can result in very large overshoots, as demonstrated in the above examples in Figures 3.30 and 3.32.

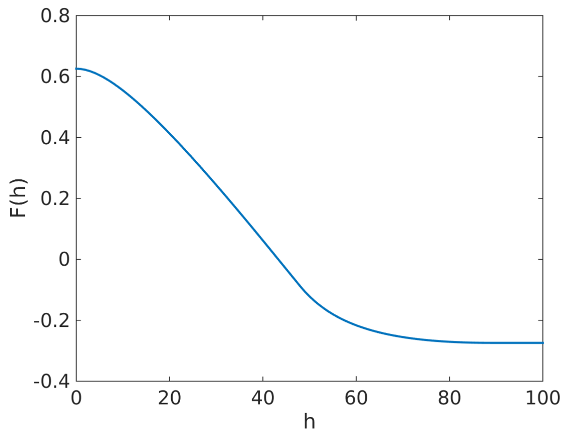


Figure 3.33: The function F(h) applied in Newtons method (3.3), derived in Equation (3.5). We see that when $h > 50$, the gradient becomes very small causing problems for Newton's method. Here $S_c = 0.63$ and $H = 47.9$.

Robust Newton Optimization

We have established that Newton's method has great difficulties converging for some cases. The question is, can we still benefit from this method? If we could filter out the problem cases, we could use a different algorithm on these cells, while performing Newton on the remaining ”normal” cells. This attractive solution presents a new challenge: how can we identify the problem cases beforehand? From the examples above it appears to be difficult to predict Newton's behaviour. For instance a large change in saturation can be problematic as this implies that h^{n+1} will differ a lot from the initial guess $h_0 = h^n$. We can also face problems when the saturation is relatively stable, but very high. In these cells h will lie in the area around $F'(h) = 0$, where the gradient is very small which possibly leads to divergence. Thus, we find that the best solution is to simply set an iteration limit for Newton's method. This means that we run Newton's method on all cells and flag the cells that do not converge within the chosen iteration limit. The flagged cells must then be processed by an alternative algorithm run by a new kernel. In the histograms in Figure 3.34, we show the convergence rate for the CO_2 filled cells at two different points in time, $T_T = 20$ and $T_T = 1000$. After 1000 years, we see that the percentage of cells that converges within two iterations is much higher than after 20 years. This is because at $T_T = 1000$, we are in a much more stable phase of the simulation, where h does not vary so much between consecutive time steps. For both points in time, 90% of the cells converge within three iterations. By increasing the number of iterations to six we achieve a small percentage increase. The decision with respect to where we set the iteration limit will depend on the cost of an additional Newton iteration versus the cost of solving this cell by another method.

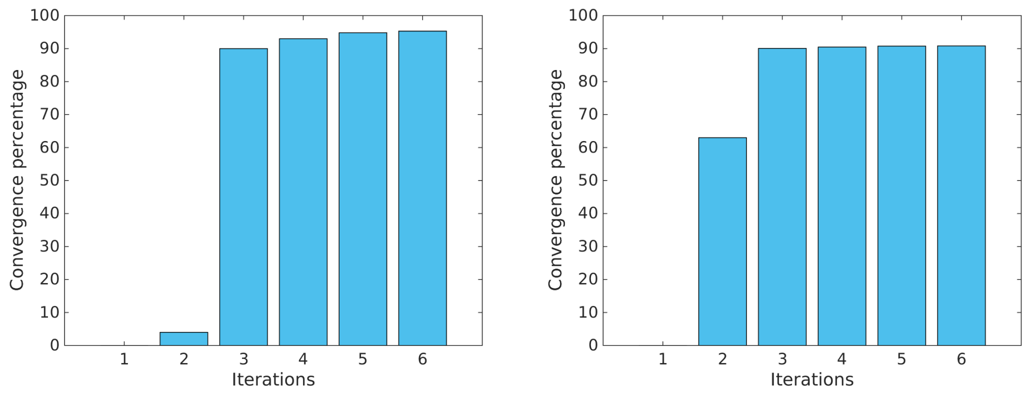


Figure 3.34: Number of iterations required for Newton's method to converge. The y axis gives the percentage of the CO_2 filled cells that has converged within the indicated number of iterations. In the left frame at $T_T = 20$ years, and on the right $T_T = 1000$ years. The tolerance is set to 10^{-6}.

To sort the remaining flagged cells, that is, those cell that do not converge within the iteration limit, we follow the approach outlined in [52]. We create two new arrays of equal length: the index array containing the linear indexes of the cells and the flag array containing only zeros. After running Newton's method in the time integration kernel we update the flag array by placing a 1 at the linear index of the non-convergent cells. The index array is then compacted using a CUDPP[28] kernel, resulting in a new array where the first contigious elements are the indexes of the non-convergent cells.

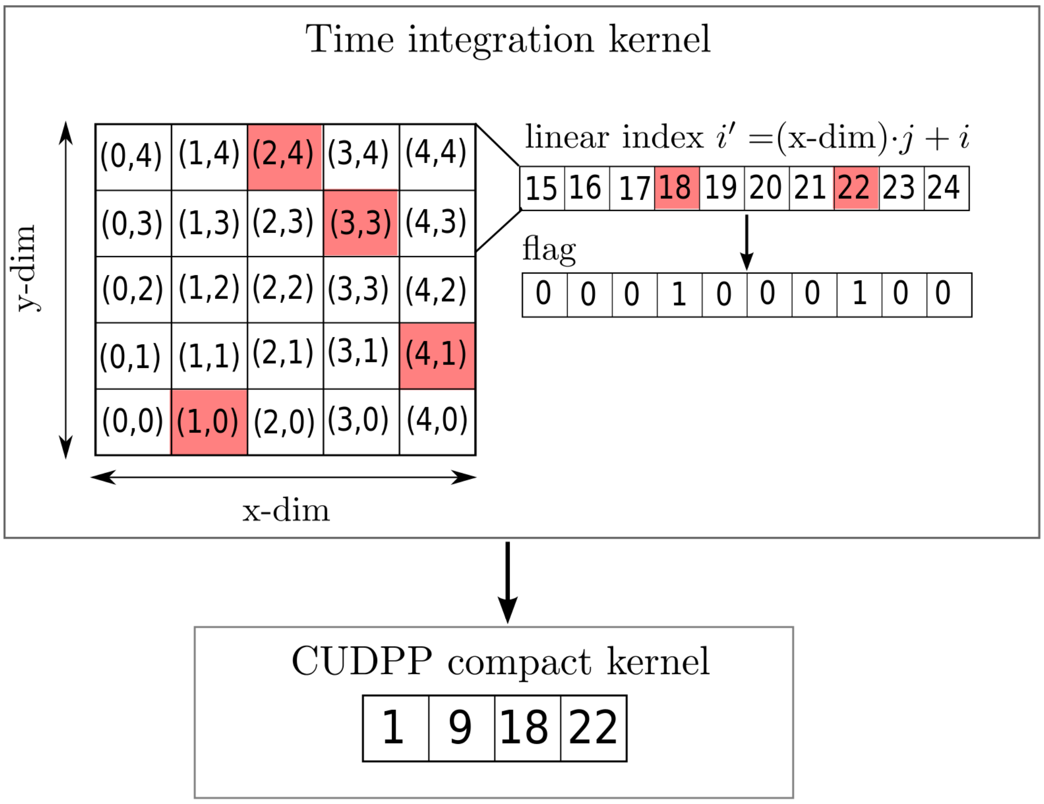


Figure 3.35: The time integration kernel identifies the non-convergent cells and updates the corresponding linear index in the flag array. Next, the CUDPP compact kernel stores the flagged indexes contigiously in an array.

We can now proceed with a new kernel to find the updated h-values of the cells whose indexes are now stored in the index array. For these cells we propose the bisection method, introduced in Section 3.1.5. This method converges slowly, but it is reliable. Since this kernel only is responsible for about 10% of the cells, we can assign a group of threads to each cell. More specificly, we will assign one warp to each cell and attempt to increase performance by computing the integral in (3.5) in parallel. For every iteration of the bisection method, each thread will be assigned a part of the integral in (3.5). The threads store their partial integral sum in shared memory and we perform a shared memory reduction to compute the total value of the integral.

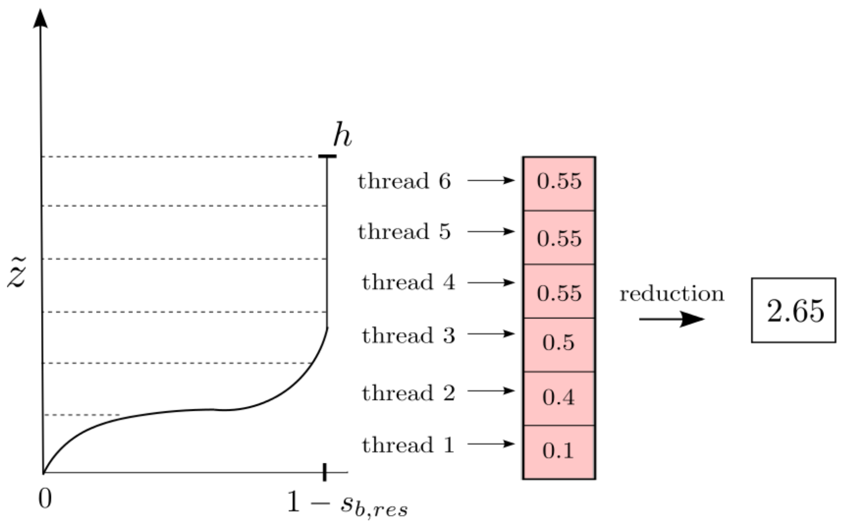


Figure 3.36: Illustration of how to compute the integral part of the function $F(h)$ in parallel. Each thread within a warp is assigned a part of the integral, indicated by the stapled lines. The thread then stores its computed partial sum in shared memory, marked in pink. After this we run a parallel reduction on the shared memory.

We make an additional optimization for the integral in (3.5), which has to be computed at every iteration for both Newton's method and the bisection method. Since h_i will not change too much between iterations, especially for the final iterations of Newton's method, we are in practice recomputing parts of the integral on every iteration. To exploit this we store the value of S^*_{c,h_i} from the previous iteration and add and subtract from this sum with respect to the new integration limits of h_{i+1}. When we implement this for the bisection method we make an adjustment where each cell is assigned *two* warps instead of one. The first one takes care of the new lower integration limit, while the second one takes care of the upper integration limit.

With the above optimization, the added cost of running an extra iteration of Newton's method becomes minimal. Thus, we set the iteration limit to 6 based on the graphs in Figure 3.34. Note that if the derivative in Newton's method becomes zero, the iteration loop is terminated and the cell is automatically flagged. The implementation of the "robust Newton optimization" results in a new program flow pictured alongside the original flow in Figure 3.37.

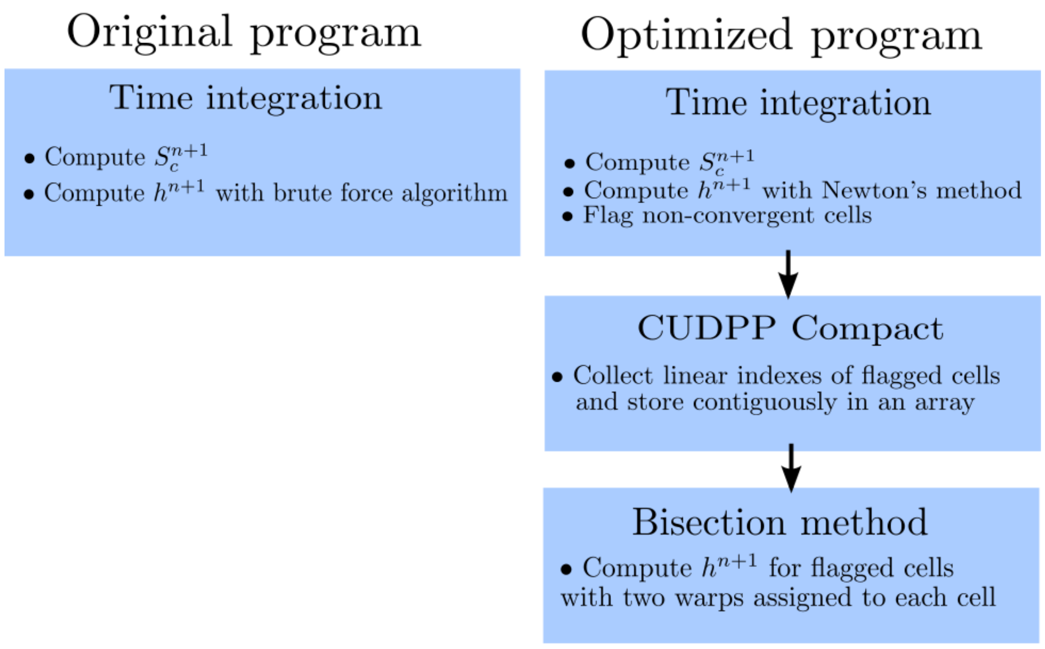

Original program
Time integration
• Compute S_c^{n+1}
• Compute h^{n+1} with brute force algorithm
Optimized program
Time integration
• Compute S_c^{n+1}
• Compute h^{n+1} with Newton's method
• Flag non-convergent cells
CUDPP Compact
• Collect linear indexes of flagged cells and store contiguously in an array
Bisection method
• Compute h^{n+1} for flagged cells with two warps assigned to each cell

Figure 3.37: The segment of the program flow affected by the robust Newton optimization before and after. Each blue box represents a CUDA kernel. With the new optimization, we have added two new kernels as a consequence of the cell division in the optimization approach.

Simpson Optimization

Inspite of the thoughtful implementation design, the performance gain of the robust Newton's method was marginal. Isolated, the new time integration kernel applying Newton's method is much faster than the original one. It runs about twice as fast, depending on the stage of the simulation. However, the bisection method is a performance killer. Requiring up to 30 iterations to reach convergence, the increased amount of computational resources per cell (64 threads) is simply not enough. Increasing the number of threads is not an option as this will require thread syncronization which comes with a cost and the overhead associated with each iteration will be higher. Also, for large domains, there might not be enough threads to supply each flagged cell with more than 64 threads. Thus, at this point we have three options; we can either attempt to decrease the number of required iterations, decrease the time for each iteration or abandon the bisection method and search for other robust methods. We choose option number two, decreasing the execution time per iteration.

Simpson's method is an alternative numerical integration method to the trapezoidal rule

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f(a+b) + f(b)\right].$$

This method also comes in an extended version known as the composite Simpson's rule [11], in which the interval $[a, b]$ is split up into n subintervals where Simpson's rule is applied to each subinterval. This approach gives a better approximation for functions which are not smooth over the entire interval, such as ours. For smooth functions, Simpson's method typically has a faster convergence rate than the trapezoidal method. By implementing Simpson's method we can thus increase the size of the subintervals in the numerical integration, whilst maintaining the same precision level. For our implementation, Simpsons's method is applicable for the coarse-mobility kernel, which we recall also performs a numerical integration, as well as the bisection and time integration kernels. Replacing the trapezoidal rule with Simspon's rule resulted in a total speed-up of 1.25 for the Utsira test case with resolution 103×416. The new distribution of the execution time among the kernels is given in Table 3.5.

Table 3.5: Runtime percentage for the six different kernels in the optimized implementation.

Coarse mob.	Flux	Time-step red.	Time int.	CUDPP	Bisection
15%	30.6%	1.6%	17%	0.8%	35%

Optimization Summary

It has proved to be challenging to optimize the time integration kernel or, more accurately, the computation of the updated h value. The achieved speed-up of 1.25, based on analysis and code improvents directed at the numerical integrations, is not particularly high. It is likely that a comparable speed-up would be achieved by applying other known optimization practices which have proved successfull for similar GPU implementations. Examples of this include early-exit optimization [52] and optimal block size configuration, both of which were found beneficial for the shallow-water GPU implementation [6] and the sharp-interface GPU implementation [49]. However, we chose to pursue the numerical integrals because these are the new components introduced by the capillary-fringe model. An inspection of these components is much more valuable for future development of capillary-fringe based simulators. Although our optimization effort did not bare great fruits for our implementation, we have laid the groundwork for future implementations. From the results in Table 3.5, it is clear that the bisection method, applied to the flagged cells, should be an optimization priority for future work.

Chapter 4

Concluding Remarks

The main purpose of this work was to develop a more comprehensive VE-based simulator for studying the movement of CO_2 injected into a saline aquifer. In contrast to many VE simulators, which model the transition zone between CO_2 and brine as a sharp-interface, our simulator applies a smooth approximation known as a capillary fringe. To obtain comparable runtimes for the more computationally-demanding smooth approximation, our simulator was implemented on a GPU. The new simulator allowed us to evaluate the sharp-interface model and check when the two modelling assumptions produce different results.

4.1 Summary

Simulation Behaviour We found three main factors which affect the behavior of the simulations. Firstly, if the mobility is assumed to be a cubic function of saturation, which is typical for a highly heterogenous media, the rate of CO_2 movement for the capillary-fringe model will be much lower than the sharp-interface model. Based on this fact, one may think that the sharp-interface model will always overestimate the migration radius of the CO_2. However, as we experienced, this is not necessarily true, which leads us to our next discovery: If we have an undulating top-surface, the CO_2 can become structually trapped inside top-surface ”pockets”, where it will remain until it reaches a certain saturation level which allows it to overflow the pocket. An analogy to this is a river that runs down into a ditch and will not flow further until the ditch has been filled up with water. The required saturation level for CO_2 to overflow is unequal for the two simulators because of differences in the underlying mathematical model. Thus, in the sharp-interface model, with a higher saturation requirement for overflow, the CO_2 will be stuck inside the pockets for longer periods of time, slowing down the advancement of the plume front. The effect of the topography is present regardless of the degree of the mobilty function. For a linear mobility approximation, where the mobility rate will be equivalent for the two models, the capillary-fringe modelled CO_2 will spread out more if there is structural trapping. The third factor among our findings involves the height of the capillary fringe. If the height is assumed to be high, this will amplify any discrepancy between the two simulators. In other words, the discrepancy between the two models is proportional to height of the capillary fringe.

Performance Through GPU acceleration we managed to create a capillary-fringe based simulator which showed no disadvantage in terms of speed. Based on numerous previous studies documenting the GPU's effect on explicit schemes for conservation laws, this was expected. Thus, a more interesting aspect is the level of hardware utilization. When profiling the GPU code to identify possible optimizations, we found several performance issues. The main cause of these issues was that the numerical integrals, introduced by the capillary-fringe model, did not map so well to the GPU architecture. As is common practice for GPU implementations of finite-volume grids, we assigned one thread to each cell. However, this lead to an imbalance between the threads as the workload corresponding to the multiple numerical integrals was proportional to the CO_2 content of the cells. This problem proved to be difficult to solve. By making some changes, mostly with respect to the numerical methods applied, we achieved a small improvement. This included replacing the trapezoidal rule with the more accurate Simpson's rule for the numerical integration. Further, we replaced the stable, but slow, "brute force" algorithm, used to solve the updated interface height h with a combination of two classic root-finding algorithms. For the majority of the cells we applied the fast, but slightly unstable, Newton method and for the rest the slow, but robust, bisection method. We believe there is still considerable potential for optimizing the capillary-fringe related parts of the GPU program.

We have also showed that the GPU simulator is capable of simulating on large aquifer models, such as the Utsira formation, while retaining an acceptable resolution.

4.2 Further Research

The results from this thesis has provided a basis for research on more intricate VE models. There is still work to be done. The topics of future research can be divided into four different categories.

Model Expansion Although our model formulation includes the effects of a non-linear mobility, a non-linear relationship between capillary pressure and saturation and a varying permeability distribution, there are still many simplifications. As already pointed out, we have not incorporated the effect of hysterisis which can cause residual trapping. Other important properties which should be taken into account are compressibilty and dissolution. For further insight into these effects we refer to [41]. In conclusion there are many possibilties to expand the mathematical model.

Code Optimization With respect to code efficiency we first suggest looking at some standard optimization routines, such as the early-exit and block configuration optimizations presented earlier. Secondly, one should address the biggest performance bottleneck, which is the computation of the updated h value. One option is to continue on the path we have laid out, where the cells are sorted into two different groups based on the convergence of Newton's method. Here we suggest attacking the bisection method, either by finding a smart way to reduce the initial interval $[a, b]$ of the method, reducing the number of required iterations, or considering other robust bracketing methods with higher convergence. Another option is to try to manipulate Newton's method so that convergence is guaranteed. One may also attempt to group the cells with equivalent interface heights in the same block. This way one can reduce the number of divergent execution paths, maximizing the number of active threads.

Numerical Accuray Evaluation When verifying the accuracy of our simulator we made some simplifications to our method so that it represented a sharp-interface simulator. We then made a comparison with the results of the `MRST-co2lab` sharp-interface simulator. As discussed, this test does not give us any information about the accuracy of the numerical integration procedures. An assessment of this aspect would require a different type of numerical analysis, as there is no other "correct" simulator to compare with. One would have to implement several different integration algorithms and measure the differences between these implementations with a varying number of evaluation points. This analysis is valuable as it might find that a high accuracy is not necessary which would increase performance - or it could be the other way around.

New Applications With a fast fully-integrated VE-based simulator at hand one can gain further insight into how the formation properties influence the migration pattern. For example, one can look deeper into the effects of top-surface morphology. We know that often there are great uncertainties in the available geological data for potential injection sites. In the paper of Syversveen et al. [51], the effects of variations in top-surface morphology are assesed through statistical analysis. Here, numerous top-surface morphologies are created stochastically and by running a CO_2 migration simulation on the different realisations one can examine if the uncertainties have an impact. We propose an equivalent study applying our simulator instead.

Bibliography

[1] The Matlab Reservoir Simulation Toolbox, version 2014a,. `http://www.sintef.no/MRST/`, May 2014.

[2] O. Andersen, S.E. Gasda, and H.M. Nilsen. Vertically averaged equations with variable density for CO_2 flow in porous media. *Transport in Porous Media*, 2014.

[3] J. Bear. *Dynamics of Fluids in Porous Media.* Courier Corporation, 2013.

[4] S. Benson and P. Cook et al. Underground geological storage. In *Carbon dioxide capture and storage*, pages 195–276. Cambridge University Press, 2005.

[5] P.E.S. Bergmo, A.-A. Grimstad, E. Lindeberg, F. Riis, and W.T Johansen. Exploring geological storage sites for CO_2 from Norwegian gas power plants: Utsira South. *Energy Procedia*, 1(1):2953–2959, 2009.

[6] A. R. Brodtkorb, M.L. Sætra, and M. Altinakar. Efficient shallow water simulations on GPUs: Implementations, visualization, verification and validation. *Computers and Fluids*, 55:1–12, 2012.

[7] R.H. Brooks and A.T. Corey. Properties of porous media affecting fluid flow. *Journal of the Irrigation and Drainage Division*, 92:61–90, 1966.

[8] M.J. Castro, S. Ortega, M. de la Asunción, J.M. Mantas, and J.M. Gallardo. GPU computing for shallow water flow simulation based on finite volume schemes. *Comptes Rendus Mécnique*, 339(2-3):165–184, 2011.

[9] F. Chen and J. Shen. A GPU parallelized spectral method for elliptic equations in rectangular domains. *Journal of Computational Physics*, 250:555–564, 2013.

[10] Z. Chen, G. Huan, and B. Li. An improved IMPES method for two-phase flow in porous media. *Transport in Porous Media*, 54:361–376, 2004.

[11] E. Cheney and D. Kincaid. *Numerical Mathematics and Computing.* Cengage Learning, 7 edition, 2012.

[12] H. Class, A. Ebigbo, R. Helmig, H. K. Dahle, J. M. Nordbotten, M. A. Celia, P. Audigane, M. Darcis, J. Ennis-King, Y. Q. Fan, B. Flemisch, S. E. Gasda, M. Jin, S. Krug, D. Labregere, A. N. Beni, R. J. Pawar, A. Sbai, S. G. Thomas, L. Trenty, and L. L. Wei. A benchmark study on problems related to CO_2 storage in geologic formations. *Computational Geosciences*, 13(4):409–434, 2009.

[13] K.H. Coats. IMPES stability: Selection of stable timesteps. *SPE Journal*, 8:181–187, 2003.

[14] B. Court, K.W. Bandilla, M.A. Celia, A. Janzen, M. Dobossy, and J.M. Nordbotten. Applicability of vertical-equilibrium and sharp-interface assumptions in CO_2 sequestration modeling. *Int J Greenh Gas Control*, 10:134–147, 2012.

[15] Z. Dai, R. Middleton, H. Viswanathan, J. Fessenden-Rahn, J. Bauman, R. Pawar, and B. McPherson. An integrated framework for optimizing CO_2 sequestration and enhanced oil recovery. *Environmental Science and Technology Letters*, 1(1):49–54, 2013.

[16] H. Deng, P.H. Stauffer, Z. Dai, Z. Jiao, and R.C. Surdam. Simulation of industrial-scale CO_2 storage: Multi-scale heterogeneity and its impacts on storage capacity, injectivity and leakage. *Int. J. Greenhouse Gas Control*, 10:397–48, 2012.

[17] Norwegian Petroleum Directorate. CO_2 storage atlas North Sea. `http://www.npd.no/en/Publications/Reports/CO2-Storage-Atlas-/`, December 2011.

[18] G.T. Eigestad, H.K. Dahle, B. Hellevang, F. Riis, W.T. Johansen, and E. Øian. Geological modeling and simulation of CO_2 injection in the Johansen formation. *Computers and Geoscience*, 13(4):435–450, 2009.

[19] M. Ertsås. Vertically integrated models of CO_2 migration: GPU accelerated simulations. Master's thesis, University of Oslo, 2011.

[20] S.E. Gasda, J.M. Nordbotten, and M.A. Celia. Vertical equilibrium with sub-scale analytical methods for geological CO_2 sequestration. *Special issue of Computational Geosciences*, 13:469–481, 2009.

[21] S.E. Gasda, J.M. Nordbotten, and M.A. Celia. Vertically-averaged approaches for CO_2 injection with solubility trapping. *Water Resources Research*, 2011.

[22] S.E. Gasda, J.M. Nordbotten, and M.A. Celia. Application of simplified models to CO_2 migration and immobilization in large-scale geological systems. *International Journal of Greenhouse Gas Control*, 9:72–84, 2012.

[23] W.G. Gray, P.A. Herrera, S.E. Gasda, and H.K. Dahle. Derivation of vertical equilibrium models for CO_2 migration from pore scale equations. *International Journal of Numerical Analysis and Modelling*, 9(2):745–776, 2012.

[24] C. H. Pentland, R. El-Maghravy, S. Iglauer, and M. J. Blunt. Measurements of the capillary trapping of super-critical carbon dioxide in berea sandstone. *Geophysical Research Letters*, 38, 2011.

[25] T.R. Hagen, M.O. Henriksen, J.M. Hjelmevik, and K.-A. Lie. How to solve systems of conservation laws numerically using the graphics processor as a high-performance computational engine. In *Geometric Modelling, Numerical Simulation and Optimization*, pages 211–264. Springer, 2007.

[26] E.K. Halland, W.T. Johansen, and F. Riis. CO_2 Storage Atlas: Norwegian North Sea. `http://www.npd.no/global/norsk/3-publikasjoner/rapporter/pdf/co2-atlas-lav.pdf`, 2011.

[27] M. Harris. *Optimizing Parallel Reduction in CUDA*. NVIDIA Developer Technology.

[28] M. Harris, S. Sengupta, and J.D. Owens. *GPU Gems 3*. Addision-Wesley Professional, first edition, 2007.

[29] C.K. Ho and S.W. Webb. *Gas Transport in Porous Media.* Springer Science and Business Media, 2006.

[30] R. Juanes, E.J. Spiteri, F.M. Orr Jr., and M.J. Blunt. Impact of relative permeability hysteresis on geological CO_2 storage. *Water Resources Research*, 42(12), 2006.

[31] J. Kou and S. Sun. On interative IMPES formulation for two-phase flow with capillarity in heterogeneous porous media. *International Journal of Numerical Analysis and Modeling, Series B*, 1:20–40, 2004.

[32] K.S. Lackner. Climate change: A guide to CO_2 sequestration. *Science*, 300:1677–1678, 2003.

[33] M.C. Levrett. Capillary behaviour in porous solids. *Transactions of the AIME*, 142:159–172, 1941.

[34] K. Li. Generalized capillary pressure and relative permeability model inferred from fractal characterization of porous media. In *Society of Petroleum Engineers, SPE Annual technical conference, Houston, Texas*, September 2004.

[35] I.S. Ligaarden and H.M. Nilsen. Numerical aspects of using vertical equilibrium models for simulating CO_2 sequestration. In *Proceedings of ECMOR XII-12th European Conference on the Mathematics of Oil Recovery, EAGE*, Oxford, UK, 2010.

[36] E. Lindeberg, J.-F. Vuillaume, and A. Ghaderi. Determination of the CO_2 storage capacity of the Utsira formation. *Energy Procedia*, 1(1):2777–2784, 2009.

[37] J.C. Martin. Some mathematical aspects of two phase flow with application to flooding and gravity segregation. *Prod. Monthly*, 22(6):22–35, 1958.

[38] K. Murawski. *Analytical and Numerical Methods for Wave Propagation in Fluid Media*. World Scientific, 2002.

[39] J.R. Natvig, T.R. Hagen, and K.-A. Lie. Solving the Euler equations on graphics processing units. In *Proceedings of the 6th international conference on computational science*, pages 220–227. Springer, 2006.

[40] H.M. Nilsen, P.A. Herrera, M. Ashraf, I.S. Ligaarden, M. Iding, C. Hermanrud, K.-A. Lie, J.M. Nordbotten, H.K. Dahle, and E. Keilegavlen. Field-case simulation of CO_2-plume migration using vertical-equilibrium models. In *Proceedings of GHGT10*, 2010.

[41] H.M. Nilsen, K.-A. Lie, and O. Andersen. `MRST-co2lab`: Fully-implicit simulation of vertical-equilibrium models with hysteresis and capillary fringe, submitted.

[42] H.M. Nilsen, K.-A. Lie, and O. Andersen. `MRST-co2lab`: sharp-interface models for fast estimation of trapping capacity, submitted.

[43] H.M. Nilsen, A.R. Syversveen, K.-A. Lie, J. Tveranger, and J.M. Nordbotten. Impact of top-surface morphology on CO_2 storage capacity. *Int. J. of Greenhouse Gas Control*, 11(0):221–235, 2012.

[44] J.M. Nordbotten and M. Celia. *Geological Storage of CO_2*. John Wiley and Sons, 2012.

[45] J.M. Nordbotten and H.K. Dahle. Impact of the capillary fringe in vertically integrated models for CO_2 storage. *Water Resources Research*, 47(2), 2011.

[46] NVIDIA. *CUDA C Programming Guide*, 6th edition, 2014.

[47] NVIDIA. CUDA samples. `http://docs.nvidia.com/cuda/cuda-samples/`, August 2014.

[48] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–25. 2005.

[49] E.K. Prestegård. A GPU accelerated simulator for CO_2 storage. Master's thesis, Norwegian University of Science and Technology, 2014.

[50] SINTEF. The Johansen Data Set. `http://www.sintef.no/Projectweb/MatMorA/Downloads/Johansen/`, May 2009.

[51] A.R. Syversveen, H.M. Nilsen, K.-A. Lie, J. Tveranger, and P. Abrahamsen. A study on how top-surface morphology influences the CO_2 storage capacity. In *Geostatistics*. Springer, 2012.

[52] M.L. Sætra. Sparse grid shallow water simulations on GPUs. In *Proceedings of ENUMATH 2011*, Leicester, UK, 2012.

[53] Z. Wei, B. Jang, Y. Zhang, and Y.Jia. Parallelizing alternating direction implicit solver on GPUs. *Procedia Computer Science*, 18:389–398, 2013.