Non-isothermal flow in porous media and reactive transport

Alina Yapparova

Department of Petroleum Engineering
Chair of Reservoir Engineering

Montanuniversität Leoben

A thesis submitted for the degree of

Doktor der montanistischen Wissenschaften

Leoben 2016

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only the literature cited in this volume.

Name:

Signed:

Date:

In memory of my mother and grandmother,
who were always proud of me

Acknowledgements

I would like to thank my supervisors Stephan Matthäi, Fiona Whitaker and Dmitrii Kulik for their guidance, patience and support. I want to thank Tatyana Gabellone for a long and productive collaboration, for the great times in Bristol, and that I could learn so much about dolomites from her.

The biggest thank you goes to my best friend Roman for being there for me every single time, helping me look for bugs in the code, for our discussions about equations and source terms, and for being my friend when I needed it the most. Without you this work wouldn't be possible.

Thank you, my dear Adrian for your love, that helped me to go through this difficult time, and for lots of proofreading (including this).

I want to thank my dear friend Katharina Flöck for her rationality, for helping me integrate in Austria, for teaching me proper German, for being my friend. Big thank you to Sarah Kettner for all the incredible moments on stage and after, for your enormous love and support. Your choir has been my happy place for the last 3.5 years.

My sincere thanks to my aunt Adelya, who has always believed in me and supported me in all my crazy dreams (including this one).

Thanks to my dearest colleagues from the Department of Reservoir Engineering, my office neighbours Philipp Lang for helping me learn CSMP and Georg Seidl for all the food in the office and for your big heart. Thanks to Julian Mindel for many skiing (haha) weekends in Präbichl, for many meshes, for remaining a friend no matter what. Thanks to my office neighbour Ina Hadziavdic for the nicest time in the office, for all the healthy cakes and our chitchats.

Thanks to Professor Gerhard Thonhauser for extending my contract and helping me in time of need. Huge thanks to our former secretary Claudia Kaiser for bringing me to Leoben in the first place, for taking care of all

my documents and always helping with any questions. Thanks to our other secretaries Bettina Matzer and Patricia Haberl for their work.

Thanks to Professor Oleg Kiselev for our long discussions about differential equations.

Last but not least, thanks to my father and my sister for their love.

The first part of this thesis was sponsored by Österreichische Forschungsförderungsgesellschaft (FFG) and Advanced Drilling Solutions, project GeoWSP [PNr.: 834533] within the research and technology program "Neue Energien 2020" of the "Klima und Energiefonds". The second part of this work was sponsored by BG Group, Chevron, Petrobras, Saudi Aramco and Wintershall in the frame of the University of Bristol ITF (Industry Technology Facilitator) project IRT-MODE.

Abstract

This thesis presents the mathematical models, numerical solution methods and simulation examples of non-isothermal single phase flow in porous media and reactive transport.

The Pressure-Temperature-Enthalpy finite element – finite volume scheme for the single phase flow in porous media and heat transport was implemented in CSMP++ software library and applied to the simulation of an underground hot water storage. Application of this scheme to the energy storage simulations is preferable to the classic Boussinesq approximation, as it uses a full equation of state/viscosity treatment for water.

In order to study the controls of hydrothermal dolomitisation by means of reactive transport modelling (RTM) with mineral dissolution/precipitation kinetics and on realistic geometries, the new CSMP++GEM coupled code was developed, tested and benchmarked against TOUGHREACT.

The prototype implementation of the RTM simulator used the Law of Mass Action approach for the chemical equilibrium calculations, but was subsequently replaced by the Gibbs Energy Minimisation method due to its numerous advantages.

The new coupled code uses a mass conservative transport scheme, an accurate equation of state for the saline water and a feedback on the fluid properties from chemical reactions, taking into account the alteration of porosity and permeability due to mineral dissolution/precipitation. Unstructured grids and explicit faults/fractures representation allow for the RTM simulations of fault-controlled dolomitisation.

CSMP++GEM was used to simulate the hydrothermal dolomite formation at the Benicassim outcrop analogue and was able to reproduce major features of the dolomitisation process.

Kurzfassung

Diese Thesis beschäftigt sich mit mathematischen Modellen, numerischen Lösungsverfahren und Simulationen des nicht-isothermischen Einphasenflusses in porösen Medien und Reaktions-Transportprozessen.

Die Druck-Temperatur-Enthalpie Finite Elemente – Finite Volumen Methode für den Einphasenfluss und Wärmetransport in porösen Medien wurde in der CSMP++ Software Library implementiert und erfolgreich zur Simulation eines unterirdischen Heißwasserspeichers eingesetzt. Zu Energiespeichersimulationen bietet diese Methode wichtige Vorteile gegenüber der klassischen Boussinesq Approximation da sie eine volle Zustandsgleichung von Wasser verwendent sowie volle Viskositätsbehandlung liefert.

Um die Kontrollparameter der hydrothermischen Dolomitisierung mittels Reaktions-Transport-Simulationen und auf realistischen Geometrien zu untersuchen wurde der neue gekoppelte CSMP++GEM Code entwickelt, getestet, und mit TOUGHREACT verglichen.

Die Prototypimplementierung des RTM Simulators nutzt das Massenwirkungsgesetz für die chemischen Gleichgewichtsrechnungen. Dieser Ansatz wurde jedoch aufgrund zahlreicher Vorteile durch die Gibbs Energie Minimierungsmethode ersetzt.

Der neue gekoppelte Code benutzt ein massenerhaltendes Transportsystem, eine genaue Zustandsgleichung für Salzwasser und Feedback zu den Flüssigkeitseigenschaften der chemischen Reaktionen, unter Berücksichtigung von Änderung der Porösität und der Permeabilität mit Auflösung und Abscheidung der Mineralien. Unstrukturierte Gitter und die explizite Darstellung von Verwerfungen und Brüchen ermöglichen RTM Simulationen von verwerfungsgesteuerter Dolomitisierung.

CSMP++GEM wurde benutzt um die hydro-thermische Dolomitisierung am Beispiel des Benicàssim Aufschlusses zu simulieren und war in der Lage, die Hauptauswirkungen des Dolomitisierungsprozesses zu reproduzieren.

Contents

[1] published as: Yapparova, A., Matthäi, S.K., Driesner, T.: Realistic simulation of an aquifer thermal energy storage: Effects of injection temperature, well placement and groundwater flow. Energy, 76, 1011-1018 (2014)

List of Figures

List of Tables

Nomenclature

β_f	fluid compressibility $[Pa^{-1}]$
β_r	rock compressibility $[Pa^{-1}]$
β_t	total system compressibility $[Pa^{-1}]$
β_ϕ	pore compressibility $[Pa^{-1}]$
Δt	time step
ϵ	exergy $[J]$
γ_i	activity coefficient of j-th DC
κ	rate constant $[mol/m^2/s]$
κ^o	rate constant at reference temperature 25 °C $[mol/m^2/s]$
Λ	Arrhenius parameter
$\mathbf{g}$	gravitational acceleration vector $[m/s^2]$
$\mathbf{J}$	component flux $[mol/l \cdot m/s]$
$\mathbf{v}$	Darcy velocity $[m/s]$
μ_f	fluid dynamic viscosity $[Pa \cdot s]$
μ_j	normalized chemical potential of the j-th DC
Ω_k	stability index of the k-th phase
$\overline{n}_j^{(x)}$	upper AMR for the j-th DC
ϕ	porosity $[-]$
ρ_f	fluid density $[kg/m^3]$

ρ_r rock density $[kg/m^3]$

$\underline{n}_j^{(x)}$ lower AMR for the j-th DC

$\widehat{n}$ GEM problem primal solution vector

$\widehat{q}$ vector of Lagrange multipliers

$\widehat{u}$ GEM problem dual solution vector

$A_{k,t}$ surface area of the k-th solid phase at time t $[m^2]$

$A_{S,k}$ specific surface area of the k-th solid phase $[m^2/kg]$

c_i aqueous concentration of the i-th IC $[mol/l]$

c_{pf} fluid specific heat capacity $[J/kg \cdot K]$

c_{pr} rock specific heat capacity $[J/kg \cdot K]$

$CaCO_3$ calcite

$CaMg(CO_3)_2$ dolomite

D diffusion-dispersion coefficient $[m^2/s]$

E energy $[J]$

E_a activation energy $[J/mol]$

G total Gibbs energy of the system $[J]$

g_j^o standard chemical potential (Gibbs energy per mole of j-th DC) $[J/mol]$

h_f specific fluid enthalpy $[J/kg]$

K thermal conductivity $[W/m \cdot K]$

k permeability $[m^2]$

m_f fluid mass per pore volume $[kg/m^3]$

M_i molar mass of the i-th IC $[kg/mol]$

m_j molality (moles per kilogram of water-solvent of the j-th species) $[mol/kg_w]$

$n^{(\phi)}$ vector of equilibrium phase amounts

$n^{(b)}$ bulk composition vector, $n(N)$ components

$n^{(x)}$ equilibrium speciation vector, $n(L)$ components

$n_{k,t}$ mineral mole amount of the k-th mineral at time t $[mol]$

p pressure $[Pa]$

Q source/sink term for the mass exchange between the aqueous and solid phases $[kg/m^3/s]$

q fluid mass source term $[kg/s]$

q_i source/sink term, accounting for mineral dissolution/precipitation of the i-th IC $[mol/l/s]$

q_v fluid volume injection/production rate $[m^3/s]$

q_{TX} source/sink term, accounting for the temperature and salinity induced solution density change at constant pressure $[kg/m^3/s]$

$R_{k,t}$ net kinetic rate of the k-th mineral at time t $[mol/m^2/s]$

T temperature $[K]$

t time $[s]$

T_0 reference temperature $[K]$

X solution salinity $[-]$

x_j mole fraction of the j-th species

AMR Additional metastability restrictions

ATES Aquifer Thermal Energy Storage

CFL Courant-Friedrichs-Lewy condition

CHPU Combined Heat and Power Unit

DC Dependent Component

GEM Gibbs energy minimization

IC Independent Component

KKT Karush-Kuhn-Tucker conditions

LMA Law of Mass Action

RTM Reactive Transport Modelling

SIA Sequential Iterative Approach

SNIA Sequential Non-Iterative Approach

Chapter 1

Introduction

The object of this work was to contribute to our understanding of complex physical systems that include single phase flow in porous media, heat transport and chemical reactions. The first goal was to accurately simulate the flow and heat transport in an underground hot water storage in three dimensions. The second goal was to study the controls of the dolomitisation process on realistic geometries.

In the frame of this thesis two computational tools were developed. The first one is the implementation of the Pressure-Temperature-Enthalpy scheme for single phase flow and heat transport. The second one is the reactive transport modelling code CSMP++GEM.

These two codes are both based on the CSMP++ software library and are aiming for realistic representation of fluid properties and model geometry. Both use the finite element – finite volume method for the solution of partial differential equations on unstructured grids, and accurate equations of state for pure and saline waters.

The Pressure-Temperature-Enthalpy scheme was applied to the full working cycle simulations of a pilot underground hot water storage project in Upper Austria. Alternative energy projects are often economically marginal and therefore require precise numerical methods to assess their efficiency.

The software development of the second project was driven by the necessity to realistically simulate hydrothermal dolomitisation. More than 60% of the world's oil and 40% of the world's gas reserves are stored in carbonate rocks, more than half of these rocks are fully or partially dolomitised. Understanding the distribution and quality of dolomite bodies is a key to the efficient recovery of oil and gas carbonate reservoirs and reactive transport modelling is a viable tool for these studies.

Hydrothermal dolomites form from fluids elevated in temperature compared to the host rock. These fluids are usually transported along faults and/or fractures, that

have a complex geometry that can not be accurately captured with standard reactive transport modelling tools that use corner-point grids.

The purpose of the new CSMP++GEM reactive transport code development was to combine the ability to solve flow and transport equations on unstructured grids with precise numerical methods for chemical speciation calculation including the kinetics of dolomite precipitation.

The Law of Mass Action (LMA) method was chosen for the prototype implementation and with this code the ability to perform reactive transport simulations on unstructured grids with sophisticated geometry was demonstrated.

However, due to the fact that the LMA method has to perform multiple iterations in order to find a stable mineral assemblage, it was superseded by the Gibbs Energy Minimisation (GEM) method and the coupled CSMP++GEM code was created. The GEMS3K standalone code that was used in the coupling has the temperature and pressure dependence of the chemical reaction rates encoded in the thermodynamic database and is able to compute (meta)stable chemical speciation in complex heterogeneous systems. The new code was benchmarked against TOUGHREACT and applied to the RTM simulations of dolomitisation process on a real case study (Benicàssim outcrop, Spain).

This thesis is organized as follows. The second chapter is devoted to the simulation of the non-isothermal single phase flow in porous media with application to underground hot water storage. The third chapter presents the results of the dolomitisation simulations with the standard LMA approach. The forth chapter gives a detailed description of the new CSMP++GEM coupled code for reactive transport simulations. In the last chapter the modelling results of the Benicàssim case study are presented. At the end of this thesis general conclusions are outlined.

Other outcomes of this work are the publications and presentations listed below.

Publications

1. A. Yapparova, S.K. Matthäi, T. Driesner, Realistic simulation of an aquifer thermal energy storage: Effects of injection temperature, well placement and groundwater flow. Energy 76, 1011-1018 (2014)

Presentations

1. A. Yapparova, J.E. Mindel, M. Maierhofer, S.K. Matthäi, Geothermal Simulation Applied to the Optimization of Underground Energy Storage Systems,

Second Sustainable Earth Science Conference and Exhibition (SES 2013), 30 September – 4 October 2013, Pau, France

2. A. Yapparova, Reactive transport modelling of carbonate diagenesis on unstructured grids, International Conference on Numerical and Mathematical Modeling of Flow and Transport in Porous Media, 29 September 2014 – 3 October 2014, Dubrovnik, Croatia

3. A. Yapparova, T. Gabellone, F. Whitaker, D.A. Kulik, S.K. Matthäi, A new CSMP++GEM reactive transport code, Goldschmidt 2015, 16 August 2015 - 21 August 2015, Prague, Czech Republic

Chapter 2

Realistic simulation of an aquifer thermal energy storage: Effects of injection temperature, well placement and groundwater flow [1]

To optimize the behaviour of an Aquifer Thermal Energy Storage (ATES), to estimate its efficiency and to identify the optimal well locations, the planned installation was simulated with a FE-FV simulator with realistic water properties, created on the basis of the CSMP++ software library.

Simulation results show that storage efficiency increases with the distance between injection and production wells and decreases with increasing injection temperature. Results also support a storage design where the storage wells are placed near the walls. Groundwater flow does not affect storage efficiency significantly, proving that the double concrete walls act as a sufficient thermal insulator.

The full operation cycle of an ATES with an optimal well placement was simulated: 120 days of hot water injection (charging), 60 days of production (discharging). The predicted energy and exergy efficiency are 35.6% and 27.4%, respectively. The storage supplies between ∼300 kW and ∼120 kW of thermal energy the first 60 days of hot water production.

[1] published as: Yapparova, A., Matthäi, S.K., Driesner, T.: Realistic simulation of an aquifer thermal energy storage: Effects of injection temperature, well placement and groundwater flow. Energy, 76, 1011-1018 (2014)

2.1 Introduction

Among European countries, Austria has one of the highest shares of renewable energy in final energy consumption (30.9% in 2011) [1]. The engineering target that this study contributes to is to design a unique innovative storage system for a waste energy utilization. The pilot site, with near surface geology suitable for an ATES construction (shallow permeable aquifer and an aquiclude in reachable depth) for corn drying is located in Upper Austria.

Aquifer thermal energy storage is usually installed as a part of a heating/cooling system of office buildings [2], supermarkets [3], hospitals [4], including a well-known example of the cooling system of the German Parliament Buildings [5]. Our application, however, has some unique characteristics. First, the aquifer storage is designed to store excess heat produced by a combined heat and power unit (CHPU) of a biogas plant. Second, the temperature range that the storage operates is much higher (hot water temperature of 85°C and unperturbed groundwater temperature of 12°C) than in the building conditioning applications (10-15°C difference). Third, the heat is usually stored in the aquifer layer sealed by impermeable clay layers on the top and bottom but unbounded from the sides and therefore can be transported by groundwater flow, while in our case the storage volume is enclosed inside two vertical concrete walls.

Numerical simulation is a viable tool for understanding and predicting the behaviour of complex systems. Prior to the storage construction the simulator can be used to assess the feasibility of the project, as well as to help in storage design and optimization of well locations. Later on the simulator will be applied in monitoring and well steering.

Like any other modelling area, heat transfer modelling has a trend of going from simple approximations to more and more sophisticated. The classic Boussinesq approximation assumes that fluid density depends linearly on temperature only. All other properties are constant. While this approximation was used about 20-25 years ago for modelling hot water storage in aquifers due to its simplicity and low computational costs [6, 7, 8], advances in computer hardware over the last 2 decades now permit to perform simulations on 3d unstructured grid models with half a million cells and realistic fluid properties on a standard PC within reasonable time. This is the first study where the behaviour of an ATES was modelled with a highly accurate pressure-enthalpy-temperature finite element – finite volume scheme and a full equation of state/viscosity treatment for water. Unlike in the commonly used Boussinesq

approximation, in our simulations density, viscosity, heat capacity, enthalpy and compressibility of water are re-computed every time step. The finite element – finite volume method supports the use of unstructured grids, allowing precise representation of the circular (cylindrical) geometry common to energy storages. In contrast to that, the finite difference method that is widely used in ATES modelling [2, 9] requires structured grids, resulting in staircase-like representations of curves and polygonal (pixelated) heat advection fronts.

Several studies provided sensitivity analyses with respect to different parameters. Parameters that can influence the performance of the geothermal energy storage include rock properties, well location and operation conditions and groundwater flow around the storage. Results of the shallow heat injection and storage experiment presented in [10] state that the most sensitive parameter is the thermal conductivity of the solid followed by the porosity, heat capacity of the solid and the longitudinal dispersivity. Kim et al. [11] investigated the influence of the distance between injection and production wells, injection rate and hydraulic conductivity on the performance of ATES. According to their results the performance of an ATES system primarily depends on the thermal interference between warm and cold thermal energy stored in an aquifer that grows as the injector-producer distance decreases. In our study we performed similar tests – for the closest and the farthest well placements – but for an energy storage in a closed volume, as well as sensitivity runs for varying injection temperature.

In many ATES modelling studies groundwater flow has been neglected, which has been justified by the low flow velocity. To quantify its influence on the storage efficiency we present numerical experiments and compare results with and without groundwater flow in the same way that it was done in [12] for a geothermal heat exchanger and in [9] for a conceptual model of an ATES system.

In our study we investigated the influence of well placement and operation conditions and groundwater flow on the performance of an aquifer thermal energy storage. We considered two different well placements and a variation in the injection temperature. It is of common sense to inject hot fluid in the upper part of the storage and produce cold fluid from the bottom to support thermal stratification, but the optimal position of a "hot" well – in the middle of the storage or closer to the wall – has remained an open question that we address in our study. The second question that we had in mind was if groundwater flow can be neglected in simulations of this particular type of storages.

The paper is structured as follows. First, we give a project overview with a brief description of the biogas plant facilities and the corn drying system. Second, we describe the simulation methodology that was used, starting with the mathematical formulation, numerical scheme and details on software and benchmarking performed. Third, geology and geometry of the storage as well as rock and fluid properties are presented, an explanation of well operation modes is given and the overall numerical experimental setup is established. In the last section we present results and conclusions.

2.2 Project overview

The study represents a performance analysis of a planned biogas plant located in Upper Austria. Biogas from a fermenter is converted to electricity and heat in the combined heat and power unit (CHPU), which generates 330 kW of electrical and 500 kW of thermal power. Thermal energy is stored in water heated up to $T = 85\,^{\circ}\mathrm{C}$ while cooling the CHPU, is used for domestic and fermenter heating and for drying of corn, crops and woodchips.

During the summer months a certain amount of thermal energy is not used. ATES connected to a biogas plant is supposed to reduce this energy waste by storing the excess energy during summer months and using it for corn drying in autumn.

Corn is dried by an air flow, that is heated consequently in two heat exchangers from the ambient temperature to $75\,^{\circ}\mathrm{C}$. In the first heat exchanger air is pre-heated to a certain temperature with hot water from the ATES. In the second heat exchanger excess heat from CHPU is utilized in order to reach the target temperature. ATES usage is aimed to increase the drying capacity.

2.3 Methodology

2.3.1 Governing equations

Single phase flow in a porous medium is usually described by Darcy's law [13]:

$$\mathbf{v} = -\frac{k}{\mu_f}(\nabla p - \rho_f \mathbf{g}), \tag{2.1}$$

here $\mathbf{v}$ is the fluid velocity, k is the permeability, μ_f is the fluid dynamic viscosity, p the pressure, ρ_f is the fluid density and $\mathbf{g}$ is the gravitational acceleration vector.

Mass balance for a single-phase fluid in a porous medium can be expressed in terms of the continuity equation [13]:

$$\frac{\partial(\phi\rho_f)}{\partial t} = -\nabla \cdot (\rho_f \mathbf{v}) + q, \tag{2.2}$$

where ϕ is the porosity, q is a fluid mass source term. Applying the chain rule for computing the total derivative of $\rho_f = \rho_f(p(t), T(t))$ we get:

$$\frac{d\rho_f}{dt} = \frac{\partial \rho_f}{\partial p}\frac{\partial p}{\partial t} + \frac{\partial \rho_f}{\partial T}\frac{\partial T}{\partial t} = \rho_f \beta_f \frac{\partial p}{\partial t} + \left.\frac{\partial \rho_f}{\partial t}\right|_{p=const}, \tag{2.3}$$

where β_f is the fluid compressibility. We express porosity change in terms of rock compressibility β_r [14]:

$$\frac{\partial \phi}{\partial t} = (1-\phi)\beta_r \frac{\partial p}{\partial t}. \tag{2.4}$$

By using equations (2.3) and (2.4) to rewrite the left-hand side of (2.2) and inserting (2.1) into its right-hand side we arrive at a transient pressure equation:

$$\rho_f((1-\phi)\beta_r + \phi\beta_f)\frac{\partial p}{\partial t} = \nabla \cdot (\rho_f \frac{k}{\mu_f}(\nabla p - \rho_f \mathbf{g})) - \phi \left.\frac{\partial \rho_f}{\partial t}\right|_{p=const} + q. \tag{2.5}$$

Energy conservation equation is written in the form [13]:

$$(\phi\rho_f c_{pf} + (1-\phi)\rho_r c_{pr})\frac{\partial T}{\partial t} = \nabla \cdot (K\nabla T) - \nabla \cdot (\mathbf{v}\rho_f h_f), \tag{2.6}$$

where T is the temperature, K the thermal conductivity, c_{pf} and c_{pr} are the fluid and rock specific heat capacities, respectively; ρ_r is the rock density, h_f is the fluid's specific enthalpy.

2.3.2 Numerical scheme

The resulting system of equations (2.1), (2.2), (2.5), (2.6) is solved using a hybrid finite element - finite volume method [14] implemented within the framework of CSMP++ (Complex Systems Modelling Platform) software library [15]. All fluid properties are stored in a lookup table to cut computational cost. The accurate, fast and robust solver SAMG from Fraunhofer Institute is used to solve the arising systems of linear algebraic equations [16].

The energy conservation equation (2.6) is solved by operator splitting into the diffusive and advective parts

$$c_{pt}\frac{\partial T_{diff}}{\partial t} = \nabla \cdot (K\nabla T), \tag{2.7}$$

$$c_{pt}\frac{\partial T_{adv}}{\partial t} = -\nabla \cdot (\mathbf{v}\rho_f h_f), \tag{2.8}$$

where c_{pt} is the total heat capacity, $c_{pt} = (\phi\rho_f c_{pf} + (1-\phi)\rho_r c_{pr})$.

Following the approach described in [17], we re-write equation (2.8) in terms of specific fluid enthalpy h_f, using the relation $dh_f = c_{pf}dT$ on the left-hand side, and neglecting the rock-related part:

$$\phi\rho_f \frac{\partial h_f}{\partial t} = -\nabla \cdot (\mathbf{v}\rho_f h_f). \tag{2.9}$$

We introduce the additional property fluid mass per pore volume m_f with units of density $kg \cdot m^{-3}$. This is necessary for calculation of the term $\phi \left.\frac{\partial \rho_f}{\partial t}\right|_{p=const}$ in the pressure equation (2.5), that describes fluid expansion/compression at constant pressure due to changes in temperature only. In the numerical solution, we substitute ρ_f with m_f in the left-hand sides of equations (2.2), (2.9) and (2.5) and re-write our system of governing equations in the following form:

$$\phi\frac{\partial m_f}{\partial t} = -\nabla \cdot (\mathbf{v}\rho_f), \tag{2.10}$$

$$c_{pt}\frac{\partial T_{diff}}{\partial t} = \nabla \cdot (K\nabla T), \tag{2.11}$$

$$\phi m_f \frac{\partial h_f}{\partial t} = -\nabla \cdot (\mathbf{v}\rho_f h_f), \tag{2.12}$$

$$dT = dT_{diff} + dT_{adv} = dT_{diff} + \frac{dh_f}{c_{pf}}, \tag{2.13}$$

$$m_f\beta_t\frac{\partial p}{\partial t} = \nabla \cdot (\rho_f \frac{k}{\mu_f}(\nabla p - \rho_f \mathbf{g})) - \phi \left.\frac{\partial \rho_f}{\partial t}\right|_{p=const} + q, \tag{2.14}$$

where $\beta_t = (1-\phi)\beta_r + \phi\beta_f$ is the total system compressibility.

It is important to start transient computations from an initial equilibrated state. Therefore, the steady state pressure distribution is iteratively computed with fluid properties from the lookup table. At time step t^0 we set $m_f^0 = \rho_f^0$.

Starting from m_f^n, h_f^n, T^n, p^n, the computational procedure is conducted as follows. First, equations (2.10) and (2.12) are solved using an explicit finite volume method, giving m_f^{n+1} and h_f^{n+1}. Second, the temperature diffusion equation (2.11) is solved using an implicit finite element method, to compute T_{diff}^{n+1}. As a third step, so called thermal equilibration is performed. Temperature at time step $n+1$ is computed via:

$$T^{n+1} = T_{diff}^{n+1} + \frac{h_f^{n+1} - h_f^n}{c_{pf}\Delta t},$$

new values for fluid properties ρ^{n+1}, h_f^{n+1}, μ_f^{n+1}, c_{pf}^{n+1}, β_f^{n+1} are obtained from a lookup table at T^{n+1} and p^n. This creates a discrepancy between how much fluid should be

in the finite pore volume at (T^{n+1}, p^n) and how much it is actually there due to the in-/outflow. The difference between the fluid density at the given temperature and pressure and fluid mass per pore volume after the simulated transport is equal to the numerical approximation of the derivative

$$\left.\frac{\partial \rho_f}{\partial t}\right|_{p=const} = \frac{\rho_f^{n+1} - m_f^{n+1}}{\Delta t}. \tag{2.15}$$

As a last step, eq. (2.14) with a special source term for mass correction from (2.15) is solved using the finite element method to find the pressure p^{n+1} at the next time step.

2.3.3 Assumptions

We assume instantaneous thermal equilibration between the fluid and the porous medium. The storage has homogeneous constant rock properties: porosity, permeability, thermal conductivity and specific heat capacity.

2.3.4 Benchmarking

A comparison with TOUGH2 simulator [18] was conducted (Fig. 2.1). Temperature and pressure ranges correspond to the storage operation conditions. The setup of this benchmark is the following: 2D $2000m \times 1000m$ rectangulat model, Dirichlet pressure and temperature boundary conditions (left: p=2[atm], T=80[°C], right: p=1[atm], T=50[°C]), rock properties ($k = 0.1[Da], \phi = 0.3, \rho_r = 2650[kg/m^3], cp_r = 1000[J/kg \cdot K], \lambda = 2[W/m \cdot K], \beta = 10^{-10}[Pa^-1]$), fluid properties taken from an equation of state of water [19]. The computed temperature profiles at different times are shown in Figure 2.1. Numerical solution obtained from a CSMP simulation is in a good agreement with results from TOUGH2, the maximum relative temperature difference is less than 2%.

2.3.5 Energy and exergy efficiency analysis

For efficiency calculations we follow the approach of Dincer and Rosen [20, 21]. Total input (output) energy is calculated, using:

$$E = \int_{t_0}^{t} \rho_f q_v c_{pf} (T(t) - T_0)\, \mathrm{d}t,$$

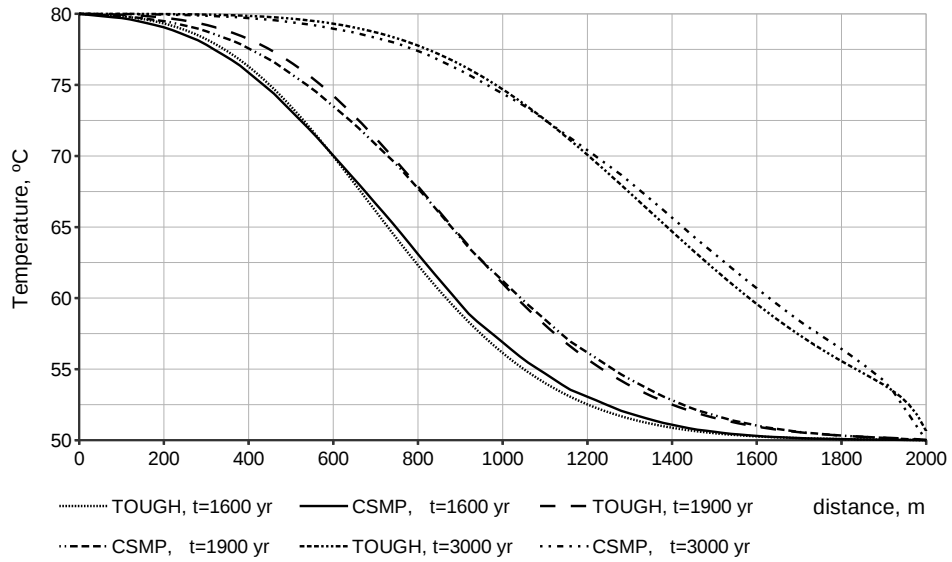

Figure 2.1: Benchmark CSMP vs. TOUGH. Temperature profile along the horizontal axis after 1600, 1900, 3000 years.

where q_v is the volume injection (production) rate, $T(t)$ — current injection (production) temperature, T_0 — reference temperature. Exergy (energy that is available to be used) is calculated via:

$$\epsilon = E - \int_{t_0}^{t} \rho_f q_v c_{pf} T_0 ln \left(\frac{T(t)}{T_0} \right) \mathrm{d}t.$$

2.4 ATES modelling

The storage system is located in the Molasse basin near St. Georgen bei Obernberg am Inn in Upper Austria. The storage cross section is shown in Figure 2.2. Above an impermeable clay layer rests a highly permeable gravel layer covered by a small impermeable loess layer. The groundwater table is it at 15 m depth. It was proposed to construct a diaphragm concrete wall in the 11 meter thick gravel layer to create a buffer zone to the natural aquifer. The storage has a cylindrical shape with a radius of 16.5 meters, total pore volume of the storage is 4045 m^3.

2.4.1 Computational model

The computational domain includes the cylinder-shaped storage with a bounding box with a side length of 70 m, oriented such that the groundwater flow occurs from left to right. The computational mesh consists of about half a million tetrahedral cells, with

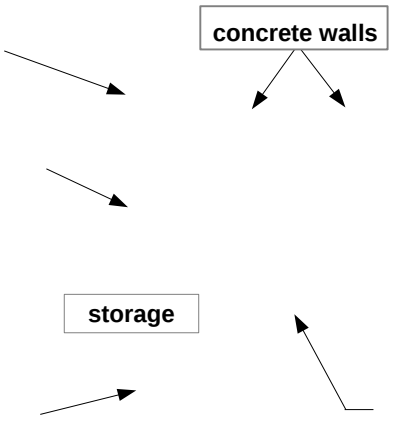

Figure 2.2: Storage geology and geometry.

Table 2.1: ATES rock properties

Rock type	$\phi, \%$	$k, [10^{-15} \times m^2]$	$\rho, [kg/m^3]$	$c_p, [J/kg \cdot K]$	$K, [W/m \cdot K]$
loess	5	1.2	2740	1500	1.15
claymarl	5	1.2	2770	1500	1.16
saturated gravel	43	12000	2650	600	2.6
unsaturated gravel	43	1.2	2350	1000	0.61
concrete walls	5	1.2	2710	1500	1.13

wells represented by line elements. The mesh is refined near the wells and storage walls. Fluid flow is simulated only in the storage region and in the groundwater region (coloured blue in Fig. 2.2), but conductive heat transport and pressure are computed in the entire model domain.

2.4.2 Rock and fluid properties

Table 2.1 summarizes rock properties based on laboratory measurements conducted by Advanced Drilling Solutions. Fluid properties are taken from the equation of state of water from [19].

2.4.3 Boundary and initial conditions. Groundwater flow

The groundwater flow direction and rate was taken from a geological digital database for Upper Austria [22]. We impose an inflow rate of $v = 1.44 \times 10^{-6} m/s$ on the left boundary in the groundwater flow region with a fixed temperature of 12 °C. Temperature at shallow depths around 10-20 m below the surface stays approximately

the same all over the year [10, 23]. We fix the temperature of the aquifer at 12 °C as an initial condition. We impose Dirichlet boundary conditions for pressure (atmospheric) and temperature (ambient) on the surface.

2.4.4 Operation modes

We consider a cyclic operation of the storage. During the charging period, water at temperature $T = 85\,°C$ is injected into a "hot" well and the same amount of water of varying temperature is produced from the "cold" well. The discharging period starts right after the charging. Water produced from the "hot" well, is cooled down to the 12 °C by going through the heat exchanger and then re-injected into the "cold" well. We study the storage behaviour with two different well placements, representing the two end member cases: maximum distance between "hot" and "cold" wells and minimum distance – with "hot" well in the middle, see Fig. 2.3. The distance between a well and the storage wall is 1.5 m, the open hole section of the well is 2 m, well radius is 0.075 m, see Fig. 2.4. "Hot" wells and the "cold" well in the first setup are operated under rate control at $q = 0.001\,m^3/s$. In the second setup, all four "cold" wells have a quarter of the "hot" well rate. Further on we refer to the first well configuration as "corner" and to the second as "middle", "gw" stands for groundwater flow.

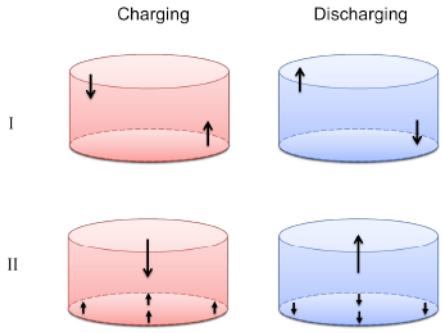

Figure 2.3: Well placements.

2.5 Results

2.5.1 Injection temperature influence

A total of 6 simulation runs were performed without groundwater flow in order to estimate the influence of injection temperature on ATES efficiency. Two well placements were considered, with injection well in the upper corner and in the middle of

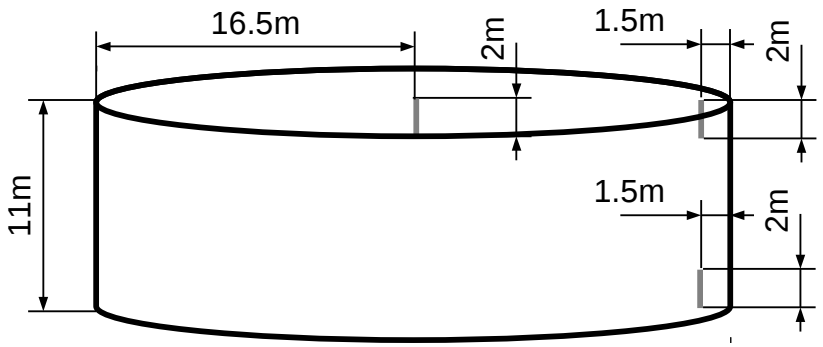

Figure 2.4: Storage dimensions. Possible well locations are displayed in lighter color.

Table 2.2: Influence of injection temperature on ATES energy and exergy efficiency

	Energy efficiency, %		Exergy efficiency, %	
	corner	middle	corner	middle
80 °C	65.3	61.9	47.7	44.2
85 °C	64.5	60.6	46.7	42.7
90 °C	63.6	59.3	45.7	41.1

the storage (Fig. 2.3). Three injection temperatures (1) $T = 80\,°\mathrm{C}$, (2) $T = 85\,°\mathrm{C}$, (3) $T = 90\,°\mathrm{C}$ were investigated. The charging for 30 days followed directly by discharging for 30 days was simulated. Figure 2.5 shows average temperatures at the production wells with respect to different injection temperatures. Table 2.2 contains the corresponding energy and exergy efficiency values.

For a fixed injection temperature efficiency is higher for the "corner" well placement, than for the "middle" one. This supports the result from [11], that thermal interference increases (and storage efficiency decreases respectively) with decreasing distance between the "hot" and the "cold" wells.

We observe that for both well placements storage efficiency slightly decreases (and heat losses increase correspondingly) with increasing injection temperature ("hot" well). The reason for that is mixing of hot and cold water inside the storage. As re-injection temperature ("cold" well) is the same (12 °C) in all cases, the energy efficiency (ratio of recovered to stored thermal energy) is influenced by the temperature at the production well only. The degree of thermal interference grows with increasing temperature difference between the "hot" and "cold" wells and efficiency decreases consequently. However, when we compare two runs with the same well placement and an injection temperature difference of 5 °C at the end of the discharging period, temperature differs by not more than $\sim 1\,°\mathrm{C}$.

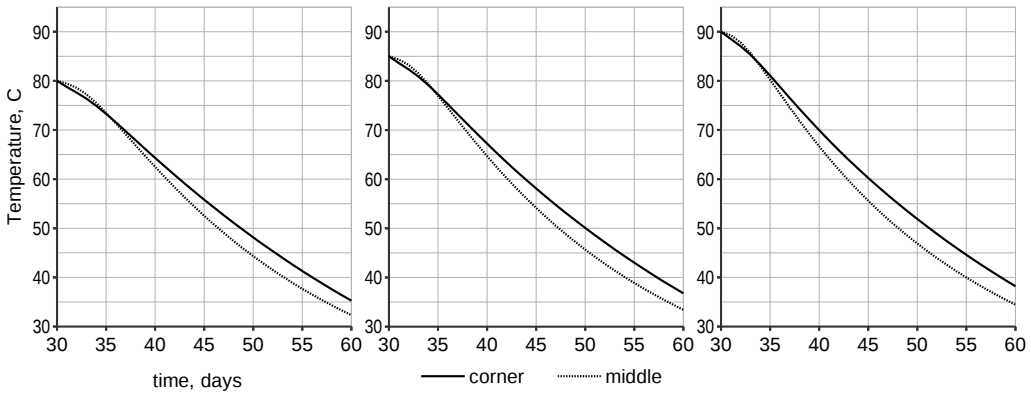

Figure 2.5: Average production well temperature during 30 days of discharging for charging temperatures of 80, 85 and 90 °C.

2.5.2 Groundwater flow influence

The results of 4 different simulations with "corner" and "middle" well placements with and without groundwater flow allow for an evaluation of its influence. In all four cases injection temperature was 85 °C and 120 days of charging were followed by 60 days of discharging. Figures 2.6 and 2.7 show vertical cross-sections of the temperature distribution in the middle of the storage for two different well placements and injection temperature of 85 °C after 120 days of charging, figures 2.8 and 2.9 – after 30 days of discharging. The velocity field in the groundwater flow region is shown in Fig. 2.10.

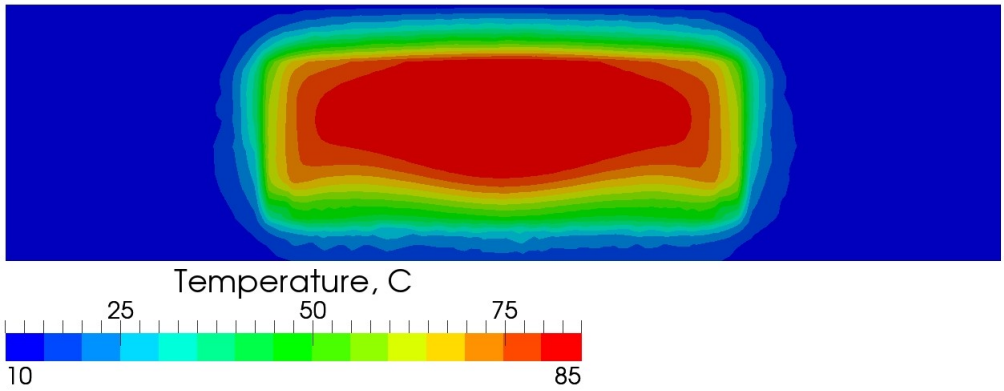

Figure 2.6: Cross-sectional temperature distribution in the ATES after 120 days of water injection for the "middle" well placement.

Comparing the temperature distributions in a horizontal cross-section through the middle of the storage, figures 2.11 and 2.12 display the temperature differences between runs with and without groundwater flow. It can be seen that groundwater

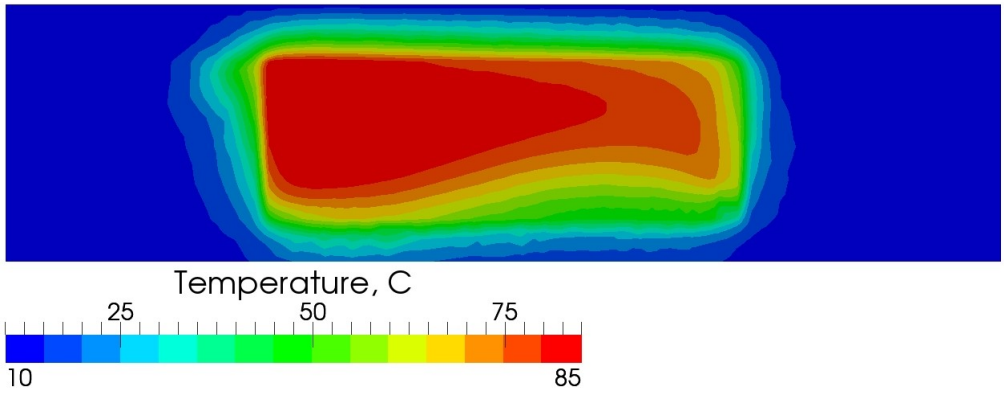

Figure 2.7: Cross-sectional temperature distribution in the ATES after 120 days of water injection for the "corner" well placement.

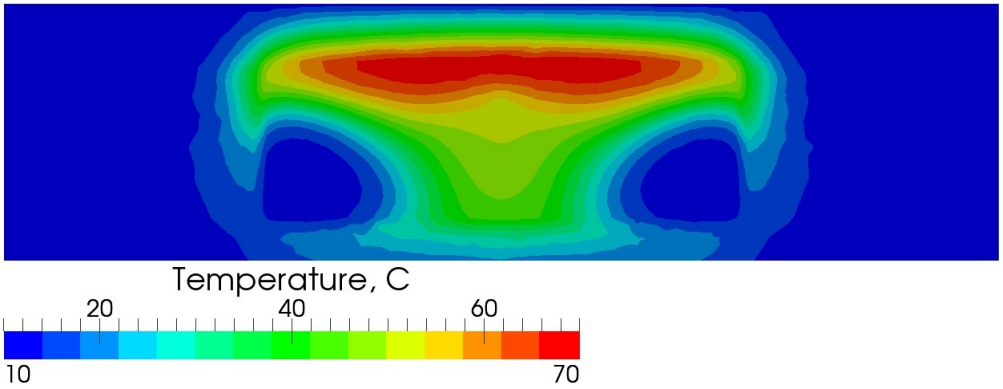

Figure 2.8: Cross-sectional temperature distribution in the ATES after 30 days of water production for the "middle" well placement.

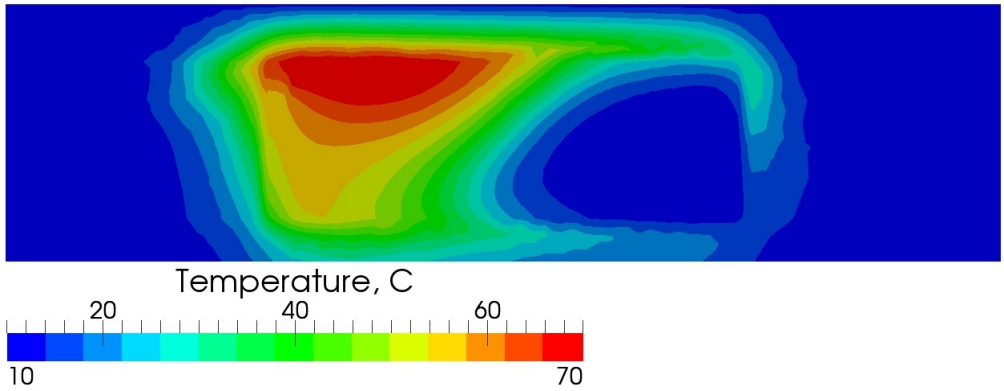

Figure 2.9: Cross-sectional temperature distribution in the ATES after 30 days of water production for the "corner" well placement.

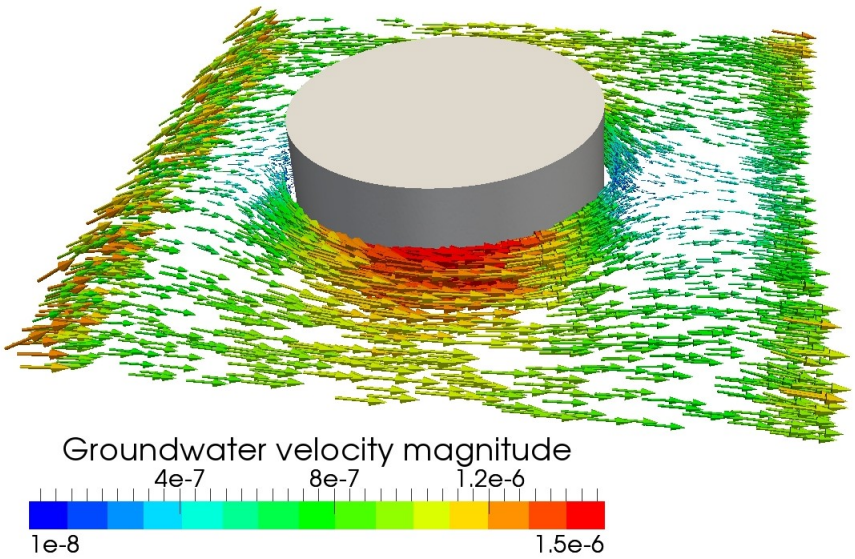

Figure 2.10: Groundwater flow velocity field around the storage.

Table 2.3: ATES energy&exergy efficiency

Case	Energy efficiency, %	Exergy efficiency, %
corner	35.9	27.9
corner gw	35.6	27.4
middle	34.3	26.2
middle gw	34.2	26.0

flow cools down the left part of the storage wall, but the temperature distribution inside the storage remains almost unaffected.

Figure 2.13 shows the average temperature at the production well and the output energy during the discharging period. In table 2.3 efficiency calculations are summarized, reference temperature of $T_0 = 12\,^{\circ}\mathrm{C}$ was assumed. The presence of the groundwater flow decreases the production temperature by less than $0.5\,^{\circ}\mathrm{C}$, having a minor effect on storage efficiency. This highlights the advantages of the storage design with double concrete walls. Reinforced concrete has a thermal conductivity more than two times smaller and a heat capacity almost three times bigger than that of the saturated gravel that comprises the aquifer. This difference in material properties makes the diaphragm wall a good insulator minimizing the heat loss due to groundwater flow.

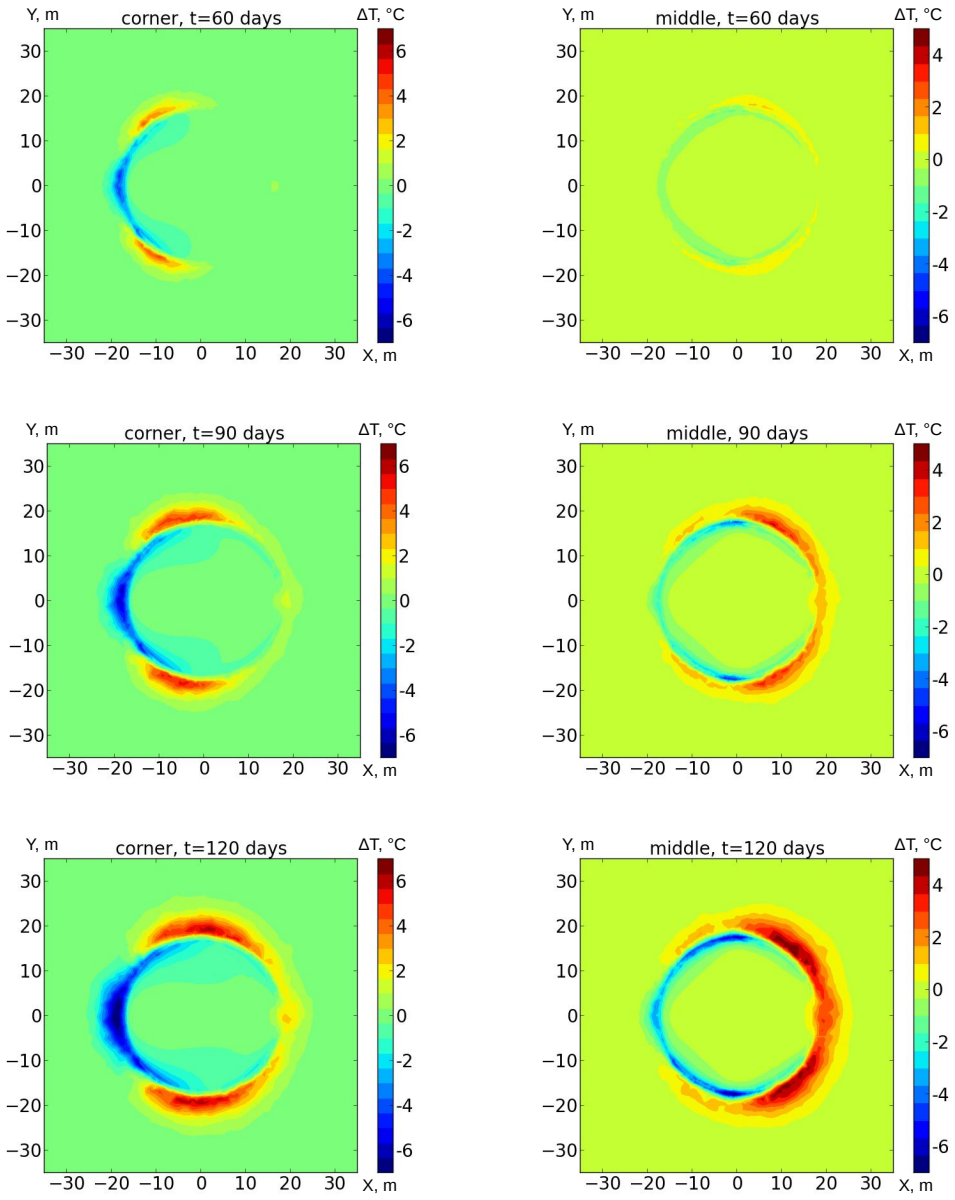

Figure 2.11: Temperature difference for simulations with and without groundwater flow ($T_{gw} - T_{no\,gw}$) after 60, 90 and 120 days of charging for "corner" (left) and "middle" (right) well placements. Horizontal cross-section.

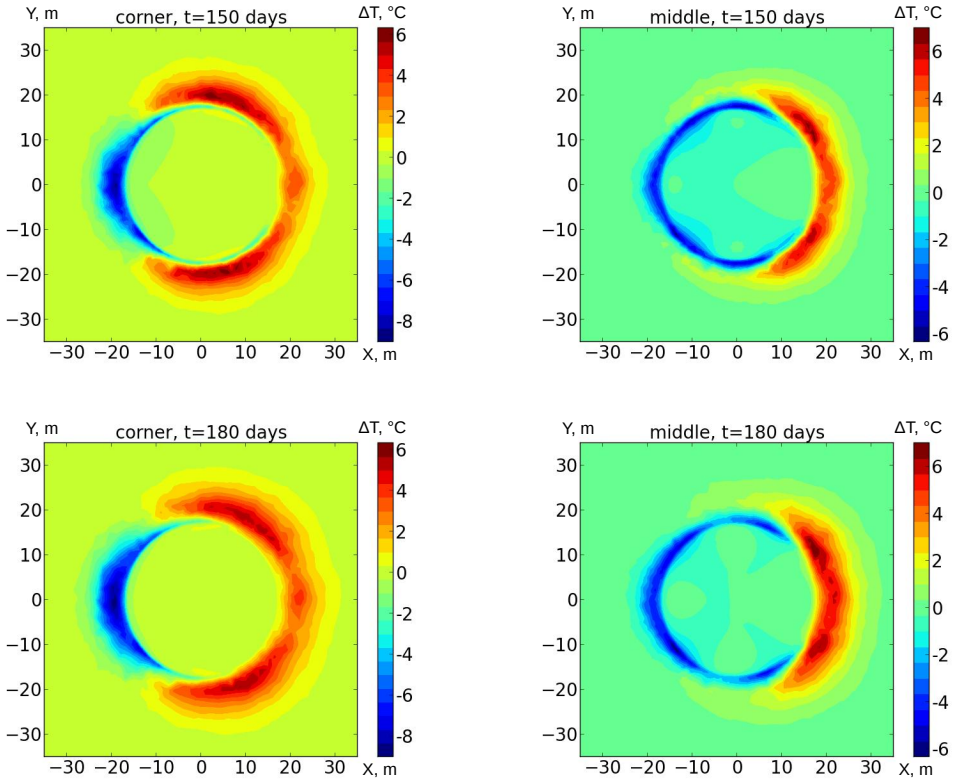

Figure 2.12: Temperature difference for simulations with and without groundwater flow ($T_{gw} - T_{no\,gw}$) after 30 and 60 days of discharging for "corner" (left) and "middle" (right) well placements. Horizontal cross-section.

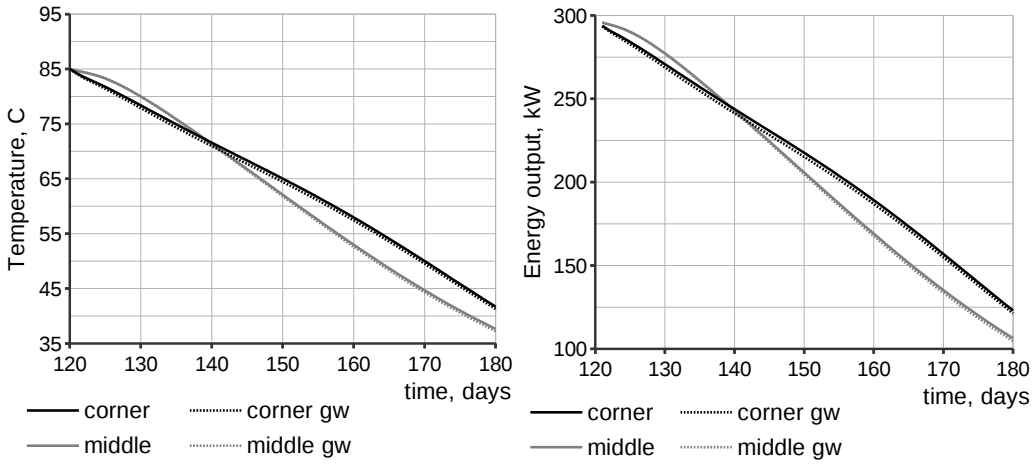

Figure 2.13: Production well temperature and output energy during ATES discharging for two different placements with and without groundwater flow.

2.6 Conclusions

Aquifer thermal energy storage with realistic water properties and geological properties from a pilot site was simulated. The simulation results will be used in an economical evaluation of the storage efficiency and for optimization of the well placement. Results presented in this paper support the decision of placing the wells near storage walls because production temperature stays higher for a longer time in this case. By placing wells as far apart from each other as possible we reduce the mixing that occurs between the hot and cold water and keep the high temperature region near injection well. We observed that within the $85 \pm 5\,^{\circ}\mathrm{C}$ injection temperature fluctuations storage efficiency shows minor changes and production temperature stays within a $\pm 2\,^{\circ}\mathrm{C}$ range. From an engineering point of view it is an important sensitivity result. Hot water that is stored in the aquifer comes from cooling the CHPU and its temperature varies with time. However our results prove that despite this unavoidable slight injection temperature variations it is possible to store and later produce sufficient amount of thermal energy. Although temperature distributions near the walls for runs with and without groundwater flow vary significantly, temperature inside the storage and well production temperature stay rather the same, which is an indicator that the double concrete wall is a sufficiently good thermal insulator. Our groundwater flow comparison runs justify the storage design and the installation of the diaphragm walls in particular. We predict ATES to be able to provide sufficient energy supply for a total of 2 months of hot water production. The main conclusion of this case study is that the overall engineering design of this storage is justified and the project is feasible. If the storage will be built there will be a unique opportunity to validate the results.

We hope that this work will encourage further highly realistic simulations of geothermal storage sites, including eos-based computations of water properties dependent on temperature and pressure, unstructured grid representations of storage geometry and new thermodynamically rigorous numerical schemes for fluid flow and heat transport.

Chapter 3

Reactive transport modelling of dolomite formation using the Law of Mass Action approach

A prototype reactive transport modelling code was implemented as part of the CSMP++ software library, which employs finite element – finite volume method for solving partial differential equations on unstructured grids. The Law of Mass Action was used to calculate the chemical equilibrium between minerals and an aqueous solution.

In this section, an equation derivation for the calcite-dolomite system is given. After that two test cases are presented: a 1D benchmark of the well-known calcite dissolution dolomite precipitation test case [24] and a 2D simulation of calcite replacement by dolomite on a cross-section model with realistic geometry.

3.1 Introduction

For more than 20 years reactive transport modelling (RTM) has been widely used to develop an understanding of hydrothermal ore deposit formation, geothermal energy production and mineral diagenetic processes like dolomite formation [25]. In our attempt to develop a reactive transport code that will be able to model the formation of dolomite, we decided to start with a simple prototype implementation using the Law of Mass Action (LMA) approach.

One of the earliest descriptions of LMA in its modern form was given by Reed in 1982 [26]. He presented the formulation of *Mass action equations* and *Mass balance equations* and introduced a solution method where the only unknowns were the so-called *component species* (that were also called *basic* or *primary species* later). The resulting system of non-linear algebraic equations was solved using the Newton-

Raphson method. This new approach had significant numerical advantages compared to methods previously used by Helgeson [27] and others. In principle, the LMA formulation in this form is still used today.

The other big step in the development of RTMs was when the chemical speciation calculations were coupled with equations for transport of solutes. Among the pioneering works is a 1994 paper by Steefel and Lasaga [28], who formulated the coupled model for transport and reactions (using Reed's formulation for equilibrium reactions) and applied it to simulate reactive flow in single phase hydrothermal systems.

One of the first thorough descriptions of different reactive transport models, including various mathematical descriptions of reaction systems and methods for coupling reactions and transport, was given by Steefel and MacQuarrie in 1996 [29] and was largely used in this work.

As stated in the recent review of RTM codes by Steefel et al. [30], many modern reactive transport codes use the LMA approach coupled with transport modules in various forms. Among the most widely used codes are PHREEQC, which is also used as a chemical module in HPx, PHT3D and one of the modules in OpenGeoSys; TOUGHREACT, HYDROCHEMGEM, CrunchFlow and MIN3P.

The Law of Mass Action approach has its limitations (which will be discussed in detail in the following chapter), but (1) LMA-based computations are much faster (when transport-chemistry operator splitting is used) than those based on the Gibbs energy minimization (GEM) method and (2) LMA is easier to implement than GEM. That is why LMA is ideal for the prototype implementation.

This chapter is structured as follows. First, I give a short decription of the solution method. Second, results of a generic 1D model of calcite dissolution – dolomite precipitation benchmark are presented. In the end, I show the results of a simulation of dolomitisation process in a 2D realistic geological cross-section.

3.2 Methodology

Reactive transport modelling comprises solution of equations describing fluid flow in porous media and equations describing chemical reactions between the aqueous species dissolved in the fluid and the minerals in the solid rock. There are three possible ways to solve the resulting system of equations [31]: to decouple chemical reactions from flow and solve two systems separately for every time step once (Sequential Non-Iterative Approach), or to solve the chemistry and transport equations separately and iterate until convergence (Sequential Iterative Approach), or to solve

all the equations simultaneously (Global Implicit Approach). The Sequential non-iterative approach (SNIA) was chosen in the current work presented in this chapter for its simplicity of implementation.

We assume Darcy flow and the steady state pressure equation is solved using the finite element method implemented in CSMP++. The advection-diffusion equations for the transport of all mobile aqueous species are solved by means of CSMP++ functionality using the finite element – finite volume method. All transport equations are solved by operator splitting: the finite element method is used for diffusion-dispersion, and the explicit finite volume method is used for advective transport.

The chemical reactions solver is implemented as a separate module that performs chemical equilibrium calculations based on the Law of Mass Action for every finite volume at each time step independently. All reactions (for aqueous complexes and minerals) are considered under equilibrium control. The Davies equation was used to calculate the activity coefficients [24]. The resulting system of non-linear algebraic equations is solved by the Newton-Raphson method using a KINSOL solver from SUNDIALS [32].

SNIA coupling of transport and chemistry is performed in the following way. First, total component concentrations are transported with a time step chosen according to the CFL condition, i.e advection-diffusion equations are solved for every basic species one by one on the whole grid. Second, the chemical equilibrium is calculated using the LMA solver in each finite volume.

3.2.1 Law of Mass Action equations for the calcite-dolomite system

A general description of the Law of Mass Action method can be found elsewhere [26, 29, 30]. Here I present an equation derivation for our particular system: aqueous solution with two minerals – calcite and dolomite – under thermodynamic control. We consider the system of the following reactions:

$$\begin{aligned} CaCO_3 &\leftrightarrows Ca^{2+} + CO_3^{2-} \\ CaMg(CO_3)_2 &\leftrightarrows Ca^{2+} + Mg^{2+} + 2CO_3^{2-} \\ OH^- &\leftrightarrows H_2O - H^+ \\ HCO_3^- &\leftrightarrows H^+ + CO_3^{2-} \\ CO_2 &\leftrightarrows -H_2O + 2H^+ + CO_3^{2-} \end{aligned}$$

with equilibrium constants $K_i,\ i = \overline{1,5}$ respectively.

We choose Ca^{2+}, Mg^{2+}, H^+, CO_3^{2-}, $CaCO_3$, $CaMg(CO_3)_2$ as primary species (components), and OH^-, HCO_3^-, CO_2 as secondary species. For basic species we consider the total concentrations in aqueous phase:

$$\begin{aligned}
T_{Ca^{2+}} &= [Ca^{2+}], \\
T_{Mg^{2+}} &= [Mg^{2+}], \\
T_{H^+} &= [H^+] - \frac{1}{K_3}\frac{1}{[H^+]} + \frac{1}{K_4}[H^+][CO_3^{2-}] + \frac{2}{K_5}[H^+]^2[CO_3^{2-}], \\
T_{CO_3^{2-}} &= [CO_3^{2-}] + \frac{1}{K_4}[H^+][CO_3^{2-}] + \frac{1}{K_5}[H^+]^2[CO_3^{2-}].
\end{aligned}$$

Here the activity coefficients have been omitted for sake of simplicity of representation and the water concentration was assumed to be equal to one.

The next step is to formulate the conservation of mass equations for each basic species:

$$T_{Ca^{2+}} + n_{CaCO3} + n_{CaMg(CO_3)_2} = \text{const}, \tag{3.1}$$

$$T_{Mg^{2+}} + n_{CaMg(CO_3)_2} = \text{const}, \tag{3.2}$$

$$T_{H^+} = \text{const}, \tag{3.3}$$

$$T_{CO_3^{2-}} + n_{CaCO3} + 2\, n_{CaMg(CO_3)_2} = \text{const}, \tag{3.4}$$

where n_{CaCO3} and $n_{CaMg(CO_3)_2}$ are the mineral concentrations (in moles per kilogram of water) for calcite and dolomite respectively. Mineral concentrations are mathematical entities used in the calculations and do not have a physical meaning. They are calculated from the total mineral amount per unit volume of porous media divided by the total mass of water in this volume.

The mineral concentrations are also unknowns, therefore we need two additional equations that describe the mineral saturation state:

$$K_1 = [Ca^{2+}][CO_3^{2-}], \tag{3.5}$$

$$K_2 = [Ca^{2+}][Mg^{2+}][CO_3^{2-}]^2. \tag{3.6}$$

If the initial total component concentrations and the mineral amounts are known, the equilibrium composition can be calculated. Species concentrations can vary by orders of magnitude, which is why the calculations are usually performed in terms of logarithms of concentrations. We re-write equations (3.1–3.6) with respect to the new unknowns $x_1 = \ln([Ca^{2+}])$, $x_2 = \ln([Mg^{2+}])$, $x_3 = \ln([H^+])$, $x_4 = \ln([CO_3^{2-}])$, $x_5 = n_{CaCO_3}$, $x_6 = n_{CaMg(CO_3)_2}$ and new constants $k_i = \ln(K_i)$, $i = \overline{1,5}$:

$$
\begin{aligned}
e^{x_1} + x_5 + x_6 - T_1 - n_1 - n_2 &= 0, && (3.7)\\
e^{x_2} + x_6 - T_2 - n_2 &= 0, && (3.8)\\
e^{x_3} - e^{-k_3 - x_3} + e^{-k_4 + x_3 + x_4} + 2e^{-k_5 + 2x_3 + x_4} - T_3 &= 0, && (3.9)\\
e^{x_4} + e^{-k_4 + x_3 + x_4} + e^{-k_5 + 2x_3 + x_4} + x_5 + 2x_6 - T_4 - n_1 - 2n_2 &= 0, && (3.10)\\
-k_1 + x_1 + x_4 &= 0, && (3.11)\\
-k_2 + x_1 + x_2 + 2x_4 &= 0, && (3.12)
\end{aligned}
$$

where T_1, T_2, T_3, T_4 are the initial total aqueous concentrations of calcium, magnesium, hydrogen and carbonate ions respectively; n_1, n_2 are the initial mineral amounts for calcite and dolomite.

Equations (3.8-3.12) are solved using the Newton-Raphson method, where the Jacobian matrix is analytically derived:

$$
\begin{pmatrix}
e^{x_1} & 0 & 0 & 0 & 1 & 1\\
0 & e^{x_2} & 0 & 0 & 0 & 1\\
0 & 0 & e^{x_3} + e^{-k_3 - x_3} & e^{-k_4 + x_3 + x_4} + 2e^{-k_5 + 2x_3 + x_4} & 0 & 0\\
 & & +e^{-k_4 + x_3 + x_4} + 4e^{-k_5 + 2x_3 + x_4} & & & \\
0 & 0 & e^{-k_4 + x_3 + x_4} + 2e^{-k_5 + 2x_3 + x_4} & e^{x_4} + e^{-k_4 + x_3 + x_4} + e^{-k_5 + 2x_3 + x_4} & 1 & 2\\
1 & 0 & 0 & 1 & 0 & 0\\
1 & 1 & 0 & 2 & 0 & 0
\end{pmatrix}
$$

In order to take the ion activities into account, one needs to simply replace the species concentrations by the new unknowns $a_i = \gamma_i c_i$ in equations (3.8-3.12), where a_i is the activity of the i-th species, c_i is the concentration and γ_i is the activity coefficient. In this work, Davies equation was used to calculate the activity coefficients [24]:

$$
\log \gamma_i = -A z_i^2 \left[\frac{\sqrt{I}}{(1 + \sqrt{I})} - 0.3\, I \right],
$$

where $A \approx 0.5$ at 25°C, I is the ionic strength of the aqueous solution and z_i is the charge of the i-th species.

3.2.2 Benchmarking

We consider a 1D column of a porous medium of $0.5\,m$ length, with the bulk density $\rho_b = 1800\,kg/m^3$ and the porosity $\phi = 0.32$. The model geometry is shown in Figure 3.1. The pore fluid is initially equilibrated with calcite. Standard conditions

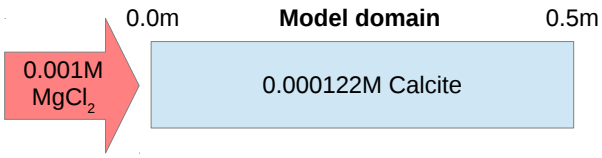

Figure 3.1: Model geometry and setup

Table 3.1: Component and solid initial concentrations

	Boundary	Initial	Units
pH	7.06	9.91	-
Ca^{2+}	0.0	1.239×10^{-4}	mol/kg_w
CO_3^{2-}	0.0	1.239×10^{-4}	mol/kg_w
Mg^{2+}	$1. \times 10^{-3}$	0.0	mol/kg_w
Cl^-	$2. \times 10^{-3}$	0.0	mol/kg_w
$CaCO_3(s)$	0.0	2.17×10^{-5}	mol/kg_{soil}
$CaMg(CO_3)_2(s)$	0.0	0.0	mol/kg_{soil}

are assumed (1 atm, 25 °C). The column is flushed from left to right with $MgCl_2$ solution with a flow rate $q = 3 \times 10^{-6}\, m/s$, resulting in a mean pore velocity $v = 9.375 \times 10^{-6}\, m/s$. As the reaction front progresses, calcite dissolves and dolomite is formed temporarily as a moving zone. Initial and boundary concentrations for aqueous species and minerals are presented in Table 3.1. Table 3.2 summarizes the reactions taken into account and corresponding equilibrium constants. Chloride does not react with anything, but serves as a tracer.

The model domain is discretized with a mesh size of $0.01\, m$, a time step $\Delta t = 200\, s$ is used in the simulation. The simulation results after $21000\, s$, compared with the results from Engesgaard & Kipp[24], are presented in Figures 3.2 and 3.3. These results are in a good agreement, and both show an increase of the calcite and magnesium ion concentrations that coincides with the beginning of the dolomite zone and a peak in the calcite concentration at the end of the dolomite zone and the beginning of the calcite zone. The comparison of the chloride concentration profiles highlights the differences between the transport schemes used in two codes.

Table 3.2: Chemical reactions with values of the equilibrium constants (in brackets)

$CaCO_3(s)$	=	$Ca^{2+} + CO_3^{2-}$	(-8.47)
$CaMg(CO_3)_2(s)$	=	$Ca^{2+} + Mg^{2+} + 2CO_3^{2-}$	(-17.17)
$Ca^{2+} + CO_3^{2-}$	=	$CaCO_3(aq)$	(3.23)
$Mg^{2+} + CO_3^{2-}$	=	$MgCO_3(aq)$	(2.98)
$H^+ + OH^-$	=	H_2O	(-14.01)
$H^+ + CO_3^{2-}$	=	HCO_3^-	(10.31)
$2H^+ + CO_3^{2-}$	=	$H_2O + CO_2(aq)$	(16.71)
Cl^-	=	Cl^-	

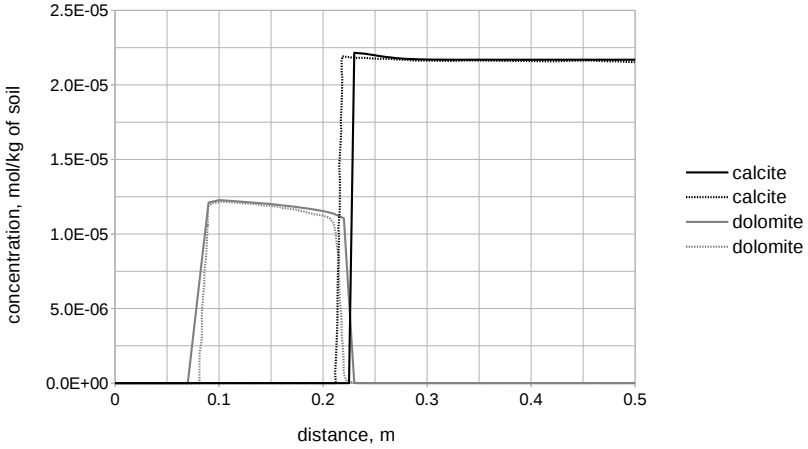

Figure 3.2: Calcite and dolomite profiles after 21000 s of simulation time: solid lines are the results obtained with CSMP++, dashed lines from Engesgaard & Kipp [24].

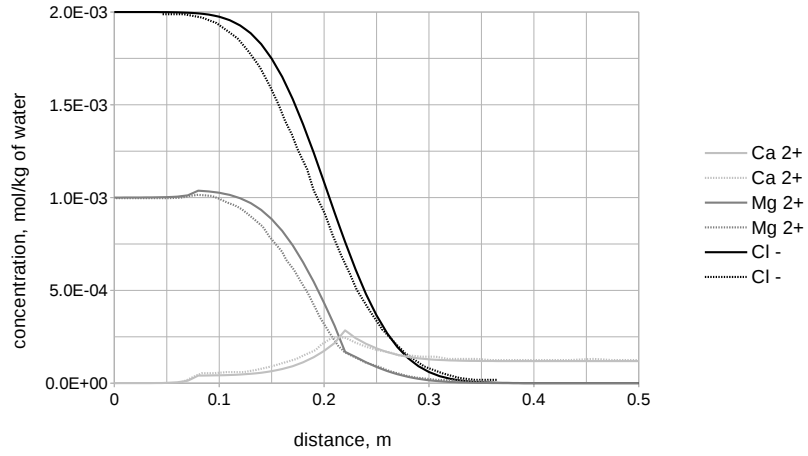

Figure 3.3: Aqueous concentration profiles after 21000 s of simulation time: solid lines are the results obtained with CSMP++, dashed lines from Engesgaard & Kipp [24].

3.3 2D simulation of a geological cross-section

The formation of dolomite was studied in 2D for the chemical system described in the previous section. A 2D geological cross-section for this simulation is shown in Figure 3.4. This sketch represents the following hypothetical geological scenario designed to illustrate interactions between permeability and mineralogy contrasts between units of geologically realistic geometry. The calcite mudstone was eroded and the quartz sandstones deposited on top of it capped by a calcite sand unit. Subsequently, during deposition of a sequence of quartz sandstone layers, a number of calcite bioherms were established. Finally this package is capped by a coarse calcite sand layer.

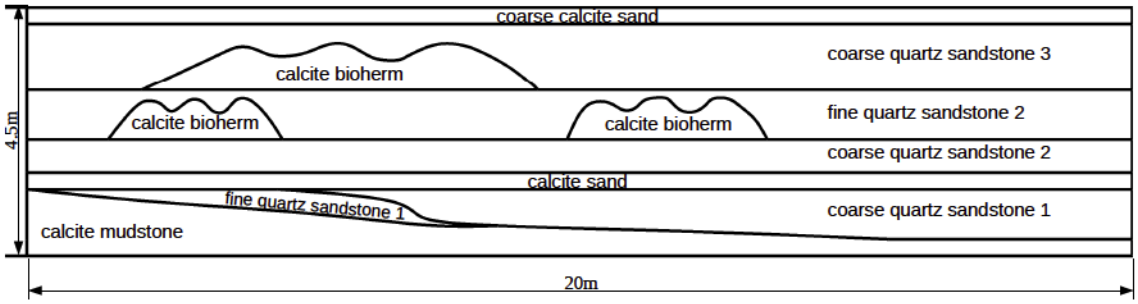

Figure 3.4: Geological cross-section

3.3.1 Computational model

The model of the geological cross-section has a rectangular geometry, with a height of $4.5m$ and a length of $20m$ and was meshed in ANSYS ICEM CFD, resulting in an unstructured grid consisting of 18712 triangles with a mesh size of $0.1\,m$.

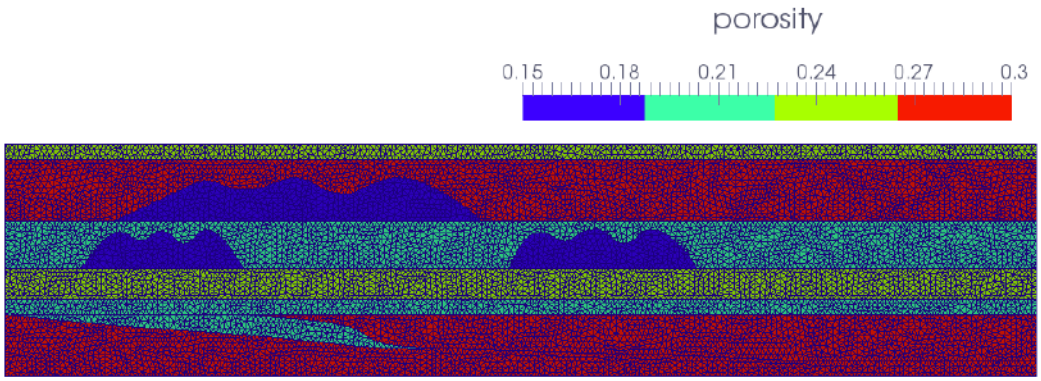

Figure 3.5: 2D cross-section properties: porosity

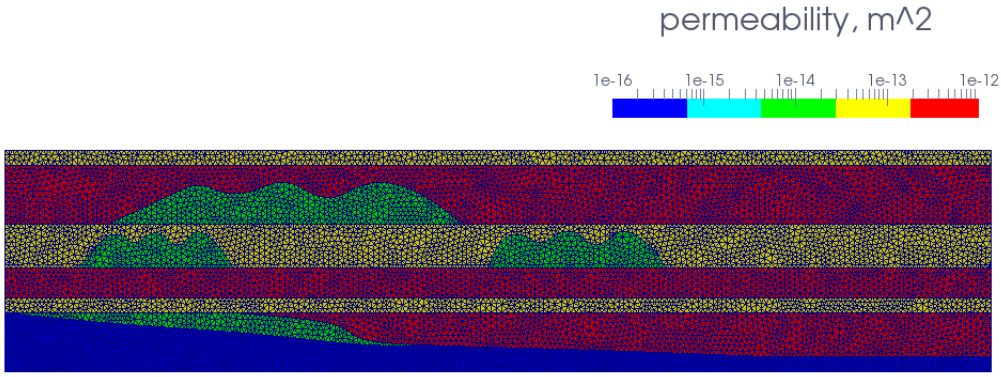

Figure 3.6: 2D cross-section properties: permeability

Rock property values were chosen typical for quartz sandstone and carbonate rocks. Figures 3.5 and 3.6 represent porosity and permeability distributions, respectively (see also Table 3.3). The sandstone layers have the highest porosity and permeability and therefore will serve as main flow paths.

We assumed very small amounts of reactive calcite (0.6% of the total rock volume for the bioherms and 1% for the rest of calcite containing rocks). The rest of the rock volume was assumed non-reactive. This assumption was made in order to be consistent with the small mineral amounts used in the 1D benchmark.

Initial calcite amounts in moles per litre of solution were calculated according to the formula:

$$n = \frac{x}{100} \cdot \frac{(1-\phi)}{\phi \cdot V_{mol}},$$

where x is the calcite rock volume percentage, ϕ is the porosity, V_{mol} is the calcite molar volume ($36.93 \cdot 10^{-3}\, l/mol$). This calculated initial calcite distribution is shown in Figure 3.7.

Fluid properties were taken for pure water at standard conditions (25 °C, 1 atm). Water equilibrated with calcite was used as the initial bulk water composition and $MgCl_2$ solution was injected through the left boundary.

Constant inflow of $MgCl_2$ solution on the left model boundary and constant pressure (atmospheric) on the right boundary were used as flow boundary conditions. The inflow rate was chosen such that the flow velocity did not exceed $1000\, m/yr$ (consistent with the velocity used in the 1D benchmark). Fluid pressure distribution and velocity field are shown in Figure 3.8.

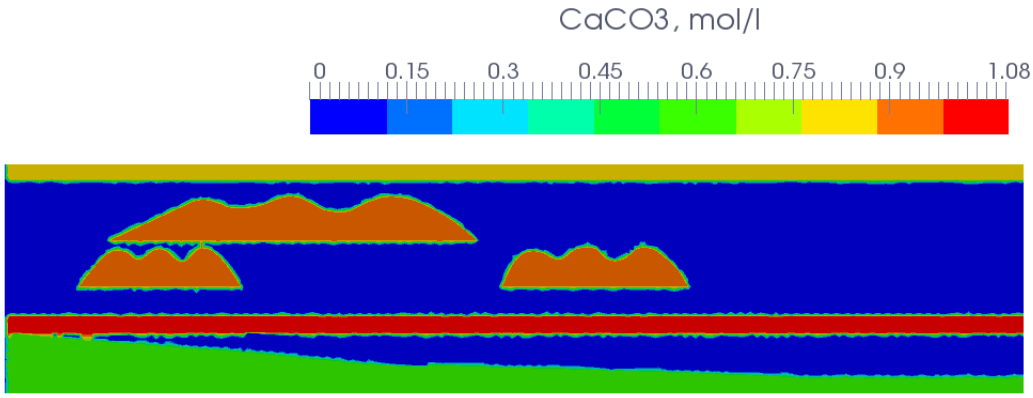

Figure 3.7: 2D cross-section properties: initial calcite amounts

Table 3.3: Rock properties

	porosity [-]	permeability $[m^2]$	calcite amount [mol/l]
calcite mudstone	0.3	10^{-16}	0.63
fine quartz sandstone 1	0.2	10^{-14}	0.0
coarse quartz sandstone 1	0.29	10^{-12}	0.0
calcite sand	0.2	10^{-13}	1.08
coarse quartz sandstone 2	0.25	10^{-12}	0.0
calcite bioherms	0.15	10^{-14}	0.92
fine quartz sandstone 2	0.2	10^{-13}	0.0
coarse quartz sandstone 3	0.3	10^{-12}	0.0
coarse calcite sand	0.25	10^{-13}	0.81

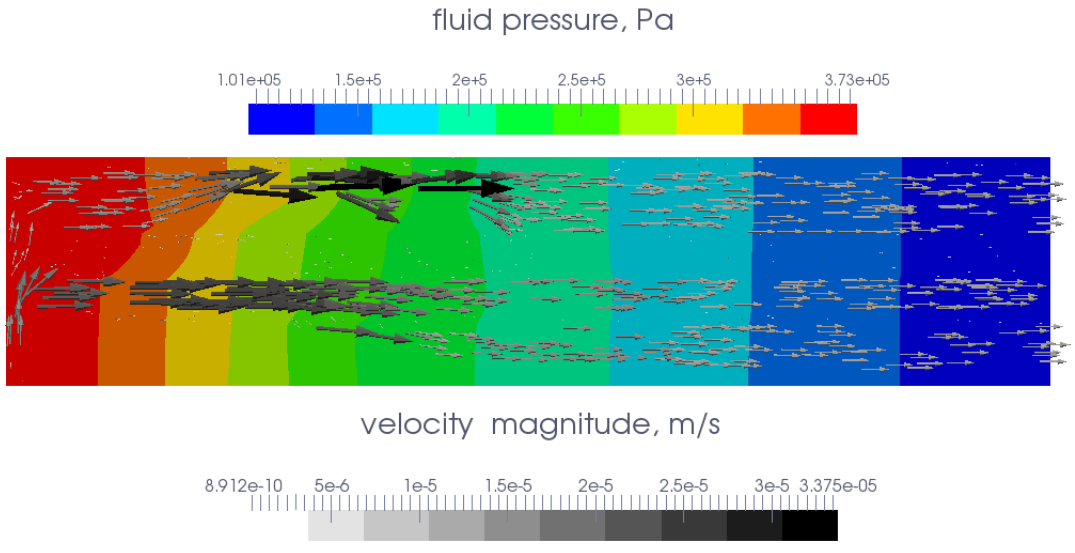

Figure 3.8: Fluid pressure and velocity field

3.3.2 Results

Calcite dissolution – dolomite precipitation was simulated for a total of 1500 days. In Figures 3.9 and 3.10 the magnesium ion concentration is shown after 500 and 1500 simulation days, respectively. The flow is initially concentrated in the three horizontal coarse and fine quartz sandstone layers as well as in the coarse *quartz sandstone 1* layer once the flow passes the low-permeability bridge in the *calcite sand* layer. Very little flow occurs in the bioherms, *fine quartz sandstone 1* and in the mudstone layers due to their low permeability.

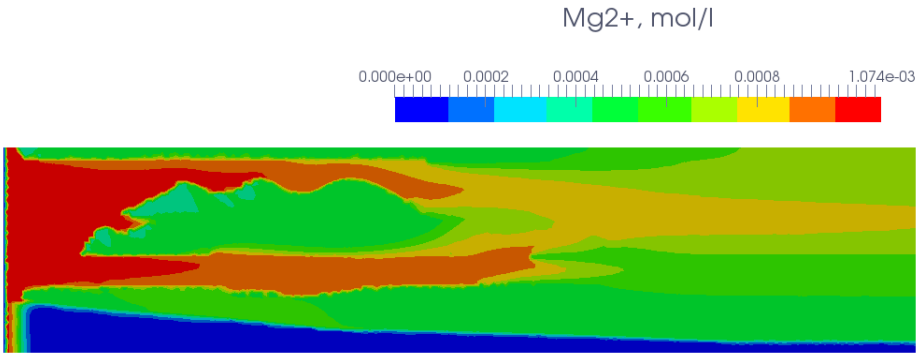

Figure 3.9: Mg^{2+} concentration distribution after 500 days

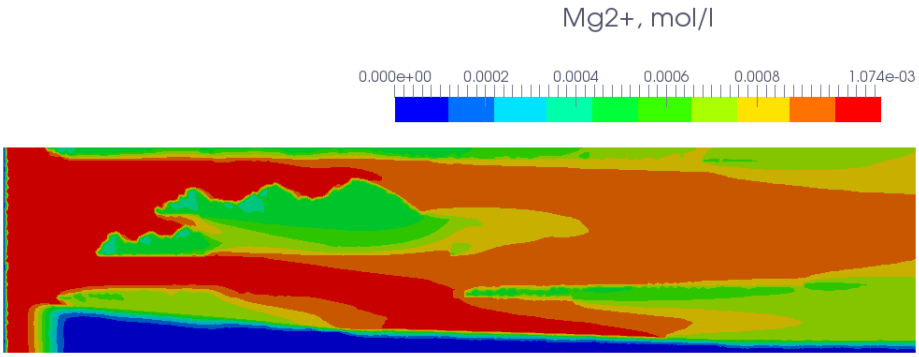

Figure 3.10: Mg^{2+} concentration distribution after 1500 days

Results in terms of calcite and dolomite molalities are presented in Figures 3.11 and 3.12 after 500 and 1500 days of simulation respectively. Two bioherms closer to the left boundary get partially dolomitised on the up-flow side, with the rightmost bioherm only dolomitsed from the bottom as the upstream bioherms obstruct the flow. Calcite sands get partially dolomitized due to Mg^{2+} ions provided from the permeable sandstone layers. The calcite mudstone layer gets partially dolomitized after the calcite sand layer has been completely dolomitized in the middle part.

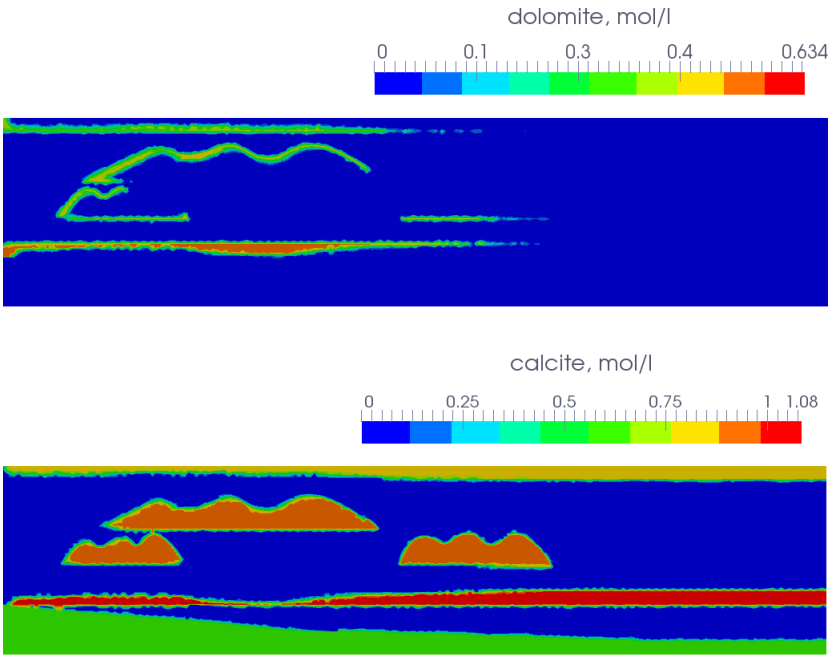

Figure 3.11: Calcite and dolomite distribution after 500 days

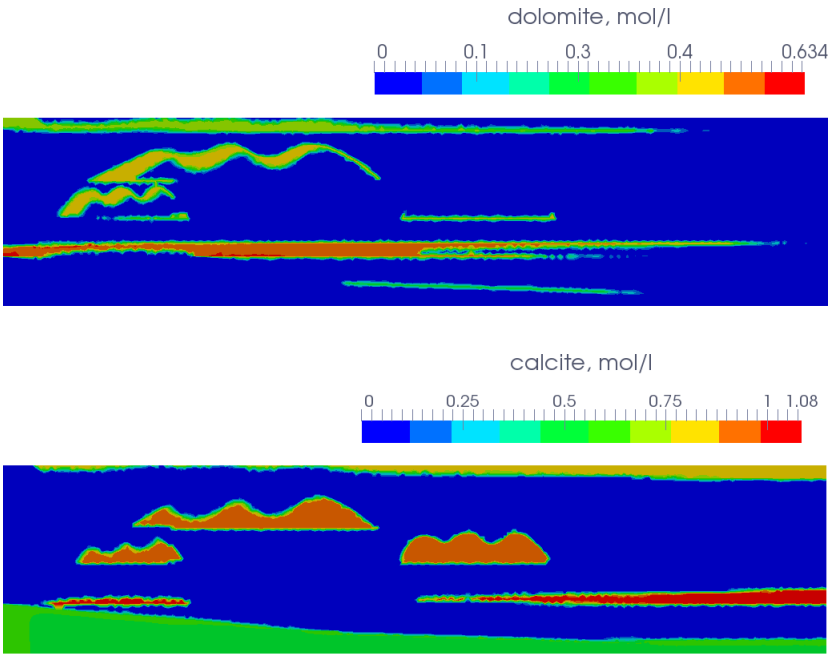

Figure 3.12: Calcite and dolomite distribution after 1500 days

3.4 Conclusions

A prototype reactive transport modelling code was implemented in the framework of the CSMP++ software library. A 1D benchmark was used to verify the method. After that a 2D simulation was performed and the ability of combining chemistry with transport with realistic geometry was shown.

The code has a modular structure and can be easily upgraded, for instance, by implementing another activity model (e.g. the Pitzer model). The code is flexible since the chemistry solver is independent of the flow computations and can be easily replaced (e.g with a GEM solver). The chemistry calculations are performed on every node separately and are thus providing a high potential for parallelization.

On the other hand, the current implementation has its drawbacks, too. First, the method is very sensitive to the initial guess for the Newton solver and some initial values can result in negative species concentrations. As a rule of thumb, we recommend to set an initial guess for a primary species concentration equal to 95% of the corresponding total primary species concentration, obtained from a previous transport step.

The second limitation of the method is that the equilibrium mineral assemblage is difficult to obtain and non-unique solutions can exist. The strategy that was used in the present work was the following. If after the Newton loop some mineral concentrations are negative (that means that the solution is undersaturated with respect to that mineral, but the mineral is already totally dissolved), we exclude the "most negative" mineral from the system and repeat the computations until we get a stable composition. This strategy, though providing physically meaningful results, is computationally very costly.

The following chapters deal with the above mentioned problems as well as include treatment of kinetically controlled reactions and dependence of porosity and permeability on mineral dissolution/precipitation.

Chapter 4

Reactive transport modelling of dolomitisation using the new CSMP++ GEM coupled code

The GEMS3K code for solving geochemical equilibria by Gibbs energy minimization was coupled with the CSMP++ framework that implements the finite element – finite volume method to solve partial differential equations for single-phase non-isothermal flow in porous media and transport of solutes.

This chapter starts with the governing equations and numerical solution methods for the reactive transport model. After that, two sets of benchmarking results are presented. The first benchmark, also used in the previous chapter, is a well-known 1D model of dolomitisation by $MgCl_2$ solution with thermodynamic reactions. In the second benchmark, CSMP++GEM was compared with TOUGHREACT on a 1D model of dolomitisation by seawater taking into account mineral reaction kinetics.

4.1 Introduction

Reactive transport modelling (RTM) is an emerging, powerful tool for understanding natural systems where fluid flow and chemical reactions occur simultaneously. RTM has been used for many different applications including the prediction of the fate of CO_2, radioactive and chemical waste, simulation of ore deposition in fractures and veins caused by the flow of hydrothermal fluids and the accompanying host rock alteration, geothermal energy production and formation damage near wells in hydrocarbon reservoirs [33].

RTM has also been used to understand the formation of dolomite by replacement of calcite, which can be an important control on carbonate reservoir quality. Dolomi-

tisation is a case of mineral replacement that modifies the volume of the solid phases [34]. Dolomite is more dense than calcite and thus 1:1 molar replacement generates up to 13% additional porosity. Dolomitisation can also significantly modify permeability by reorganising the pore geometry of the pore network, for example by replacing allochems and matrix with sucrosic dolomite crystals [35]. Dolomite can also form as a primary precipitate from supersaturated fluids, leading to porosity occlusion. With regard to carbonate-hosted petroleum reservoirs, understanding the spatial distribution of dolomite bodies and the resulting changes in petrophysical properties (porosity-permeability) is key for accurate estimation of hydrocarbon reserves and improving recovery efficiency.

Dolomitisation has been simulated using RTM in a number of diagenetic settings, driven by different fluid circulation systems. These include near surface and shallow burial reflux of platform-top evaporative brines [36, 37, 38, 39] , intermediate and deep burial geothermal circulation and compactional flow [40, 41, 42], and high-temperature fluid expulsion through faults and fractures [43].

Dolomitisation (replacement of calcite $CaCO_3$ by dolomite $CaMg(CO_3)_2$) is a kinetically controlled, partial equilibrium process [44]. At near-surface conditions the precipitation rate of dolomite [45] can be a million times slower than that of calcite (at the same supersaturation) or calcium sulphates, whereas the dissolution rates of dolomite and calcite are comparable [46]. However, this situation rapidly changes at elevated temperatures, generating potential for dolomite formation under hydrothermal conditions [47] (i.e. at temperature 5-10 C higher than that of the host rock).

Although RTM has more than a 20 years long history [28, 29], only now has sufficient computational power become available to simulate processes over geologic spans of time on detailed grids, although full-scale 3-D simulations are still very rare. Slow dolomitisation processes need to be simulated over geological times (hundreds of thousands up to millions of years), which implies long computation times, even if precise and fast numerical methods and algebraic solvers are in use.

When simulating Thermal-Hydrological-Chemical (THC) processes, it is important to account for the density dependence on pressure, temperature and salinity, as well as changes in fluid mass due to chemical reactions such as aqueous and surface complexation, mineral dissolution or precipitation.

A fully implicit solution of coupled flow-transport-chemistry equations is still challenging or intractable in many real-world applications. Thus, the sequential

"operator-splitting" approach for solving equations is implemented in codes that couple the transport (advection-diffusion) simulator with a chemical equilibrium speciation solver.

The choice of the finite volume method for solute transport is crucial for maintaining local mass conservation. Unlike the finite element method that is used in OpenGeoSys code for solving both flow and solute transport (http://opengeosys.org), in the finite element - finite volume method used in the CSMP++GEM coupling, flux continuity across the finite volume confining surfaces guarantees the local mass conservation. The Integral Finite Differences method (IFD) that is used in TOUGH2 and TOUGHREACT [18] is based on the conservation laws, but only allows the use of corner point grids that are much less geometrically flexible than unstructured grids.

In codes such as TOUGHREACT (http://esd1.lbl.gov/research/projects/tough/software/toughreact.html), the chemical speciation solver uses the so-called LMA (Law of Mass Action) method, which is based on a selection of *master species* (usually aqueous ions and water, sometimes minerals) whose amounts enter the material balance equations directly, and *product species* whose amounts are defined via the LMA equations for reactions of formation of product species from master species and respective equilibrium constants. The systems of material balance and LMA equations are then solved simultaneously using the Newton-Raphson method [26]. Implicit in the LMA method are the assumptions that the aqueous solution phase is predominant in the system; redox state, alkalinity and assemblage of stable phases are known beforehand; and, if a non-ideal solid solution is involved, its equilibrium composition can be obtained from the aqueous phase composition.

As stated by Reed [26], the real challenge of the LMA method lies in the selection of mineral phases, especially solid solutions, to be included into the mass balance. This is normally done upon checking the saturation indices (SI) over the entire list of minerals in the process of solving the LMA speciation task many times and adding/removing individual mineral phases one-by-one. Some LMA algorithms, e.g. the Geochemists Workbench [33], cannot solve equilibria with gas mixtures, solid solutions or melts in the material balance because gases, minerals or melt components are treated as master species.

A complementary Gibbs Energy Minimization (GEM) method [48, 49] finds the unknown phase assemblage and speciation of all phases from the elemental bulk composition of the system by minimizing its total Gibbs energy while maintaining the elemental material balance. All species (components) in all phases must be provided

with their elemental formulae and standard Gibbs energy per mole; no separation into master and product species is needed.

Compared with LMA, the GEM methods do not require any assumptions about the equilibrium state, and are capable of solving equilibria in complex heterogeneous chemical systems with many non-ideal multicomponent solution phases [49, 50]. This makes possible RTM simulations of complex heterogeneous equilibria with intrinsic redox states, aqueous electrolyte, non-ideal gas mixtures (fluids), mineral solid solutions, melts, adsorption and ion exchange. GEM algorithms are already used for solving chemical speciation in the OpenGeoSys–GEM [51] and CSMP–GEM [52] coupled RTM codes.

Given these considerations, my motivation was to create a versatile coupled RTM code (called CSMP++GEM) by combining the best features of the CSMP and GEM families of codes. Compared with OpenGeoSys-GEM, this coupling uses the finite element – finite volume method which guarantees local mass conservation and flexible adaptation to mesh geometries especially along permeability discontinuities such as fractures and interfaces between contrasting beds. As an advantage over the existing CSMP-GEM coupled code, my code implements detailed controls on mineral-water reaction kinetics, which is a crucial issue for modelling reactions such as dolomitisation [45].

The new CSMP++GEM reactive transport code combines relevant features of the Complex Systems Modelling Platform (CSMP++) framework [15] and the GEMS3K standalone code [49] enabling the simulation of THC systems with complex geometry and complex chemistry across wide ranges of temperatures, pressures and compositions, and for various transport regimes.

CSMP++ is an object-oriented software library written in C++ that uses unstructured grids to represent model geometry and finite element – finite volume method to solve partial differential equations. To solve systems of algebraic linear equations CSMP++ uses the SAMG solver (Fraunhofer, Germany) [16], for being accurate, robust and fast. Other strengths of CSMP++ are: (1) rigorous treatment of material interfaces, (2) region based computations and (3) realistic boundary conditions.

GEMS3K (http://gems.web.psi.ch/GEMS3K) is an open-source standalone C++ code for Gibbs energy minimization that computes (partial) equilibruim chemical speciation in complex heterogeneous multi-phase systems from the elemental bulk composition of the system, thermodynamic data for temperature and pressure of interest, parameters of mixing [49, 50], and kinetic rate parameters imposing additional metastability restrictions for certain phases and components. The initial chemical systems for

RTM can be set up and tested using the GEM-Selektor v.3 graphical user interface with built-in thermodynamic and model databases (http://gems.web.psi.ch/GEMS3) and then exported as sets of GEMS3K input files.

This chapter starts with the problem statement and equations derivation, followed by the numerical methods description and benchmarking on a simple 1D model. In the end the results of a comparison of our reactive transport model with TOUGHREACT are presented.

4.2 Methodology

In this section, I first present the governing equations that describe a single-phase multi-component flow in fully saturated porous media coupled with chemical reactions. The flow is assumed to be slightly compressible and non-isothermal. Minerals (calcite, dolomite) are either in equilibrium or under kinetic control; their dissolution and/or precipitation result in changes in porosity and permeability. We briefly describe the Gibbs energy minimization method with emphasis on the phase stability index and on how partial equilibria are controlled by kinetic rates of mineral-water interaction. After the problem statement, we outline the numerical solution procedure that was used in this work.

4.2.1 Governing equations for single phase flow and reactive transport in porous media

The chemical composition of the system is defined in terms of the total amounts of so-called independent components (IC), typically chemical elements and electrical charge. Each IC can be present in the form of various aqueous ions and molecules dissolved in water, as well as in different minerals of the solid phase. Aqueous concentration c_i is the total amount of moles of the i-th independent component in all species dissolved in the aqueous phase per unit volume.

For each IC, the conservation of mass in the following form holds:

$$\frac{\partial(\phi c_i)}{\partial t} + \nabla \cdot (c_i \mathbf{v}_i) = q_i, \ \forall i = \overline{1, N}, \tag{4.1}$$

where ϕ is the porosity, $\mathbf{v}_i$ is the flow velocity of the i-th independent component, q_i is the source/sink term that accounts for the exchange of i-th IC between solid and aqueous phases due to mineral dissolution/precipitation, and N is the number of independent components. We assume that there are no sources/sinks other than those caused by chemical reactions.

Let us define the aqueous solution density $\rho = \sum_{i=1}^{N} c_i M_i$, using M_i – the molar mass of the i-th independent component. The solution mass flux is equal to the sum of mass fluxes of individual components:

$$\rho \mathbf{v} = \sum_{i=1}^{N} c_i M_i \mathbf{v}_i.$$

The sum of the product of equations (4.1) by M_i yields the continuity equation for the aqueous phase:

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = Q, \tag{4.2}$$

where $\mathbf{v}$ is the solution velocity, and Q is the source term that accounts for the mass exchange between aqueous and solid phases due to chemical reactions:

$$Q = \sum_{i=1}^{N} q_i M_i.$$

The fluid velocity is related to the fluid pressure by means of Darcy's law:

$$\mathbf{v} = -\frac{k}{\mu}(\nabla p - \rho \mathbf{g}), \tag{4.3}$$

where k is the permeability, μ is the fluid viscosity, p is the fluid pressure, and $\mathbf{g}$ is the gravity acceleration vector.

The chemical composition of a solution is expressed in terms of equivalent salinity:

$$X = 1 - \frac{m_w}{m},$$

where m is the mass of the solution, m_w is the mass of pure water (solvent).

We assume that solution density $\rho = \rho(p, T, X)$ depends on pressure $p = p(t, \mathbf{x})$, temperature $T = T(t, \mathbf{x})$ and chemical composition $X = X(t, \mathbf{x})$, and calculate it's time derivative:

$$\frac{\partial \rho}{\partial t} = \frac{\partial \rho}{\partial p}\frac{\partial p}{\partial t} + \frac{\partial \rho}{\partial T}\frac{\partial T}{\partial t} + \frac{\partial \rho}{\partial X}\frac{\partial X}{\partial t} = \rho \beta_f \frac{\partial p}{\partial t} + q_{TX}, \tag{4.4}$$

where

$$\beta_f = -\frac{1}{V}\frac{\partial V}{\partial p} = \frac{1}{\rho}\frac{\partial \rho}{\partial p}$$

is the isothermal isosaline fluid compressibility, defined as the relative change of the solution volume V (or solution density ρ) with a change in pressure, and

$$q_{TX} = \frac{\partial \rho}{\partial T}\frac{\partial T}{\partial t} + \frac{\partial \rho}{\partial X}\frac{\partial X}{\partial t}$$

is the source term that accounts for the temperature and salinity induced solution density change at constant pressure.

Neglecting the thermal expansion of the rock, we express the porosity change in terms of isothermal pore compressibility:

$$\beta_\phi = \frac{1}{\phi}\frac{\partial \phi}{\partial p},$$

$$\frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial p}\frac{\partial p}{\partial t} = \phi \beta_\phi \frac{\partial p}{\partial t}. \tag{4.5}$$

Porosity changes caused by chemical reactions are discussed below.

Using equations (4.4) and (4.5), we rewrite the left-hand side of (4.2) to first get:

$$\frac{\partial(\phi\rho)}{\partial t} = \phi\frac{\partial \rho}{\partial t} + \rho\frac{\partial \phi}{\partial t} =$$
$$= \phi\rho\beta_f \frac{\partial p}{\partial t} + \phi\, q_{TX} + \rho\phi\beta_\phi \frac{\partial p}{\partial t},$$

and then inserting (4.3) into the right hand side of (4.2), we arrive at the transient pressure equation:

$$\rho\phi(\beta_f + \beta_\phi)\frac{\partial p}{\partial t} = \nabla \cdot (\rho\frac{k}{\mu}(\nabla p - \rho \mathbf{g})) + q_{TX} + Q. \tag{4.6}$$

The energy conservation equation is written in the form [53]:

$$(\phi\rho c_{pf} + (1-\phi)\rho_r c_{pr})\frac{\partial T}{\partial t} = \nabla \cdot ((\phi K_f + (1-\phi)K_r)\nabla T) - \nabla \cdot (\mathbf{v}\rho c_{pf} T), \tag{4.7}$$

where T is the temperature, K_f and K_r are the fluid and rock thermal conductivities, c_{pf} and c_{pr} are the fluid and rock specific heat capacities, respectively; ρ_r is the rock density. No thermal effects of chemical reactions are taken into account.

Returning to the component transport equation (4.1), we introduce the entity $\mathbf{J} = c_i(\mathbf{v}_i - \mathbf{v})$ – the component flux due to the deviation from the mean flow velocity.

$$\nabla \cdot (c_i \mathbf{v}_i) = \nabla \cdot \mathbf{J} + \nabla \cdot (c_i \mathbf{v})$$

This flux can be described using the Fick's law $\mathbf{J} = -D\nabla c_i$, and equation (4.1) can be rearanged in the following form:

$$\frac{\partial(\phi c_i)}{\partial t} + \nabla \cdot (\mathbf{v}_i c_i) - \nabla \cdot (D\nabla c_i) = q_i,\ \forall i = \overline{1, N}, \tag{4.8}$$

where D is the diffusion-dispersion coefficient, assumed to be constant and the same for all species.

Equations (4.6), (4.7) and (4.8) together with the equation of state form the total system of equations that describe the non-isothermal slightly compressible single phase multi-component transport in porous media. In this work, an equation of state for brine from [54] was used to calculate fluid properties for the seawater salinity range.

4.2.2 Gibbs energy minimization method and partial equilibrium kinetics

The GEM IPM3 algorithm [49], as implemented in the GEM software (http://gems.web.psi.ch), is capable of thermodynamic modelling of partial equilibria controlled by mineral-water reaction kinetics with multiple reaction pathways. In GEM IPM3, the chemical system is defined by a bulk composition vector, $n^{(b)}$, specifying the input amounts of chemical elements and charge; the standard Gibbs energy per mole for all dependent components (DC, chemical species), g^o, at T, p of interest; the parameters of (non)ideal models of mixing in solution phases [50], needed to calculate the activity coefficients γ_j of DCs indexed with j; and the additional metastability restrictions (AMR) on mole amounts of some DCs in some phases.

The GEM problem consists in finding the (unknown) equilibrium speciation $n^{(x)}$ and phase amounts $n^{(\phi)}$ in the system at given p, T, and bulk composition $n^{(b)}$ (i.e. mole amounts of ICs). The number of elements in this $n^{(b)}$ vector is $n(N)$, i.e. the number of ICs including the charge; the size of the $n^{(x)}$ vector is $n(L)$, i.e. the total number of DCs in all phases.

The Gibbs energy minimization problem

Find $n^{(x)} = \left\{ n_j^{(x)}, j \in L \right\}$ such that

$$\begin{aligned} G(n^{(x)}) &= \min \left\{ G(n^{(x)}) : n^{(x)} \in M_1 \right\}, \\ M_1 &= \left\{ n^{(x)} \in R_1 : An^{(x)} = n^{(b)} \right\}, \end{aligned} \tag{4.9}$$

where $A = \{a_{ij}, i \in N, j \in L\}$ is a matrix composed of the stoichiometry coefficients of ICs in the formulae of DCs; L is the set of $n(L)$ DC indices; R_1 is the set of constraints on $n^{(x)}$ composed of trivial non-negativity constraints (set D_0), and optional two-side AMRs (set D_3),

$$R_1 = \left\{ n^{(x)} : \begin{array}{rcll} n_j^{(x)} & \geq & 0, & j \in D_0 \\ \underline{n}_j^{(x)} & \leq & n_j^{(x)} \leq \overline{n}_j^{(x)}, & j \in D_3 \end{array} \right\}$$

and sets of indices D_0, D_3 are such that $L = D_0 \bigcup D_3$ [55, 49]. Note that $\underline{n}_j^{(x)} \geq 0$; $\overline{n}_j^{(x)} \geq 0$, and $\overline{n}_j^{(x)} \geq \underline{n}_j^{(x)}$.

The normalized total Gibbs energy of the chemical system is a scalar:

$$G(n^{(x)}) = \sum_j n_j^{(x)} \mu_j,\ j \in L,$$

where μ_j is the normalized chemical potential of the j-th dependent component, written in a simplified dimensionless form as:

$$\mu_j = \frac{g_j^o}{RT} + lnC_j + ln\gamma_j + \Xi,\ j \in L, \tag{4.10}$$

with g_j^o is the standard chemical potential (Gibbs energy per mole) of j-th DC at p, T of interest; R is the universal gas constant ($8.31451\ J/K/mol$), and $C_j = f(n_j^{(x)})$ is the j-th DC concentration relative to the standard concentration scale for the respective phase. For a DC in k-th condensed non-electrolyte solution phase, and for the water-solvent in aqueous electrolyte, C_j is defined as mole fraction x_j:

$$C_j = x_j = \frac{n_j^{(x)}}{n_k^{(\phi)}} = \frac{n_j^{(x)}}{\sum_{jp} n_{jp}^{(x)}},\ jp \in l_k, \tag{4.11}$$

where l_k is the subset of indices of all DCs belonging to the k-th phase. For an aqueous electrolyte species (not water-solvent), concentration is defined as molality m_j (moles per kilogram of water-solvent),

$$C_j = m_j = \frac{1000}{18.0153} \cdot \frac{x_j}{x_{jw}},\ j, jw \in l_{aq}, \tag{4.12}$$

where jw is the index of water solvent, 18.0153 is the molar mass of water in g/mol, and l_{aq} is the subset of indices of DCs in the aqueous electrolyte phase. For gas or plasma or gaseous fluid DCs with indices belonging to the subset l_g, the concentration is partial pressure,

$$C_j = x_j p,\ j \in l_g,$$

and for any DC forming a stable pure substance phase, the concentration is unity,

$$C_j = x_j = 1,\ j \in l_k \ \text{ and } \ n(l_k) = 1.$$

The activity coefficient γ_j of the j-th DC in its respective phase is obtained from the chosen model of non-ideal mixing (details in [50]). The non-logarithmic asymmetry

term Ξ is

$$\Xi = 1 - x_{jw},\ \forall j \in l_{aq} \backslash jw$$

for the aqueous species,

$$\Xi = 2 - x_{jw} - 1/x_{jw},\ jw \in l_{aq}$$

for the water-solvent, while $\Xi = 0$ for components of condensed mixtures, gaseous, and pure-substance phases [55].

The convex set M_1 is called a *feasible domain*, composed of the system of mass balance constraints, with an additional set of constraints R_1. If only trivial non-negativity constraints are present in the R_1 set, i.e. $D_3 = \varnothing$, then the speciation vector $\widehat{n}^{(x)}$ will be the primal solution of the problem (4.9) only if such a dual solution vector $\widehat{u}$ exists such that Karush-Kuhn-Tucker (KKT) necessary and sufficient conditions (written in vector-matrix notation) are satisfied:

$$\begin{aligned} \mu - A^T \widehat{u} &\geq 0, \\ A\widehat{n}^{(x)} = n^{(b)},\ \widehat{n}^{(x)} &\geq 0, \\ \widehat{n}^{(x)}(\mu - A^T \widehat{u}) &= 0, \end{aligned} \tag{4.13}$$

where $\widehat{\ }$ denotes optimal. For components with two-side AMRs ($D_3 \neq \varnothing$), the extended KKT conditions [55, 49] must be satisfied:

$$\begin{aligned} \mu - A^T \widehat{u} + \widehat{q} &\geq 0, \\ A\widehat{n}^{(x)} = n^{(b)},\ \widehat{n}^{(x)} &\geq 0, \\ (\underline{n}^{(x)} - \widehat{n}^{(x)})(\mu - A^T \widehat{u} + \widehat{q}) &= 0, \\ \widehat{q} \geq 0,\ (\widehat{n}^{(x)} - \overline{n}^{(x)})\widehat{q} &= 0, \end{aligned}$$

where $\widehat{q}$ is a vector of Lagrange multipliers conjugate to AMRs, and $\underline{n}^{(x)}$, $\overline{n}^{(x)}$ are vectors of lower- and upper AMRs, respectively. These KKT conditions are used by the IPM3 algorithm [49] to iteratively find accurate and precise primal $\widehat{n}^{(x)}$ and dual $\widehat{u}$ optimal solutions of the GEM problem.

The first condition from (4.13), re-written with indices using (4.10)

$$\frac{g_j^o}{RT} + lnC_j + ln\gamma_j + \Xi - \widehat{\eta}_j \geq 0,\ j \in L,\ i \in N, \tag{4.14}$$

implies that, for a j-th DC present in some equilibrium concentration $C_j \geq 0$ in its phase, the primal chemical potential μ_j must numerically equal the dual chemical potential $\widehat{\eta}_j$:

$$\widehat{\eta}_j = \sum_i a_{ij} \widehat{u}_i,\ j \in L,\ i \in N. \tag{4.15}$$

After a GEM run has converged (see more about the GEM IPM3 algorithm in GEMS3K code and its performance compared with LMA codes in [49]), the results, namely the primal speciation vector $\widehat{n}^{(x)}$, and the dual vector $\widehat{u}^{(b)}$ of chemical potentials of chemical elements and charge, provide for concentrations, activities and amounts of all components in each phase.

The stability index Ω_k of any phase, including any phases absent from the mass balance, is found as a sum of anticipated mole fractions $\widehat{x}_j$ of all DCs belonging to this phase:

$$\Omega_k = \sum_j \widehat{x}_j = \sum_j exp\left(\widehat{\eta}_j - g_j^o/RT - ln\gamma_j - \Xi_k\right),\ j \in l_k. \tag{4.16}$$

Equation (4.16) follows from combining equations (4.14, 4.15), with (4.11) or (4.12).

Modelling kinetics as partial equilibrium

In the GEM IPM3 algorithm, the Ω_k index (eq. 4.16), which has the physical meaning of a saturation index of a (mineral) phase, is used as a criterion for checking the stability of any phase. If $-\epsilon < log_{10}\Omega_k < \epsilon$ then the non-negative amount of this phase is in equilibrium with the rest of the system with a numerical tolerance $0 < \epsilon \ll 1$. If $log_{10}\Omega_k < \epsilon$ then the phase is unstable (under-saturated), but may be kept in the mass balance by the lower AMR(s) $\underline{n}_j^{(x)} > 0$ set on some of its components. If $log_{10}\Omega_k > \epsilon$ then the phase is over-stable (oversaturated) due to a positive or zero upper AMR(s) $\overline{n}_j^{(x)} > 0$ set on some or all of its components.

Overall, the GEM output phase stability index Ω_k together with input AMRs make the GEMS3K code a versatile tool for simulating phase metastability and kinetics as a series of partial equilibrium states. Thus, lower-AMRs allow stepwise simulation of dissolution of a mineral as long as its $\Omega_k < 1$; upper-AMRs allow stepwise simulation of mineral precipitation as long as $\Omega_k > 1$. In this way, it is possible to model the kinetics of mineral-aqueous reactions and of trace element uptake [56].

In a sequence of partial equilibria, each AMR can be set as a function of the time step duration Δt, the time variable t, the surface area $A_{k,t}$ of k-th solid phase, and the (absolute) net kinetic rate $R_{n,k,t}$. In a stepwise simulation, the mole amount $n_{k,t+\Delta t}$ of the mineral at time $t + \Delta t$ is set by the upper AMR $\overline{n}_{k,t+\Delta t}$ for precipitation or by the lower AMR $\underline{n}_{k,t+\Delta t}$ for dissolution:

$$\overline{n}_{k,t+\Delta t} = n_{k,t} + A_{k,t}R_{k,t}\Delta t \text{ if } log_{10}\Omega_k > \epsilon$$
$$\underline{n}_{k,t+\Delta t} = n_{k,t} - A_{k,t}R_{k,t}\Delta t \text{ if } log_{10}\Omega_k < -\epsilon$$

The surface area of k-th solid phase is obtained as

$$A_{k,t} = A_{S,k}M_{M,k}n_{k,t},$$

where $A_{S,k}$ is the input specific surface area (m^2/kg); $M_{M,k}$ is the molar mass (kg/mol), and $n_{k,t}$ is the current amount (mol) of the k-th phase.

Implementation of metastability and kinetics differs from code to code. To date there is no conventional data structure for kinetic rate parameters. Because the experimental rate constants are typically normalized per unit area, they must be scaled by the current reactive surface area of the mineral. This depends on many factors, some of them are external to the chemical system, and some are related to the particle/pore morphology, initial size distributions, and surface roughness. This is in focus of current research efforts in the geochemistry of mineral water interfaces [57, 58, 59]. In RTM, this must be recast into the impact of porosity changes on transport parameters and on reactive surface areas of minerals.

Kinetic rate laws usually contain the so-called *activity product term* related to a particular reaction mechanism, catalysis, inhibition, etc. [60], [46]. Near-equilibrium kinetic rates also depend on the *affinity term* based on the phase stability index Ω_k. Particular forms of this term reflect different nucleation, growth or dissolution mechanisms [60].

Some relevant kinetic rate equations for dissolution, precipitation, and trace element uptake in solid solutions have been implemented in the TKinMet code library used in the GEM-Selektor and the GEMS3K codes [61], [56]. The mineral-water kinetic rate laws are considered in the general form (with input parameters implemented in Phase definition records):

$$\frac{dn_k}{dt} = A_{k,t}R_{k,t} = -A_{k,t}\sum_{r}^{N(r)_k}\theta_{k,r,t}f(\kappa,E)_{k,r}f(\Pi a)_{k,r,t}f(\Omega)_{k,r,t} \tag{4.17}$$

where k is the index of solid phase of interest (pure solid or solid solution); n_k is the mole amount of k-th phase at time t; $A_{k,t}$ is the current surface area of the phase in m^2; $R_{n,k,t}$ is the total precipitation or dissolution rate (in $mol/m^2/s$); $N(r)_k$ is the number of parallel reaction mechanisms or pathways that affect the amount of k-th phase; r

is the index of a mechanism or pathway (dissolution, nucleation, and precipitation can be treated simultaneously as different mechanisms); $\theta_{k,r,t}$ is the effective fraction of surface area of k-th phase assigned to the r-th reaction mechanism.

Time-dependent parameters $A_{k,t}$ and $\theta_{k,r,t}$ may either depend on a built-in model of particle size/area evolution or be externally controlled from the mass transport code. In eq. (4.17),

$$f(\kappa, E) = \kappa^o_{k,r} \Lambda_{k,r} e^{\frac{-E_{k,r}}{RT}} \tag{4.18}$$

is the reaction rate constant term including the temperature correction, where: $\kappa^o_{k,r}$ is the rate constant at reference temperature (25 °C) in $(mol/m^2/s)$ or other appropriate units, having a positive sign for dissolution and a negative sign for precipitation; T is temperature in K; $\Lambda_{k,r}$ is the Arrhenius factor (1 by default); and $E_{k,r}$ is the activation energy (J/mol) of r-th reaction mechanism.

The expression $e^{\frac{-E_{k,r}}{RT}}$ in eq. (4.18) occurs in the literature in a different form $e^{\frac{-E^*_{k,r}}{R}(\frac{1}{T}-\frac{1}{298.15})}$ that involves the reference temperature $298.15\,K$ [46]. Both forms are connected as:

$$\Lambda_{k,r} e^{\frac{-E_{k,r}}{RT}} = \Lambda^*_{k,r} e^{\frac{-E^*_{k,r}}{R}(\frac{1}{T}-\frac{1}{298.15})},$$

where

$$\Lambda_{k,r} = \Lambda^*_{k,r} e^{\frac{-E^*_{k,r}}{R \cdot 298.15}}$$

and $E^*_{k,r} = E_{k,r}$.

In eq. (4.17), $f(\Pi a)_{k,r,t}$ is the activity product term that involves a product term:

$$(\prod_{j}^{n(j)_{k,r}} a_{j,k,r}^{b_{j,k,r}})^{p_{k,r}},$$

where $p_{k,r}$ is the reaction order parameter; $n(j)_{k,r}$ is the number of (aqueous or gaseous or surface) species from other reacting phases involved; $a_{j,k,r}$ is the activity (fugacity) of j-th species; $b_{j,k,r}$ is the reaction stoichiometry coefficient parameter. The activity product term also involves for convenience the respective power coefficients for ionic strength, pH, pe and Eh for the aqueous electrolyte. Note that any activity dependence of the rate can be disabled, if the respective power coefficient is set to the default value of zero.

Finally, in eq. (4.17), $f(\Omega_{k,r})$ is the affinity term for r-th reaction, which can take several different forms, all using the current (at time t) k-th phase stability index Ω_k (eq. 4.16). The classic affinity term is taken in the form

$$(1 + u_{k,r} - \Omega_k^{q_{k,r}})^{m_{k,r}},$$

where $q_{k,r}$ and $m_{k,r}$ are the reaction order parameters (default 1; $m_{k,r} = 0$ disables the affinity term); and $u_{k,r}$ is the empirical parameter (default 0).

In eq (4.17), the net rate $R_{k,t}$ is taken in $(mol/m^2/s)$ by default. However, in many models of mineral dissolution or growth [62, 63], the mean orthogonal velocity of surface propagation $R_{L,k,t}$ in (m/s) is considered. $R_{L,k,t}$ is related to $R_{k,t}$ as

$$R_{L,k,t} = V_{M,k} R_{k,t} = \frac{M_{M,k}}{\rho_k} R_{k,t},$$

where $V_{M,k}$ is the molar volume in m^3/mol; $M_{M,k}$ is the molar mass in kg/mol; and ρ_k is the density in kg/m^3 of the mineral phase.

The specific surface area of the mineral is defined as $A_{S,k} = A_k/m_k$ in (m^2/kg) or $A_{V,k} = A_k/V_k$ in $(1/m)$. Upon growth or dissolution, both $A_{S,k}$ and $A_{V,k}$ values vary with time because of changing particle size, shape, and surface roughness. Hence, specific surface areas must be corrected after each time step, either internally in TKinMet functions, or externally by the reactive transport model.

Some parameters, e.g. the dissolution rate constant, the activation energy, the reaction type and order constants for parallel mechanisms, can be considered as chemical properties of the solid phase, kept in the respective Phase definition record in the GEM-Selektor project database or in the GEMS3K input file. Other, "non-chemical" parameters, such as the reactive specific surface area assigned to r-th mechanism, are related to evolving particle or pore size and shape distributions. Such varying parameters should come at each time step into TKinMet calculations from the transport part of the coupled RTM code.

4.2.3 Numerical solution procedure

The resulting system of equations (4.6), (4.7) and (4.8) is solved using a hybrid finite element - finite volume method [14] implemented within the CSMP++ software library [15]. The SAMG solver is used for solving the arising systems of linear algebraic equations [16]. The chemical speciation calculations are performed using the GEMS3K software library [49].

My new reactive transport code is written in the C++ programming language. It has a flexible modular structure and can be configured for particular simulation needs: stationary or transient pressure with/without gravity, constant, stationary or transient temperature, implicit or explicit time stepping. Reactive transport simulations based on 1D, 2D and 3D unstructured grids can be performed.

A consistent initialization is important for RTM. The initial chemical composition/speciation is read from the text-format input files exported from the GEM-Selektor code package [49]. Initial and boundary conditions for pressure and temperature are read from a CSMP++ configuration file. Dirichlet, Neumann and mixed boundary conditions for pressure and temperature, as well as fluid and heat sources can be specified. First, initial pressure and temperature distributions across the model are calculated. Next, chemical speciation is computed for each grid node at initial pressure-temperature conditions. Finally, fluid properties are computed from the equation of state for the given pressure, temperature and salinity.

After the model initialization, the main time loop is executed. A flow chart for a single time step is shown in Figure 4.1, with a detailed description given in the following subsections.

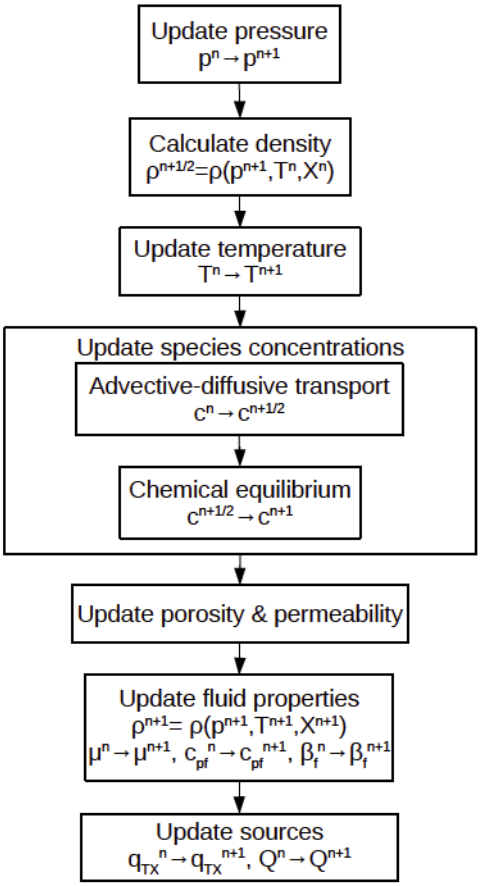

Figure 4.1: Flowchart of a single time step as implemented in CSMP++GEM

Pressure-temperature coupling

The pressure equation (4.6) and the heat transport equation (4.7) are solved in a sequential order implicitly in time, as represented by the following semi-discrete equations (4.19) and (4.20) in which constant properties lack upper indices. The rule for the porosity/permeability update is explained subsequently. The pressure equation is solved using the finite element method:

$$\phi^n \rho^n (\beta_f^n + \beta_\phi) \frac{p^{n+1} - p^n}{\Delta t} = \nabla \cdot \left(\left(\rho \frac{k}{\mu} \right)^n (\nabla p^{n+1} - \rho^n \mathbf{g}) \right) + q_{TX}^n + Q^n, \quad (4.19)$$

where p^0 is the initial pressure distribution, $\rho^0,\ \beta_f^0,\ \mu^0$ are the initial fluid properties, $\phi^0,\ k^0$ are the initial rock properties, $q^0 = 0,\ Q^0 = 0$.

After the pressure calculation, the density is updated from the equation of state [54]: $\rho^{n+1/2} = \rho(p^{n+1}, T^n, X^n)$ and stored for the further calculation of q_{TX}.

The heat transport equation with the diffusive term is solved using the finite volume method on a complementary sub–grid [64]:

$$\begin{aligned} (\phi^n \rho^n\ c_{pf}^n + (1-\phi^n)\rho_r c_{pr}) \frac{T^{n+1} - T^n}{\Delta t} = \\ = \nabla \cdot \left((\phi^n K_f + (1-\phi^n) K_r) \nabla T^{n+1} \right) - \nabla \cdot (\mathbf{v} \rho^n\ c_{pf}^n\ T^{n+1}), \end{aligned} \quad (4.20)$$

with T^0 – initial temperature distribution.

After the pressure-temperature calculations, the transport-chemistry calculations are performed. Subsequently, the fluid properties: $\rho^{n+1}\ c_{pf}^{n+1}$, μ^{n+1}, β_f^{n+1} are updated using the equation of state, and new values for q_{TX}^{n+1} and Q^{n+1} are calculated.

Transport-chemistry coupling: Sequential Non-iterative Approach

The transport equation (4.8) can be solved using the Sequential Non-Iterative Approach (SNIA) [31] in two steps:

$$\phi^n \frac{(c_i^{n+1/2} - c_i^n)}{\Delta t} + \nabla \cdot (\mathbf{v} c_i^{n+1/2}) - \nabla \cdot (D \nabla c_i^{n+1/2}) = 0, \quad (4.21)$$

$$\phi^n \frac{(c_i^{n+1} - c_i^{n+1/2})}{\Delta t} = q_i^{n+1},\ \forall i \in \overline{1, N}. \quad (4.22)$$

This approach allows to use the chemical partial equilibrium solver as a "black box" to calculate the values for the chemical source q_i. The source term q_i in (4.22) can be expressed as:

$$q_i^{n+1} = -\phi^n \frac{f_i^{n+1} - f_i^n}{\Delta t},$$

where f_i is the amount of i-th IC in the solid phase per unit pore volume. The negative sign indicates that, as the amount of IC in the solid phase decreases, its amount in the aqueous phase increases, and vice versa.

Calculations are performed in the following way. First, equations (4.21) are solved using the finite volume method. For each aqueous concentration c_i, a transport equation is solved on the whole grid. Subsequently, equations (4.22) are solved using the GEM IPM3 algorithm implemented in the GEMS3K code. Chemical speciation calculations are performed for each finite volume independently. The new values for c_i^{n+1} are computed from:

$$(\mathbf{c}^{n+1}, \mathbf{f}^{n+1}) = F(\mathbf{c}^{n+1/2} + \mathbf{f}^n),$$

where F denotes the GEMS3K solver that takes as an input the vector of total concentrations (the sum of aqueous and solid concentrations) and yields the vectors of aqueous and solid concentrations: $\mathbf{c} = (c_1 \dots c_{IC})$ and $\mathbf{f} = (f_1 \dots f_{IC})$, respectively.

A new CSMP++ data structure – the Array Variable – was implemented to store the vectors $\mathbf{c}$ and $\mathbf{f}$ associated with the nodes. This tree–like structure allows efficient advective-diffusive transport and chemical speciation computations.

Porosity/permeability feedback from reactions

The complete discrete form of the equation for species transport would be:

$$\frac{\phi^{n+1}c_i^{n+1} - \phi^n c_i^n}{\Delta t} + \nabla \cdot (\mathbf{v}_i c_i^{n+1}) - \nabla \cdot (D \nabla c_i^{n+1}) = -\frac{\phi^{n+1} f_i^{n+1} - \phi^n f_i^n}{\Delta t}. \quad (4.23)$$

In this work, we use a simplification by assuming porosity to be constant during the transport/chemistry computations, and solving the following equation instead:

$$\phi^n \frac{\tilde{c}_i^{n+1} - c_i^n}{\Delta t} + \nabla \cdot (\mathbf{v}_i \tilde{c}_i^{n+1}) - \nabla \cdot (D \nabla \tilde{c}_i^{n+1}) = -\phi^n \frac{\tilde{f}_i^{n+1} - f_i^n}{\Delta t}.$$

Following the transport step and the chemical equilibration step, the porosity is updated using the formula:

$$\phi^{n+1} = 1 - V_{inert} - \sum_{i=1}^{N_{min}} V_{min_i}^{n+1},$$

where V_{inert} is the volume fraction of the non-reactive rock , $V_{min_i}^{n+1}$ is the new volume fraction of the i-th mineral, calculated by multiplication of f_i^{n+1} by the mineral molar volume, N_{min} is the number of minerals.

The updated value of permeability is calculated using the Kozeny-Carman correlation [65]:

$$k^{n+1} = k^n \frac{(1-\phi^n)^2}{(1-\phi^{n+1})^2} \frac{(\phi^{n+1})^3}{(\phi^n)^3} \tag{4.24}$$

The new porosity and permeability values are used in the pressure calculation (4.19) at the next time level.

In order to maintain the mass balance, the species concentrations c_i and f_i are rescaled with respect to the new porosity, before being used in the transport calculations in the following time step:

$$c_i^{n+1} = \frac{\phi^n}{\phi^{n+1}} \tilde{c}_i^{n+1},$$
$$f_i^{n+1} = \frac{\phi^n}{\phi^{n+1}} \tilde{f}_i^{n+1}.$$

Calculation of q_{TX} and Q

After the transport and chemistry calculations are finished, source terms can be determined. The fluid expansion source is calculated from the change in fluid density due to temperature and salinity changes:

$$q_{TX}^{n+1} = -\phi^{n+1} \frac{\rho^{n+1} - \rho^{n+1/2}}{\Delta t} =$$
$$= -\phi^{n+1} \frac{\rho(p^{n+1}, T^{n+1}, X^{n+1}) - \rho(p^{n+1}, T^n, X^n)}{\Delta t},$$

where density is calculated from the equation of state.

The chemical source is calculated using the output data from GEMS3K:

$$Q^{n+1} = -\phi^{n+1} \frac{\sum_{i=1}^{N} (f_i^{n+1} - f_i^n) M_i}{\Delta t},$$

where $f_i^{n+1} M_i$ is the new mass of the i-th IC in the solid phase, $f_i^n M_i$ is its value at the previous time step.

Property values placement

As described above, porosity is updated based on the mineral amounts that are stored on the nodes, as a consequence of finite volume- (nodal-) based chemical calculations. There is no easy conservative way to interpolate or extrapolate porosity from the nodes to the elements and back. For this reason, we keep both the nodal and the

elemental porosity. For the permeability update, we first interpolate porosity to the elements, and then calculate the new permeability values using equation (4.24).

The transient pressure equation (4.6) is solved using linear finite elements. Within the finite element – finite volume framework of CSMP++, fluid properties are stored on the nodes (density, viscosity, fluid compressibility, fluid thermal expansion coefficient), whereas the material properties (rock compressibility, permeability) are stored on the elements. In order to increase the accuracy of the finite element solution of pressure equation, the following properties:

$$\rho\phi(\beta_f + \beta_\phi)$$

– the total system mass compressibility,

$$\lambda = \rho\frac{k}{\mu}$$

– the mass conductivity,

$$\rho\frac{k}{\mu}\rho\mathbf{g}$$

– the mass gravity term, are stored on the element integration points. This process is exemplified for a calculation of mass conductivity in the left part of the Figure 4.2.

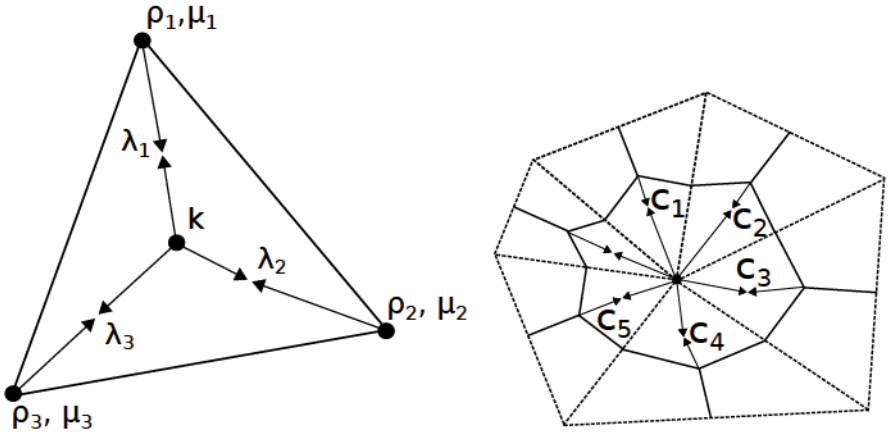

Figure 4.2: Mass conductivity is placed on finite element integration points (left). Total mass heat capacity is placed on the finite volume sector integration points (right). Finite elements are dashed line triangles, finite volume of the dual mesh is drawn in solid lines

The advection-diffusion heat transport equation (4.7) is solved using the finite volume method, where the finite element integral corresponding to diffusion is accumulated into the same solution matrix [66].

Table 4.1: Calcite-dolomite benchmark: aqueous and solid boundary and initial concentrations

	Boundary	Initial	Units
Ca^{2+}	$1 \cdot 10^{-7}$	$1.22 \cdot 10^{-4}$	mol/kg_{water}
CO_3^{2-}	$1 \cdot 10^{-7}$	$1.22 \cdot 10^{-4}$	mol/kg_{water}
Mg^{2+}	$1 \cdot 10^{-3}$	$1 \cdot 10^{-7}$	mol/kg_{water}
Cl^-	$2 \cdot 10^{-3}$	$1 \cdot 10^{-7}$	mol/kg_{water}
$Calcite$	0.0	$2.17 \cdot 10^{-5}$	mol/kg_{solid}
$Dolomite$	0.0	0.0	mol/kg_{solid}

The total mass heat capacity, $C_t = \phi\rho c_{pf} + (1-\phi)\rho_r c_{pr}$, is stored on the finite volume sector integration points; the values for fluid mass heat capacity ρc_{pf} are taken upstream. Figure 4.2 illustrates calculation of the first, fluid–related term in total mass heat capacity ($\phi\rho c_{pf}$): porosity ϕ, density ρ and fluid heat capacity c_{pf} are placed on the nodes; each finite volume sector has one integration point where $c_i = \phi_i \rho c_{pf}$ is stored. The second, rock–related term ($(1-\phi)\rho_r c_{pr}$) is interpolated from the finite elements to sector integration points.

4.3 Benchmarking results

In order to verify our new reactive transport code, we first compared it to the OpenGeoSys-GEM coupled code (that also uses GEM-IPM3 as chemical solver) on well-known calcite dissolution – dolomite precipitation benchmark from [51], which ignores mineral dissolution/precipitation kinetics.

After that, we benchmarked CSMP++GEM against TOUGHREACT on a 1D dolomitisation model that accounts for kinetics of dolomite.

4.3.1 Dolomitisation by $MgCl_2$ with equilibrium reactions

The test model is a 1D porous medium column of $0.5\,m$ length, with the bulk density $\rho_b = 1800\,kg/m^3$ and the porosity $\phi = 0.32$. Fluid pressure of $1bar$ and temperature of 25 °C are assumed. The pore fluid is initially equilibrated with calcite. The column is flushed from left to right with $MgCl_2$ solution at a flow rate $q = 3 \cdot 10^{-6}\,m/s$, with a diffusion-dispersion coefficient D of $2 \cdot 10^{-8}\,m^2/s$. Initial and boundary concentrations for aqueous species and minerals are presented in Table 4.1. Chloride does not react with solids, but serves as a tracer. Both calcite and dolomite are equilibrium-controlled minerals.

The model domain is discretized into 100 elements of equal length ($\Delta x = 0.005\,m$) and a time step $\Delta t = 200\,s$ is used in the simulation. As the reaction front progresses, dolomite is formed temporarily as a moving zone, and calcite is dissolved. Simulation results after 21000s, compared with the results from [51] for the concentrations of ions and minerals, are presented in Fig. 4.3 and 4.4 respectively. The results are consistent; minor deviations are due to different numerical methods that were used for transport calculations: the finite element method (OpenGeoSys) and the finite volume method (CSMP++).

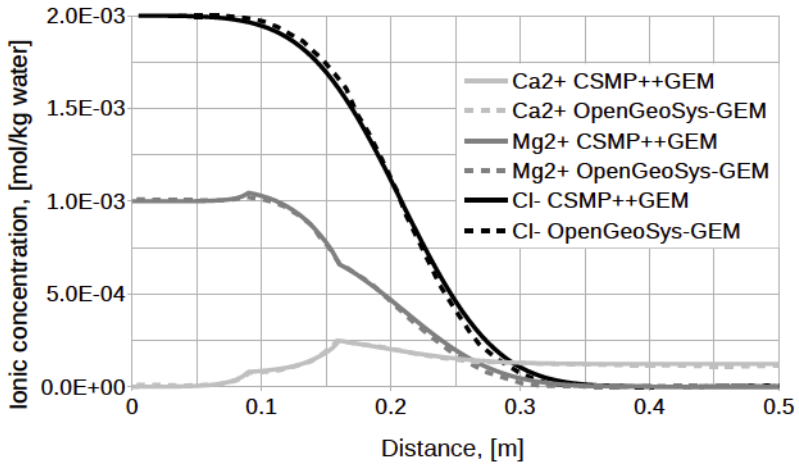

Figure 4.3: The calcite-dolomite benchmark: concentrations of Ca^{2+}, Mg^{2+} and Cl^{-} ions after 21000s

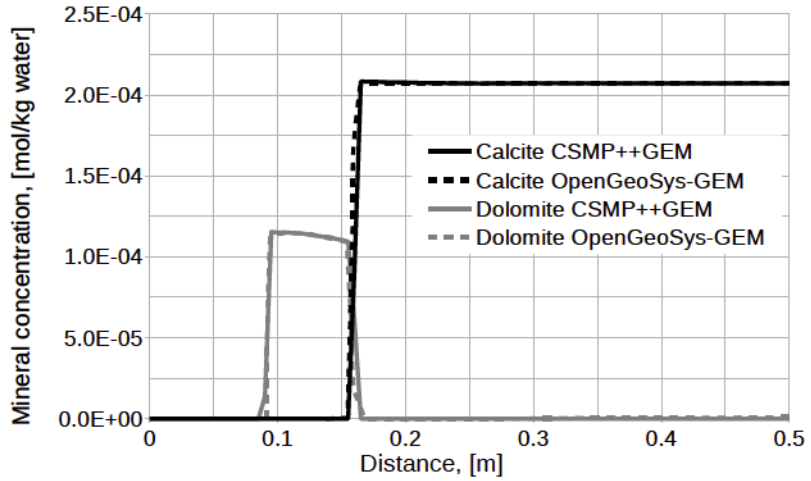

Figure 4.4: The calcite-dolomite benchmark: concentrations of calcite and dolomite after 21000s

Table 4.2: Rock properties

porosity	0.4	-
permeability	$1 \cdot 10^{-12}$	m^2
rock compressibility	$1 \cdot 10^{-10}$	Pa^{-1}
rock density	2710	kg/m^3
rock specific heat capacity	1000	$J/kg \cdot K$
total thermal conductivity	2	$W/m \cdot K$
diffusion-dispersion coefficient	$3 \cdot 10^{-11}$	m^2/s

4.3.2 Dolomitisation by seawater with mineral kinetics

Following the previous work on 1D reactive transport simulations of dolomitisation in reflux systems using TOUGHREACT [38], we created a more realistic 1D benchmark in order to compare these results with those obtained using the CSMP++GEM code. The goal was to simulate changes in mineralogy that a calcite column undergoes during the flow-through of seawater, taking into account the kinetics of dolomite.

Description

The simulation model is a $10m$–long vertical column, with a cross section of $1m^2$, divided into 50 cells ($\Delta x = 0.2m$). Rock properties are listed in the Table 4.2. The initial mineral composition was 99% calcite and 1% dolomite. Calcite was under equilibrium control, whereas ordered dolomite was under kinetic control.

The normal seawater composition (0.35% salinity), taken from Nordstrom [67], is supersaturated to both calcite and dolomite. The initial water composition was derived from this modern seawater by equilibrating it with calcite and ordered dolomite. The boundary water injected at the top of the column was modern seawater equilibrated with calcite only. Because the precipitation rate of dolomite is very slow at these conditions (5-6 orders of magnitude less than the dissolution rate of calcite or dolomite), we assumed that at the required large time step lengths, calcite would dissolve or precipitate instantly, and thus should be considered as an equilibrium-controlled phase.

A thermodynamic database suitable for both CSMP++GEM and TOUGHREACT was not available, and therefore we used two different databases presenting quite close equilibrium constants for dolomite and calcite (see Table 4.3). The PSI/-Nagra thermodynamic database ([68]) was used to prepare the CSMP++GEM input in GEM-Selektor and in the simulation runs, whilst the THERMODDEM database ([69]) was used in TOUGHREACT simulations.

Table 4.3: Thermodynamic data comparison: equilibrium constants at 1 bar, 25 °C

logK	PSI/Nagra	THERMODDEM
$CaCO_3$	1.8490	1.8470
$CaMg(CO_3)_2$	3.5680	3.5328

Table 4.4: Initial water compositions for CSMP++GEM and TOUGHREACT runs at 1 bar, 30 °C: basic species molalities

	CSM++GEM	TOUGHREACT
pH	7.117	7.037
Ca^{2+}	$3.874 \cdot 10^{-2}$	$3.666 \cdot 10^{-2}$
Mg^{2+}	$2.644 \cdot 10^{-2}$	$2.871 \cdot 10^{-2}$
HCO_3^-	$1.594 \cdot 10^{-3}$	$1.652 \cdot 10^{-3}$
Na^+	$4.839 \cdot 10^{-1}$	$4.854 \cdot 10^{-1}$
K^+	$1.055 \cdot 10^{-2}$	$1.058 \cdot 10^{-2}$
SO_4^{2-}	$2.917 \cdot 10^{-2}$	$2.926 \cdot 10^{-2}$
Cl^-	$5.649 \cdot 10^{-1}$	$5.657 \cdot 10^{-1}$

The extended Debye-Huckel activity model with parameters derived by Helgeson et al. [70] was used in both software packages. Initial and boundary water compositions for CSMP++GEM and TOUGHREACT at 30 °C are listed in Table 4.4 and Table 4.5.

Kinetic rate of dolomite precipitation was taken in the following form:

$$r = \kappa A(1 - \Omega^\theta)^\eta,$$

where κ is the rate constant, A is the reactive surface area, Ω is the mineral saturation ratio, θ and η are empirical parameters, and the values of corresponding parameters,

Table 4.5: Boundary water compositions for CSMP++GEM and TOUGHREACT runs at 1 bar, 30 °C: basic species molalities

	CSMP++GEM	TOUGHREACT
pH	7.626	7.528
Ca^{2+}	$1.040 \cdot 10^{-2}$	$1.041 \cdot 10^{-2}$
Mg^{2+}	$5.489 \cdot 10^{-2}$	$5.508 \cdot 10^{-2}$
HCO_3^-	$1.710 \cdot 10^{-3}$	$1.913 \cdot 10^{-3}$
Na^+	$4.839 \cdot 10^{-1}$	$4.854 \cdot 10^{-1}$
K^+	$1.055 \cdot 10^{-2}$	$1.058 \cdot 10^{-2}$
SO_4^{2-}	$2.917 \cdot 10^{-2}$	$2.926 \cdot 10^{-2}$
Cl^-	$5.649 \cdot 10^{-1}$	$5.657 \cdot 10^{-1}$

Table 4.6: Kinetic rate parameters for ordered dolomite

	CSMP++GEM	TOUGHREACT	
k_{25}	-	$4.58 \cdot 10^{-19}$	$mol/m^2 \cdot s$
$\tilde{k}_{25}$	10000	-	$mol/m^2 \cdot s$
Λ	11.22	-	-
E_a	$133.47 \cdot 10^3$	$133.5 \cdot 10^3$	J/mol
k at 30 °C	$1.129 \cdot 10^{-18}$	$1.113 \cdot 10^{-18}$	$mol/m^2 \cdot s$
A	1000	1000	m^2/kg
η	2.26	2.2	-
θ	1	1	-

were taken from Arvidson and Mackenzie [45]. This ensures that simulations are consistent with previous RTM simulations of dolomitisation [38, 40, 36, 37].

TOUGHREACT has the following built-in temperature correction for the rate constant:

$$\kappa = \kappa^o e^{\frac{-E_a}{R}}(\frac{1}{T} - \frac{1}{298.15}),$$

whereas in CSMP++GEM we used a slightly different but equivalent formulation:

$$\kappa = \tilde{\kappa}^o \Lambda e^{\frac{-E_a}{RT}},$$

where E_a is the activation energy, κ^o and $\tilde{\kappa}^o$ are rate constants at 25 °C, Λ is the Arrhenius parameter, R is the universal gas constant. Kinetic parameters for dolomite precipitation are listed in Table 4.6, and values for the rate constant at 30 °C are compared. A constant reactive surface area of $1000 m^2/kg$ was assumed, corresponding to small dolomite rhombs ($2.5 \mu m$).

The system was assumed to be isothermal; simulations were performed at 30, 40 and 50 °C. Dirichlet boundary conditions for pressure at the top and the bottom of the column were assigned, resulting in a flow rate of $\sim$ $1 m/yr$ ($2.79 \pm 0.25 \times 10^{-8} m/s$). Essential conditions for the flow simulation are listed in Table 4.7. In TOUGHREACT runs, top and bottom cells were infinite volume cells; in CSMP++GEM, Dirichlet boundary conditions were assigned for species concentrations at both column ends. In CSMP++GEM simulations, thermophysical properties of seawater were taken from the equation of state for brine from [54]. In TOUGHREACT, the EOS7 module (water, brine, air) was used [71].

In our simulations, we used two different time steps. In the main time loop, we used a time step of 10 years, and in the inner loop (solute transport and chemistry)

Table 4.7: Essential conditions for the flow simulation

initial pressure	101325 Pa
initial temperature	30 °C
pressure top	101623 Pa
pressure bottom	201375 Pa

a time step duration was chosen according to the Courant–Friedrichs–Lewy (CFL) condition. This is a necessary condition for convergence of SNIA, as with a $\Delta t = CFL$ the fluid will move no more than one cell at a time, so it is guaranteed that in the subsequent chemistry calculation, the fluid will react with the rock before it leaves the cell.

Results

We performed three sets of simulations for 30, 40 and 50 °C in CSMP++GEM and TOUGHREACT, and compared the results after 10kyrs. Figures 4.5 and 4.6 show the results at 50 °C. Within the first meter of the column some boundary effects occurred, due to the fact that the boundary water was slightly undersaturated with respect to calcite and therefore a higher amount of calcite (compared to the rest of the column) dissolved in the first few cells adjacent to the column top.

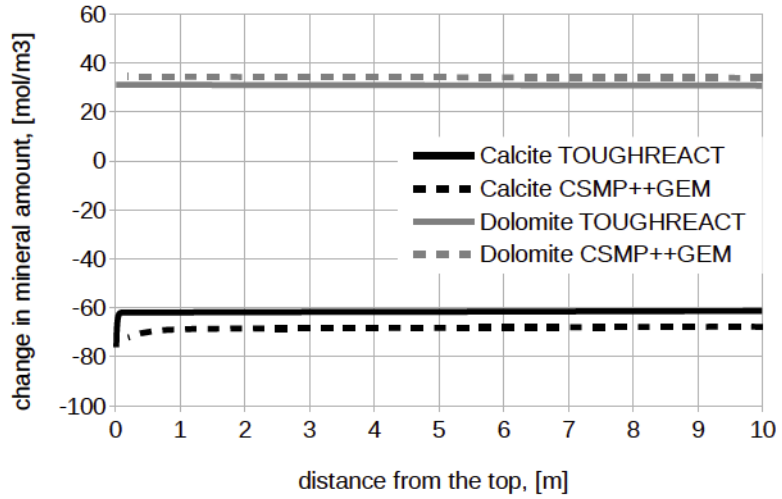

Figure 4.5: Results of the simulation at 50 °C: changes in mineral amounts after 10kyrs

Results using the two different simulators are similar and minor differences can be explained by (1) different numerical methods for flow and transport (finite difference method in TOUGHREACT and finite element - finite volume method in

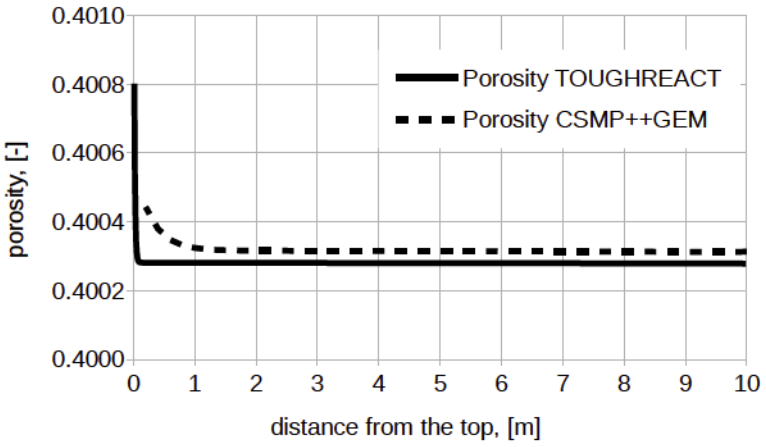

Figure 4.6: Results of the simulation at 50 °C: porosity after 10kyrs

Table 4.8: Amount of calcite disssolved and dolomite precipitated $[mol/m^3]$ in simulations at three different temperatures: average values across the column after 10kyrs

	Calcite		Dolomite	
	CSMP++GEM	TOUGHREACT	CSMP++GEM	TOUGHREACT
30 °C	0.91	0.79	0.46	0.39
40 °C	8.03	7.06	4.03	3.55
50 °C	68.1	61.5	34.2	31.0

CSMP++GEM), (2) different numerical methods for chemical reactions (Law of Mass Action in TOUGHREACT and Gibbs energy minimization in CSMP++GEM), (3) different thermodynamic databases and small differences in aqueous activity models and mineral kinetic rate models, (4) differences in equations of state for the aqueous fluid.

Apart from the first meter from the top, the changes in mineral amounts are almost constant across the column. After the first few years of injection, the water composition becomes approximately the same along the whole column. Consequently, the kinetic rate of dolomite precipitation and the resulting rate of calcite dissolution are constant, indicating flow rates are high relative to reaction rates. The amount of dissolved calcite is approximately two times greater than the amount of precipitated dolomite, consistent with the stoichiometric assumption, i.e. that two moles of calcite are consumed to form one mole of dolomite.

The porosity evolution is shown in Fig. 4.6. After such a short (in geological time scale) period of time, the change in porosity is minor; the slight increase is due to the differences in molar volumes of calcite and dolomite.

Table 4.9: Change in porosity in simulations at three different temperatures: average values across the column after 10kyrs

	CSMP++GEM	TOUGHREACT
30 °C	$4.6 \cdot 10^{-6}$	$3.9 \cdot 10^{-6}$
40 °C	$3.7 \cdot 10^{-5}$	$3.3 \cdot 10^{-5}$
50 °C	$3.3 \cdot 10^{-4}$	$2.8 \cdot 10^{-4}$

Table 4.10: Saturation indices of dolomite for boundary water at calcite equilibrium at different temperatures

	CSMP++GEM	TOUGHREACT
30 °C	0.8837	0.8885
40 °C	0.9704	0.9671
50 °C	1.0515	1.04

Tables 4.8 and 4.9 compare the results for simulations at 30, 40 and 50 °C, presenting the average values of calcite dissolved, dolomite precipitated, and porosity increase across the column after 10kyrs of simulation time. The changes of mineral amounts and change in porosity increase by a factor of 8-9 with a change of 10 degrees in temperature. These results agree with the saturation indices for dolomite (boundary water) at different temperatures (Table 4.10), and with the acceleration of dolomite precipitation rates at increasing temperature (eq. 4.18). At all three temperatures, calcite dissolution is driven by dolomite growth, and the ratio between the amount of calcite dissolved and dolomite precipitated remains approximately equal to 2. After such a short time calcite remains dominant (98.6% of mineral phase after 10 kyrs) and dolomitisation rate is limited by reactive surface area.

For the 50 °C case, we ran a simulation for 200 kyrs, and compared the results in two cells at distances of 1 and 5 meters from the top of the column (see Fig. 4.7 and 4.8). The time of the complete calcite dissolution increases with the distance from the top of the column. The highest porosity value coincides with the time to total calcite replacement, after which porosity starts to decrease.

After a period of very slow dolomitisation at the start of the replacement phase, the dolomite precipitation rate increases non-linearly with the increase in reactive surface area (the dolomite saturation index stays constant) and porosity increases from 40% to 47% in CSMP++GEM and to 50% in TOUGHREACT from an initial value of 40%. Post-replacement dolomite cement is observed down to 5m depth from the injection point. During the overdolomitisation phase (primary precipita-

tion of dolomite cement), porosity reduces to 42% in CSMP++GEM and to 47% in TOUGHREACT at the distance of 1m from the top of the column, whilst there is barely any change (< 1%) in the middle of the column.

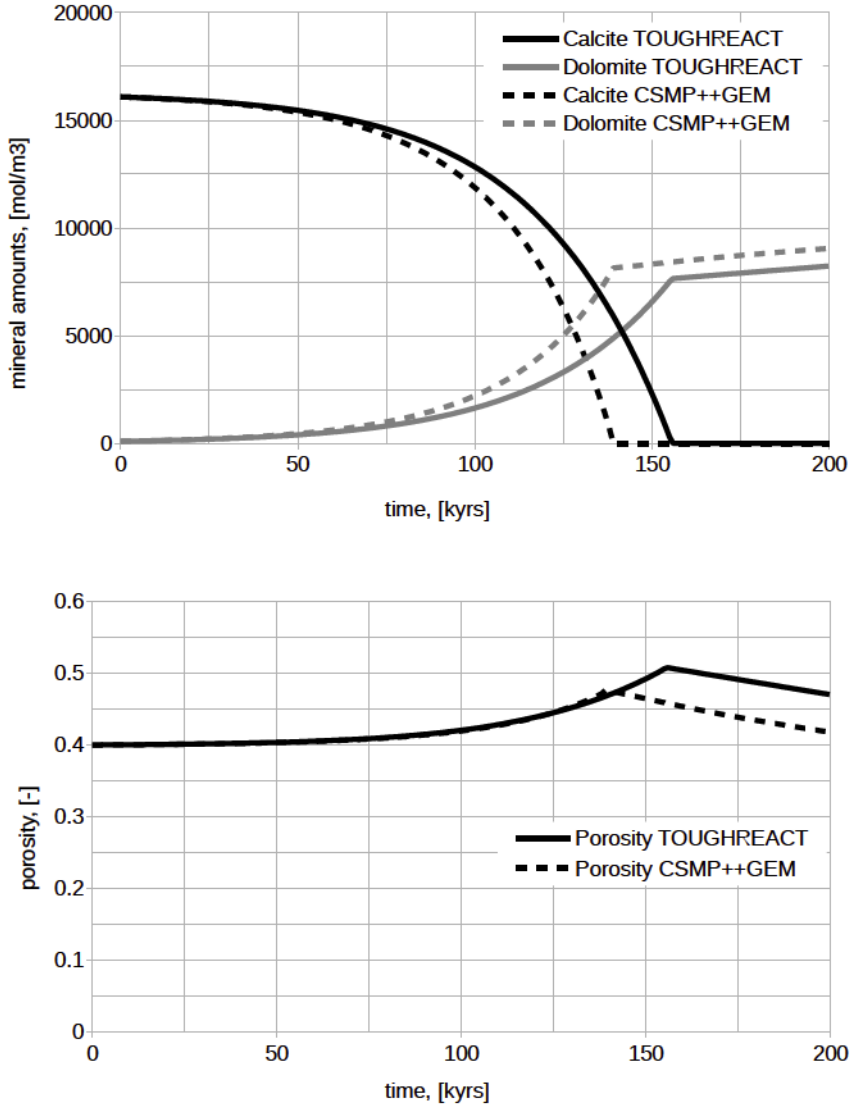

Figure 4.7: Results of the simulation at 50 °C: changes in mineral amounts (top) and changes in porosity (bottom) plot over time of 200kyrs at x=1m from the column top

Although our results are in a good agreement during the first 50 kyrs of simulation time (replacement of the first 2-3% of calcite), they progressively diverge thereafter as the precipitation rate increases with the increasing dolomite reactive surface area and the differences between the kinetic rates in two codes get more prominent. The time point of complete calcite replacement is about 25kyrs delayed for the TOUGHREACT

simulation compared to CSMP++GEM. In the middle of the column the match between the two codes is closer than at the distance of 1m.

This example shows how even small differences in numerical methods, activity models and kinetic rates can lead to significant differences between model predictions over geological times.

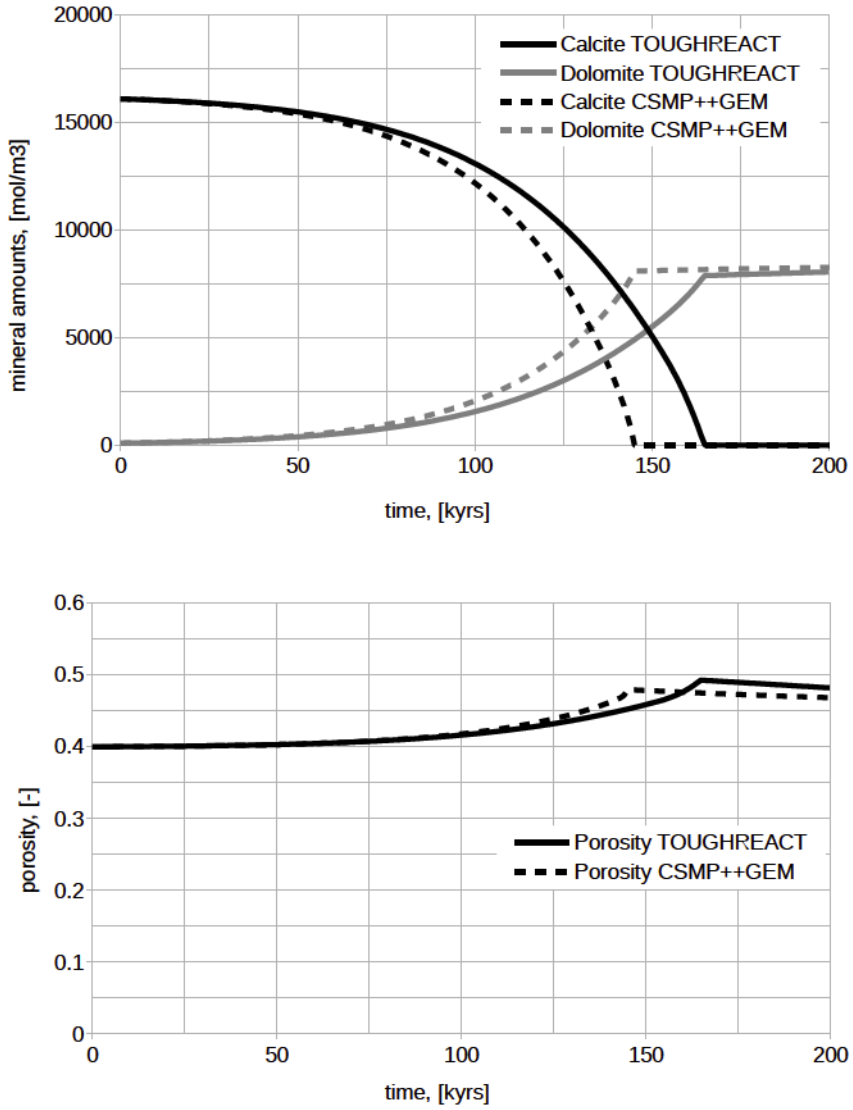

Figure 4.8: Results of the simulation at 50 °C: changes in mineral amounts (top) and changes in porosity (bottom) plot over time of 200kyrs at x=5m from the column top

4.4 Conclusions

A new reactive transport code CSMP++GEM was developed and benchmarked against OpenGeoSys-GEM and TOUGHREACT calculations. Small differences (especially regarding calcite dissolution) are due to differences in numerical methods for flow (FE-FV vs FD) and chemistry (LMA vs GEM), equations of state, kinetic rate models.

Calcite dissolution is a reactive transport phenomenon driven by a slow dolomite precipitation. The models show that the rate of dolomitisation by replacement of calcite increases by factor of 9 with an increase of 10 °C in temperature.

The CSMP++GEM code correctly represents all qualitative effects of dolomitisation. At the replacement stage, the amount of precipitated dolomite is equal to the half of the dissolved calcite; porosity increases in the whole model during the mole per mole replacement, but is subsequently plugged by dolomite cementation in the first few cells close to the model top.

Our results demonstrate the importance of running reactive transport simulations with the time step constrained by the CFL condition, as otherwise the resulting mineral amounts may be overestimated (smaller time steps) or underestimated (large time steps). The use of the Sequential Iterative Approach can increase solution accuracy, but the simulation time will grow proportionally to the number of SIA iterations. Adaptive time stepping, if implemented in the numerical integration of kinetic rates, similar to [72], might make the numerical solution more stable.

Kinetic rates of mineral precipitation/dissolution depend on the mineral saturation index, time and mineral reactive surface area. Specific surface area correction upon growth or dissolution dependent on the particle/pore size distribution, particle/pore size evolution and shape factor should be implemented in the future, which will make the model more realistic.

Our results illustrate the challenges faced when comparing RTM software that uses different methods for transport and chemistry and different thermodynamic databases. We believe that our match is reasonably close, but it can be seen that the discrepancy grows proportional to the increasing amount of precipitated dolomite.

The new coupled CSMP++GEM code can now be used to simulate more complex systems that include reactive transport and flow in faults. It is therefore suitable for modelling of the hydrothermal dolomitisation.

Chapter 5

Reactive transport modelling of dolomitisation of the Benicàssim case study

The new coupled reactive transport code CSMP++GEM was applied to study the fault-controlled dolomitisation at the Benicàssim outcrop.

This section contains an overview of this Benicàssim case study and previous research that was conducted there. After that two sets of reactive transport simulation results are presented: a temperature sensitivity study on a homogeneous model and a comparison between two different flow scenarios. Reactive transport modelling in this area was previously conducted by Corbella et al. [43] using a different model, that did not have a permeability update due to porosity evolution.

5.1 Introduction

Hydrothermal dolomites form by ingress of Mg-rich fluids with temperature elevated (usually by 5-10$^{\circ C}$) relative to the host carbonate rock, mostly around faults or high-permeability zones, often in association with mineral ores or hydrocarbon reservoirs [73, 47]. A mole-per-mole replacement of calcite by dolomite would result in a volume loss, generating an increase in porosity up to 13% [44]. On the contrary, post-replacement dolomite cementation ("overdolomitisation" [74]) can occlude the pores and significantly reduce porosity. The porosity evolution of the rock is connected to its permeability evolution in a non-linear fashion. The flow velocity can be enhanced in the dolomitised region, or completely blocked if cement precipitation closes all pathways. Prediction of the spatial distribution of dolomite bodies and assessment

of their petrophysical properties is important for a predictive understanding of the recovery of oil and gas in carbonate reservoirs.

Reactive transport modelling (RTM) has been applied to study the formation of dolomites in low-temperature reflux systems [36, 37, 38, 39] and early burial dolomitisation by geothermal convection [40, 41], as well as structurally controlled hydrothermal dolomitisation [75]. In order to show the capabilities of the newly created CSMP++GEM reactive transport code, hydrothermal dolomite formation on a realistic geometry, based on geological and geochemical data from the Benicàssim case study, was simulated. The Benicàssim outcrop is an excellent example of fault-controlled hydrothermal dolomitisation that was extensively studied before [43, 76, 77, 78].

In this section we present the results of the RTM simulations in the Benicàssim study area, performed using the CSMP++GEM code and based on the previous research of the study area, investigating the controls of temperature and flow via the permeability distribution on dolomitisation.

5.1.1 The Benicàssim case study

The Benicàssim area is located in the southern part of the Maestrat Basin, on the east coast of Spain (see Fig. 5.1). There are two major sets of extensional large-scale faults that controlled the sediment deposition and fluid circulation in this area: the Campello (NW-trending) and Benicàssim (NE-trending) faults. The Lower Cretaceous (Aptian-Albian) Benassal Formation cropping out in the Benicàssim area is a 1500 m thick succession consisting almost entirely of shallow-marine carbonates which have been partially replaced by dolomites [43].

The spatial distribution of the dolomitised geobodies was mapped, the rocks were characterized and a sequence stratigraphic analysis was performed by Martin-Martin et al. [77]. Reactivity of dolomitising fluids was studied and the possible sources of magnesium were evaluated by Gomez-Rivas et al. [76], as well as the various flow scenarios that can provide the plausible driving mechanism for dolomitisation [78]. Despite these extensive studies, the mechanisms of dolomite formation in the Benicàssim area are still not clear, nonetheless the provided data gives robust boundary conditions for a modelling study.

Dolomitisation in the Benicàssim area is a fault-controlled process [77]. The stratabound dolostone geobodies, which develop preferentially in high-permeability grain-dominated facies, are up to $150\,m$ thick and extend up to $7\,km$ away from the fault zones [43]. Analysis based on field observations and regional geology suggests

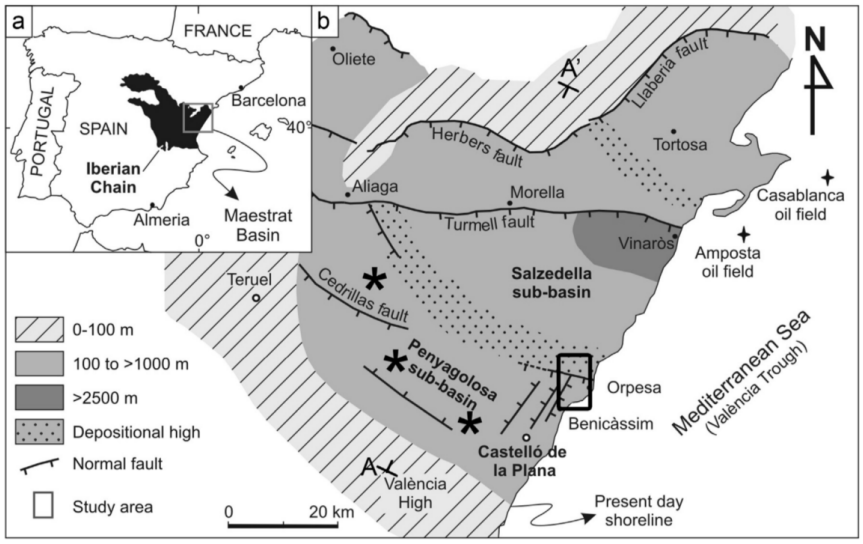

Figure 5.1: (a) Simplified map of the Iberian Peninsula showing the location of the Iberian Chain and Maestrat basin; (b) Paleogeographic map of the Maestrat basin during the Late Jurassic - Early Cretaceous rifting cycle showing thickness of syn-rift deposits and main fault traces. The black square indicates the location of the study area, while the black asterisks show the location of dolostones of the same age and type in the Maestrat basin. Figure taken from Gomez-Rivas et al. [76]

that the dolomitisation occured at burial depths less then $1000\,m$ [77]. The dolostones in Benicàssim are interpreted to be of hydrothermal origin with replacement temperatures higher than $60\,°C$ [77, 78].

There are two major requirements for dolomitisation: a driving mechanism for the fluid flow and a source of fluids rich in magnesium. Fluid and heat flow simulations performed by Gomez-Rivas et al. [78] suggest that a long-term fluid circulation is the most plausible fluid flow scenario as it provides the flow rates (meters per year to tens of meters per year) as well as the temperature gradient necessary for the dolomitisation in Benicàssim over a long period of time. Geochemical and geological data suggests that seawater or seawater-derived brines served as a source of magnesium for dolomitisation and that the most likely fluid drive mechanism was thermal convection which also focused fluids through seismic-scale faults. From the faults the fluids would have spread and flowed along high-permeability beds. The dolomitising brines could have been generated by interaction of the infiltrated seawater with basement fluids [76].

The stratabound geometry of the dolomite bodies is explained by the permeability differences between layers, due to differences in the original depositional facies and early diagenesis [76, 43]. This conceptual model for dolomitisation in the Benicàssim

area was proposed by Corbella et al. [43] and is shown in Figure 5.2.

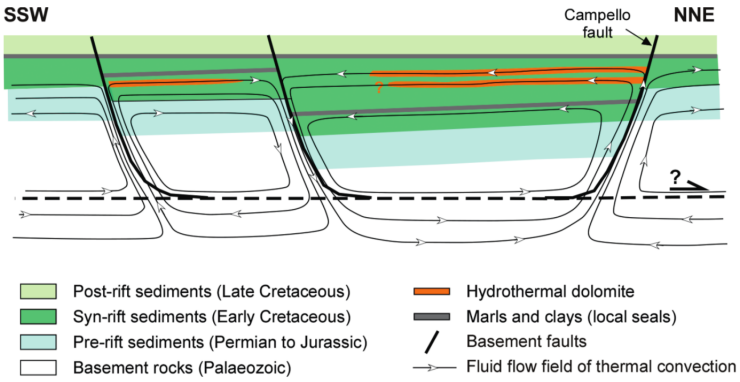

Figure 5.2: Sketch illustrating the conceptual model for the genesis of the Benicàssim and Maestrat Basin dolomitisation. Thermal convection during the Late Cretaceous post-rift period was the driving force for the transportation of seawater-derived fluids along faults and layers. These fluids also flowed through basement and pre-rift sediments. Fluid fluxes would have been higher in high-permeability beds, which were preferentially dolomitised. The model is approximately 25 km long and 6 km thick. The dashed line indicates the detachment level of large-scale faults. Figure taken from Corbella et al. [43]

5.2 Methodology

The new CSMP++GEM reactive transport code was applied to simulate dolomitisation in the Benicàssim case study. CSMP++ allows flow simulations with transient pressure including gravity, the use of a mass conservative transport scheme and an accurate equation of state for saline water, and calculation of the porosity/permeability evolution feedback. GEM is responsible for precise chemical speciation calculations at different temperatures and pressures, taking into account kinetics of mineral dissoluion/precipitation. The two codes, coupled together, provide a powerful tool for realistic reactive transport simulations on unstructured grids.

5.2.1 Reactive transport model setup

The model geometry was adapted from Corbella et al. [43] (see Fig. 5.3). It is a 2D rectangular model, 1310 m long and 190 m high, with a 22.5 m wide permeable fault in the middle and horizontal layers with a thickness of 45, 30, 40, 30 and 45 m.

The CAD model was meshed in ANSYS ICEM CFD. Two triangular meshes were created, a coarse and a fine one (see Fig. 5.4). The coarse mesh consists of 1128

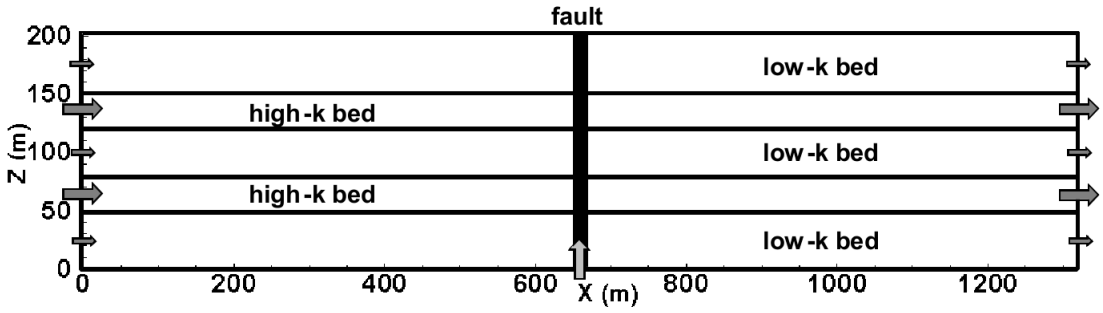

Figure 5.3: Idealized 2D model geometry of Benicàssim. Figure taken from Corbella et al. [43]

Table 5.1: Rock properties for the Benicàssim model

	Mudstone–wackestone	Packstone–grainstone	Fault
porosity, [%]	10	30	45
permeability, $[m^2]$	$1 \cdot 10^{-14}$	$3 \cdot 10^{-13}$	$5.8 \cdot 10^{-13}$

triangular elements with 629 nodes. The fine mesh has 4614 triangular elements with 2446 nodes.

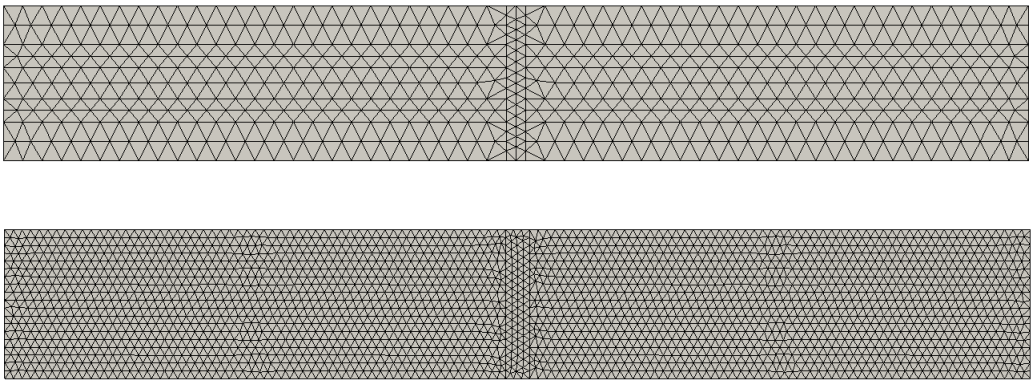

Figure 5.4: Coarse mesh (top) and fine mesh (bottom)

Rock property values based on petrographical observations for mudstone– wackestone, packstone–grainstone and the fault zone were adapted from Corbella et al. [43]. Porosity and permeability values are presented in Table 5.1.

The system was assumed isothermal and simulations were conducted at 70 and 100 °C and an initial pressure of 110 bar. Initial and injected (boundary) waters were derived from normal modern seawater from Nordstrom et al. [67]. For the initial water composition seawater was equilibrated with calcite and dolomite. For the boundary

Table 5.2: Aqueous solution compositions for the Benicàssim model at 70 and 100 °C, 110 *bar*

	Boundary 70 °C	Boundary 100 °C	Units
SI dolomite	1.192	1.332	-
pH	7.418	7.343	-
$CO_{2(gas)}$	277	293	ppm
Ca^{2+}	$1.001 \cdot 10^{-2}$	$9.847 \cdot 10^{-3}$	mol/kg_w
Mg^{2+}	$5.49 \cdot 10^{-2}$	$5.49 \cdot 10^{-2}$	mol/kg_w
HCO_3^-	$1.053 \cdot 10^{-3}$	$6.19 \cdot 10^{-4}$	mol/kg_w
Na^+	$4.839 \cdot 10^{-1}$	$4.839 \cdot 10^{-1}$	mol/kg_w
K^+	$1.055 \cdot 10^{-2}$	$1.055 \cdot 10^{-2}$	mol/kg_w
SO_4^{2-}	$2.917 \cdot 10^{-2}$	$2.917 \cdot 10^{-2}$	mol/kg_w
Cl^-	$5.65 \cdot 10^{-1}$	$5.65 \cdot 10^{-1}$	mol/kg_w

water seawater was equilibrated with calcite only. In our case only the boundary water plays an important role as the initial formation water is displaced by injected water within the first years of simulation. Unlike in the LMA codes, the GEM input consists of the total mole amounts of Independent Components (chemical elements), but for the sake of representation simplicity we present it in molal total concentration of main ions and pH (see Table 5.2).

An initial mineral rock composition of 99% calcite and 1% dolomite was assumed. The rate of calcite dissolution is orders of magnitude higher than the rate of dolomite precipitation, especially at high temperatures, which is why in our simulations dolomite was a kinetically controlled mineral, while calcite was under thermodynamic control. The specific reactive surface area for dolomite was set as $10000\,cm^2/g$, which based on geometric calculations corresponds to fine rhombs of $2.5\mu m$ diameter.

An accurate calculation of the initial state is very important for reactive transport simulations. Initial model equilibration was performed in multiple steps. First, the pressure at the top model boundary was set to $110\,bar$ and the gravitational pressure distribution was calculated. Then fluid properties were updated from the equation of state and the chemical equilibrium was calculated in all the nodes. Second, pressure at the top boundary was released and specific boundary conditions for pressure were applied (dependent on the flow scenario). After that, the chemical equilibrium was calculated and fluid properties were updated one more time.

Simulations were carried out for time periods of tens of thousands of years with a time step of 10 years for the pressure loop and a CFL timestep for the transport-chemistry loop. Fluid properties were taken from the equation of state for brine

Table 5.3: Mean values of fluid properties across the model at 70 and 100 °C at initial pressure distribution

	70 °C	100 °C	Units
Density	1006.2	987.5	kg/m^3
Viscosity	$4.49 \cdot 10^{-4}$	$3.15 \cdot 10^{-4}$	$Pa \cdot s$

from [54].

Temperature influence on dolomitisation

Two simulations, at 70 and 100 °C were performed on the coarse mesh with homogeneous rock properties (packstone–grainstone, see Table 5.1) in order to assess the temperature influence on the dolomitising capacity of fluids. The rock properties on the left boundary were fixed during the entire simulation (porosity 30%, permeability $3 \cdot 10^{-13}\, m^2$, zero dolomite amount).

Both simulations started with an average lateral flow velocity of $1.66 \cdot 10^{-7}\, m/s$ ($5.24\, m/yr$, similar to the value of $6\, m/s$ used in Corbella et al.[43]). This flow velocity was achieved by applying a pressure gradient across the model by assigning a super hydrostatic pressure on the left and a hydrostatic pressure on the right model boundaries. Due to the significant fluid density and fluid viscosity differences at 70 and 100 °C (see Table 5.3) two different pressure difference values were used $\Delta p_{70\,^\circ\mathrm{C}} = 3.3\, bar$ and $\Delta p_{100\,^\circ\mathrm{C}} = 2.3\, bar$.

Fluid flow scenarios

The two fluid flow scenarios were compared in an attempt to reproduce the stratabound dolomitisation that is observed in the Benicàssim outcrop. These simulations were performed on the fine mesh with layered rock properties (after Corbella et al.[43]) at 100 °C as at this temperature the dolomitising capacity of seawater is the highest [43].

The rock properties on the left boundary were fixed during the entire simulation (porosity and permeability according to the layer properties, zero dolomite amount). Due to the fact that the chemistry calculations are node-based, the boundaries between the layers are going through the cells and there is a transition zone between each pair of layers, where the rock properties are averaged. For the porosity the volume based average was used. The use of harmonic average for permeability results in the transition zone permeability being closer in value to the higher permeability layer

(fault or packstone–grainstone). The initial porosity and permeability distributions are shown in Figure 5.5.With the boundary between the layers going through the cells, initial mineral amounts were assigned node-based on each side of the boundary line respectively.

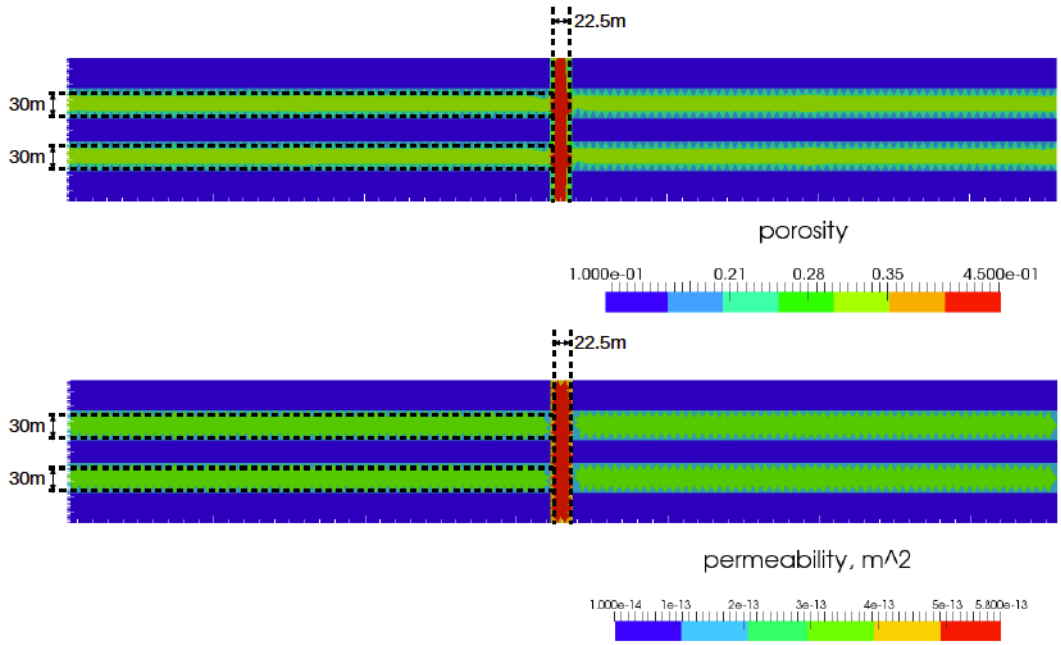

Figure 5.5: Initial porosity (top) and permeability (bottom) distribution for the Benicàssim model. The actual boundaries between the layers and the fault are marked with black lines.

Effects of layering on dolomitisation

The first fluid flow scenario is flow from left to right. A hydrostatic pressure distribution was calculated in the whole model, then the pressure on the right boundary was fixed and an additional constant pressure difference ($\Delta p_{100\,^\circ\mathrm{C}} = 2.3\,bar$) was applied across the model, by setting the higher pressure boundary condition on the left boundary. The top and bottom boundaries were assigned no-flow boundary conditions. The resulting initial average lateral velocity was $1.7 \cdot 10^{-7}\,m/s$ ($5.4\,m/yr$) in the high-permeability layers and $5.6 \cdot 10^{-}9\,m/s$ ($0.18\,m/yr$) in the low-permeability layers. This contrast is similar to the values of $6\,m/yr$ and $0.06\,m/yr$ used in Corbella et al. [43].

Dolomitising fluids entering through the fault

Based on the long-term fluid circulation model by Gomez-Rivas et al. [78], this scenario was designed to study the effects of fluid inflow at the base of the fault zone. The pressure on the left and right boundaries was hydrostatic and fixed, an additional pressure increment of $2.9\,bar$ (with respect to hydrostatic pressure) was assigned at the bottom of the fault to simulate a constant inflow. The top boundary and the rest of the bottom boundary was no-flow. This setup resulted in the initial average flow velocities of order of $10^{-7}\,m/s$ in the high-permeability layers, $10^{-8} \div 10^{-9}\,m/s$ in the low-permeability layers and of order of $10^{-6}\,m/s$ in the fault (see Fig. 5.6). High velocities in the fault result in a small CFL constraint, for this reason this simulation was running much slower than with the previous setup and only 2.5kyrs could have been simulated.

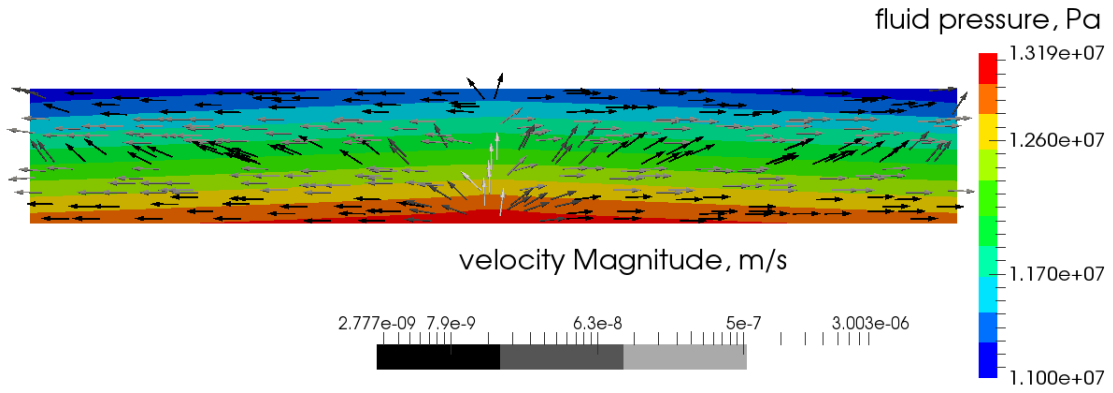

Figure 5.6: Fault injection: initial pressure distribution and velocity field

5.3 Results

5.3.1 Temperature influence on dolomitisation

Simulation results for the homogeneous permeability simulation for both temperatures after 25kyrs are presented in Figures 5.7 and 5.8. Initial total volumes of calcite (69.3%) and dolomite (0.7%) together comprise the 70% of the total volume (30% initial porosity). At 100 °C the dolomitisation process proceeds much faster than at the lower temperature, which agrees with the difference in the dolomite saturation index values for injected waters – 1.192 at 70 °C versus 1.332 at 100 °C and reflects the kinetic equation for dolomite precipitation, which is temperature-dependent. Calcite

dissolution is driven by dolomite precipitation, therefore the calcite and dolomite fronts advance with the same speed (see Fig. 5.7).

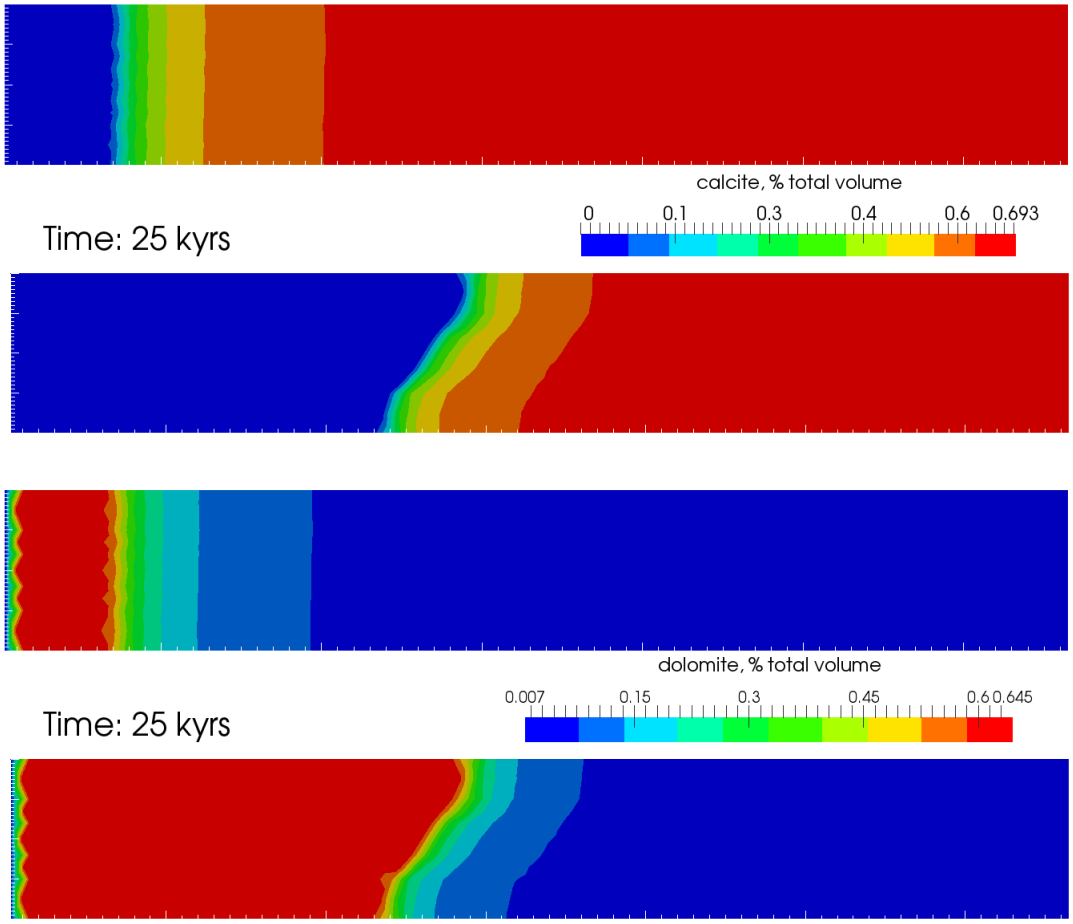

Figure 5.7: Calcite and dolomite amounts after 25kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

The porosity evolution reflects the changes in mineral amounts (see Fig. 5.8), with an increase from 30% up to 39% in the zone where all calcite gets replaced by dolomite. The permeability increases with porosity according to the Kozeny-Carman correlation (defined in the previous chapter) and leads to the minor flow acceleration as dolomite front progresses.

An interesting effect observed is that the dolomitisation front is significantly inclined at the higher temperature. This can be explained by comparing the fluid salinity distribution at different temperatures (see Fig. 5.9). At 100 °C the salinity difference between the fluids behind the front and ahead of the front is twice as much as that at 70 °C.

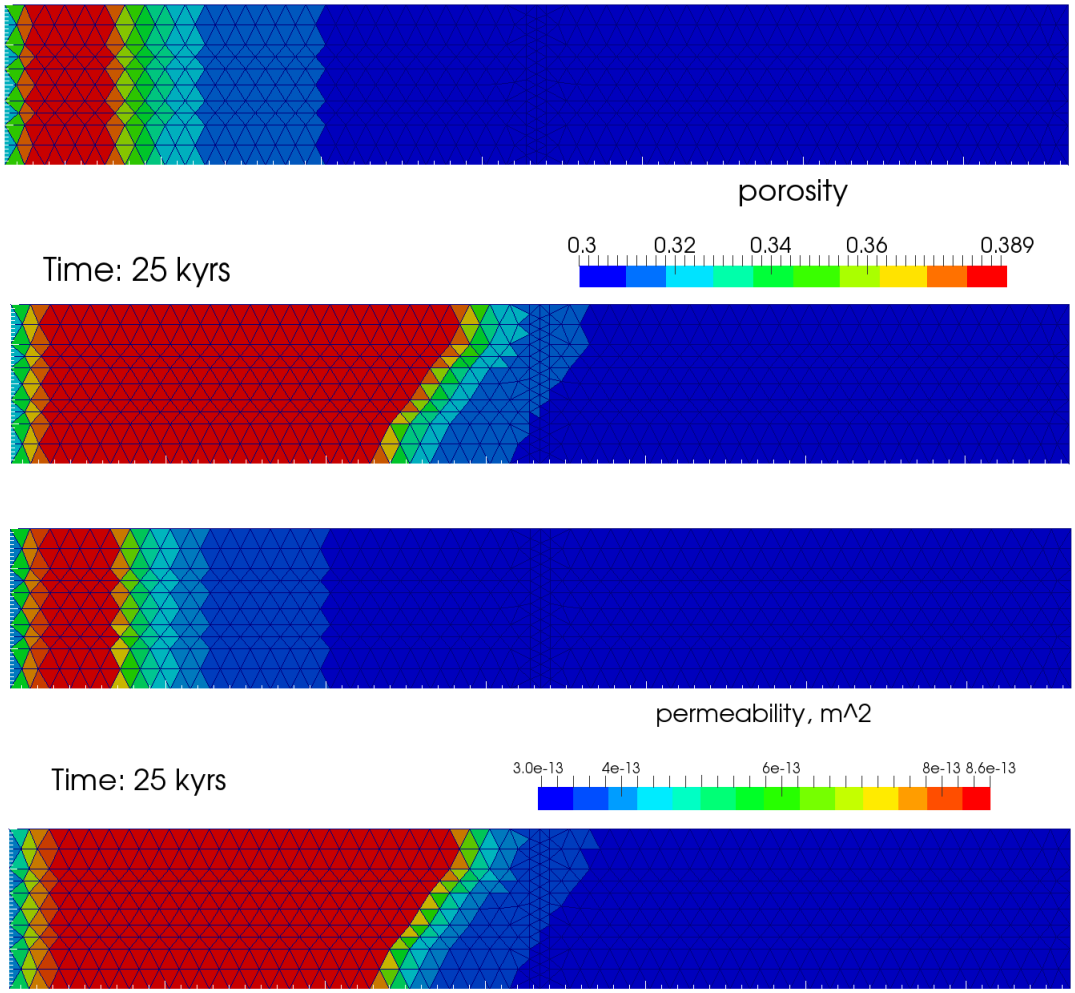

Figure 5.8: Porosity and permeability after 25kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

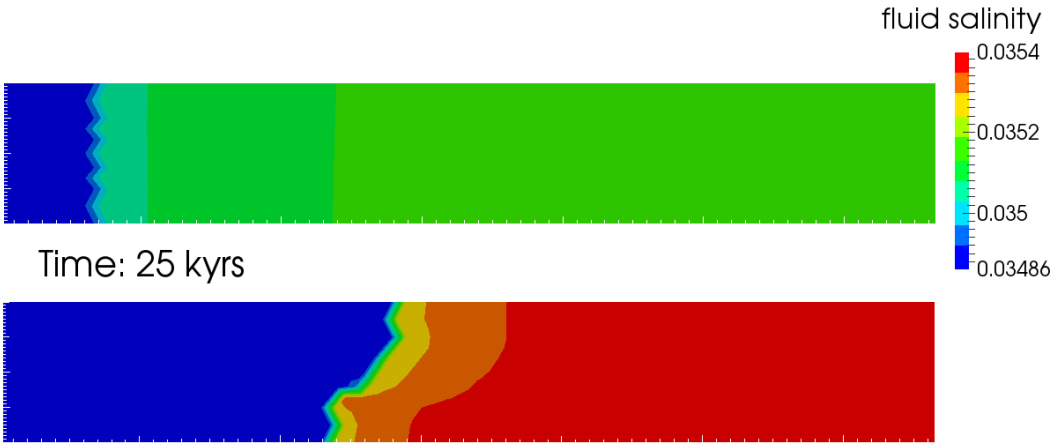

Figure 5.9: Fluid salinity after 25kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

This fluid salinity distribution affects the fluid density and viscosity, resulting in the density and velocity fields shown in Figure 5.10. At the higher temperature an additional density driven flow gradient appears, making the fluid behind the front move upwards and in general making the fluid in the upper part move faster in the horizontal direction.

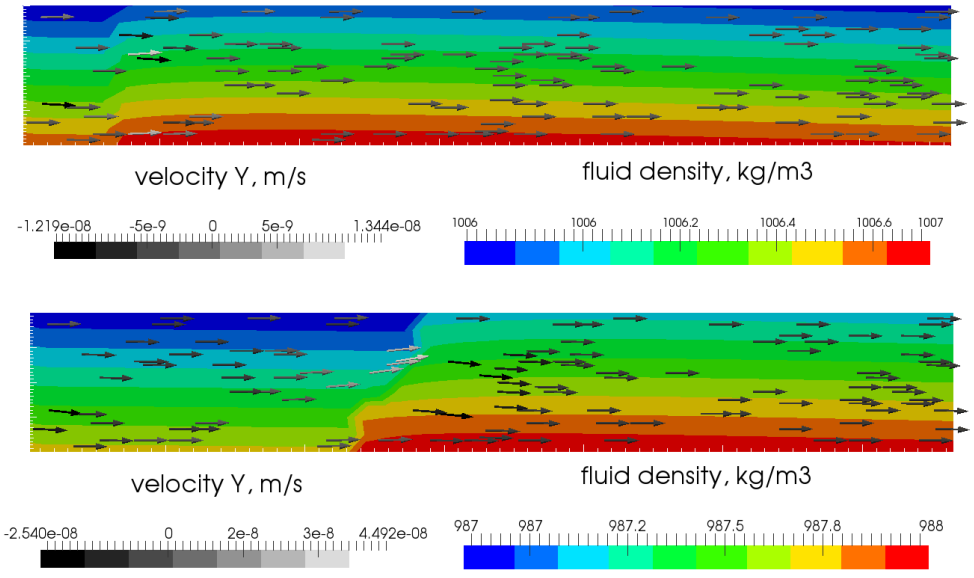

Figure 5.10: Fluid density and vertical velocity after 25kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

At 100 °C the cross-section gets completely dolomitised by about 51kyrs. By the same time only one third of the model at 70 °C is totally dolomitised (see Fig. 5.11

and 5.12). After all calcite has been replaced by dolomite, dolomite continues to precipitate as a cement from the injected water, causing the porosity to decrease near the injection boundary. As a consequence permeability also decreases.

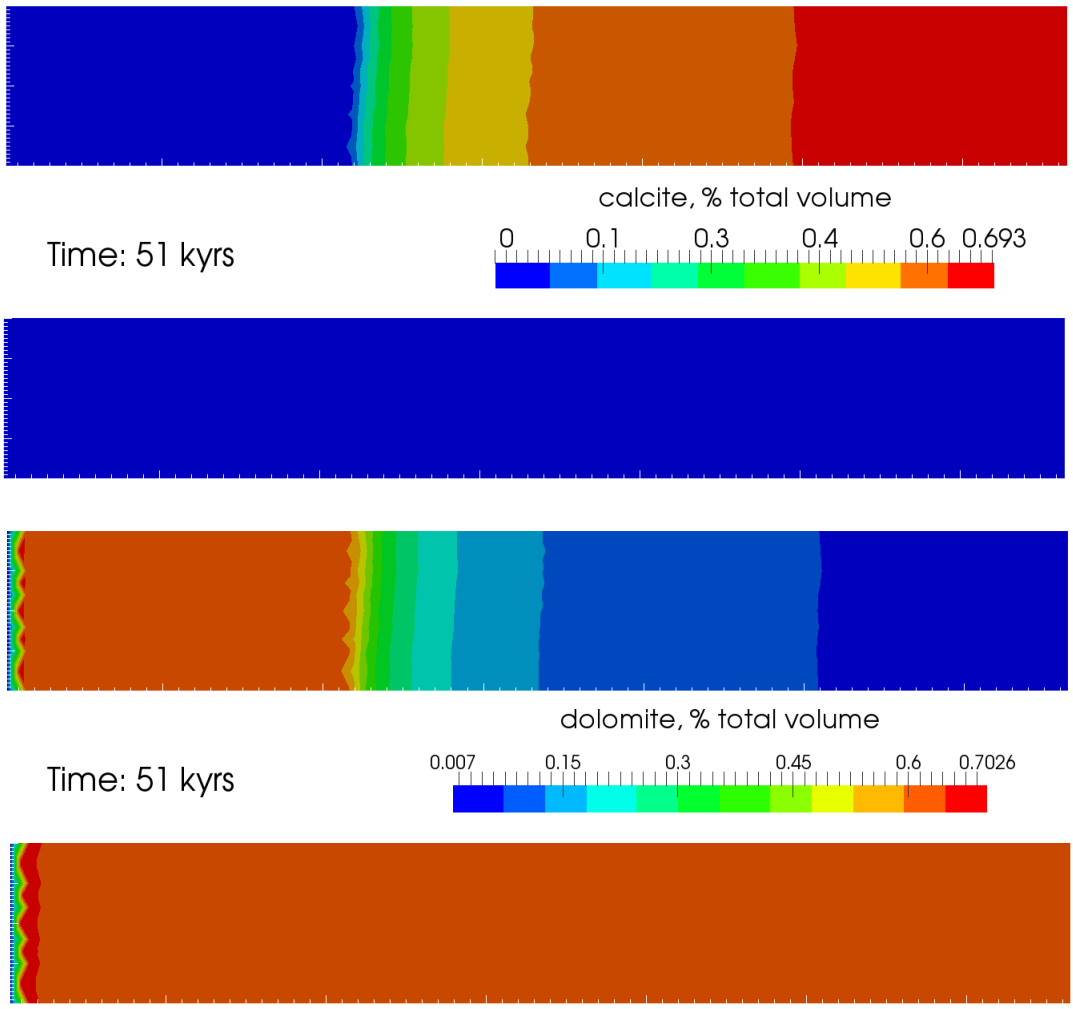

Figure 5.11: Calcite and dolomite amounts after 51kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

For the simulation at 70 °C, the overdolomitisation occurs in the first 70 m of the model; at the distance of 20 m from the left boundary porosity is reduced down to 34.6% after 51kyrs of simulation, at the distance of 40 m to 37.8% and at the distance of 60 m to 38.8%, compared to the value of 39% after the complete replacement.

For the simulation at 100 °C the overdolomitisation occurs in the first 100 m of the model; at the distance of 20 m from the left boundary porosity is reduced down to 30% after 51kyrs of simulation, at the distance of 40 m to 36.2% and at the distance of 60 m to 38.2%, compared to the value of 38.9% after the complete replacement.

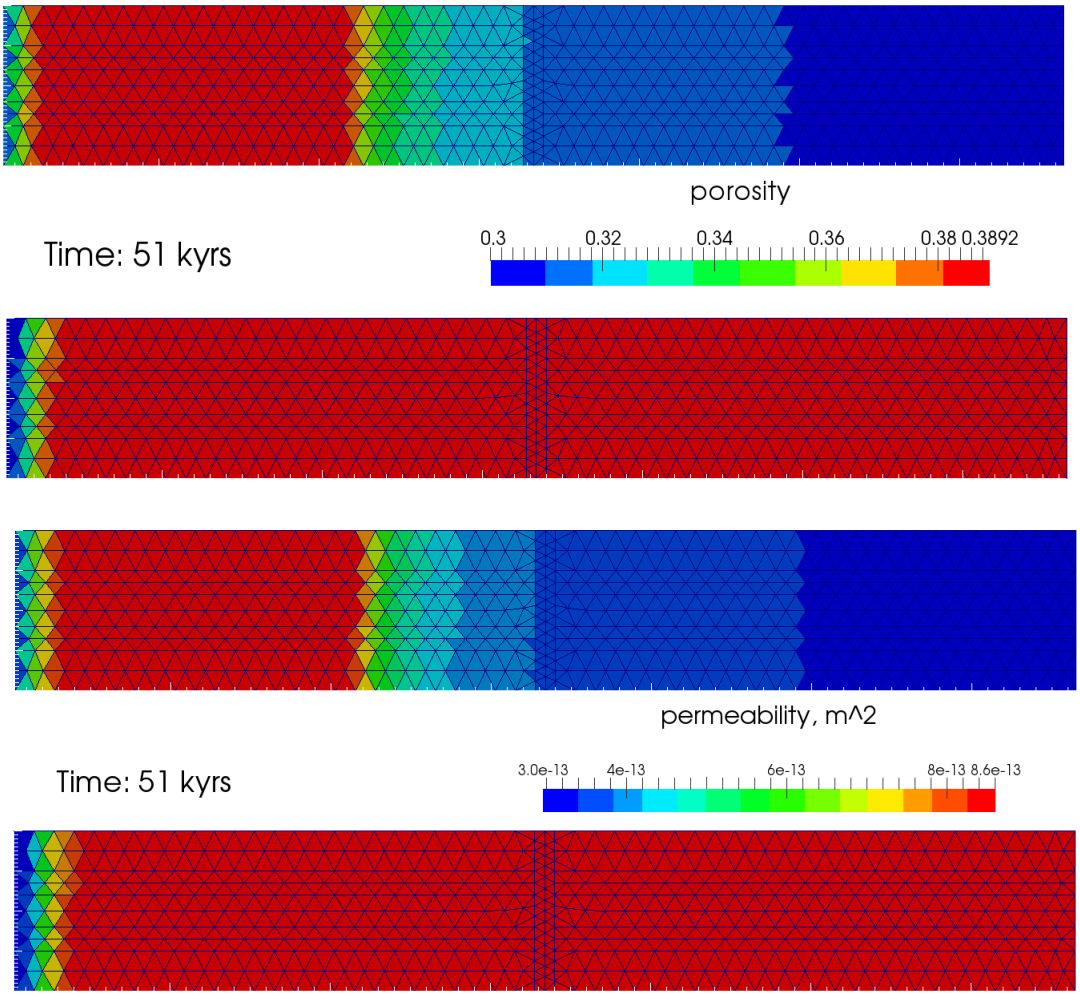

Figure 5.12: Porosity and permeability after 51kyrs of simulation time at 70 °C (top) and 100 °C (bottom).

Figure 5.13 shows calcite and dolomite amounts and the porosity evolution in a node at a distance of 192.5 m from the left boundary and 100 m from the top boundary for the simulations at 70 °C and 100 °C. The graphs highlight the differences in the kinetic rates of dolomite precipitation at different temperatures: at the higher temperature the curves are steeper than at the lower temperature. It takes 11kyrs to dolomitise the corresponding finite volume at 100 °C and 31kyrs at 70 °C. At the time of complete replacement the dolomite amount (9.48 mol/m^3) is approximately equal to the half of the initial calcite amount (18.76 mol/m^3). It corresponds to the highest porosity value – 38.9% for 100 °C and 39% for 70 °C. At this distance no overdolomitisation occurs at both temperatures and the dolomite amount stays constant after the complete calcite replacement.

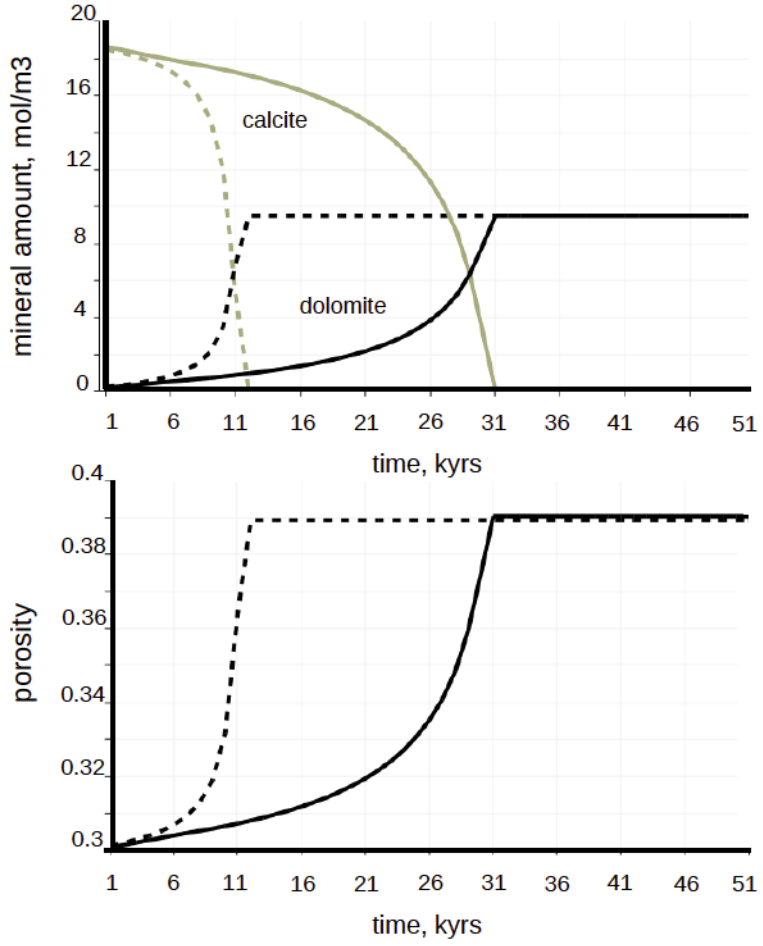

Figure 5.13: Plot over time at the distance of 192.5m from the left boundary and 100m from the top boundary: calcite and dolomite amounts (top) and porosity evolution (bottom) at 70 °C (solid lines) and 100 °C (dashed lines).

5.3.2 Effects of layering on dolomitisation

The results of the simulation with layered rock properties and flow from left to right at 100 °C after 45 and 90kyrs are presented in Figure 5.14. During the first 45kyrs the high-permeability layers are preferentially dolomitised, however after 90kyrs the low-permeability layers get dolomitised as well. Calcite is mostly dissolved in the high-permeability layers were the flow is concentrated, but also near the boundaries between the high- and low-permeability layers. Replacement dolomitisation near the boundaries increases the porosity and therefore the permeability and enhances the flow from the high-permeable layers into the low-permeable layers. This effect is more prominent in the middle layer that gets the inflow both from the upper and the lower high-permeable layers. After the dolomitising front reaches the fault and dolomitises it completely, the low-permeable layers start to dolomitise next to the fault. After complete calcite replacement, dolomite continues to precipitate and reduces the porosity near the left boundary. The high-permeable layers get completely dolomitised within the 90kyrs. At the time of the complete replacement of calcite by dolomite porosity increases to 38.9% in the high-permeability layers (initial value – 30%), to 21.5% in the low-permeability layers (10% initial porosity) and to 52% in the fault (from 45% initial).

Figure 5.15 shows the time evolution of calcite and dolomite amounts and porosity for two nodes in the middle of the upper high-permeability layer (vertical distance from the top boundary – 65 m, horizontal from the left boundary – 18.3 m and 49.1 m). Overdolomitisation occurs up to 50 m from the left boundary in the high-permeability layers. At the distance of 18.34 m porosity reduces to 15.6% (from the maximum value of 38.9% after the complete replacement) after 100kyrs, while at the distance of 49.1 m the change in porosity is less than 1%.

Figure 5.16 shows the time evolution of calcite and dolomite amounts and porosity for two nodes in the middle of the upper low-permeability layer (vertical distance from the top boundary – 25 m, horizontal from the left boundary – 18.3 m and 28.0 m). Overdolomitisation occurs in the first 30 m of the low-permeability layer. At the distance of 18.3 m porosity decreases to 15.6% from the complete replacement value of 21.5% and almost does not change at the distance of 28 m ($< 1\%$).

Figures 5.15 and 5.16 illustrate how the dolomitising front progresses in the layers with different rock properties. In the high-permeability layer the front reaches 18.3 m within the first thousands of years, whereas it takes about 20kyrs in the low-permeability layer. This can be explained by the differences in flow velocities and in

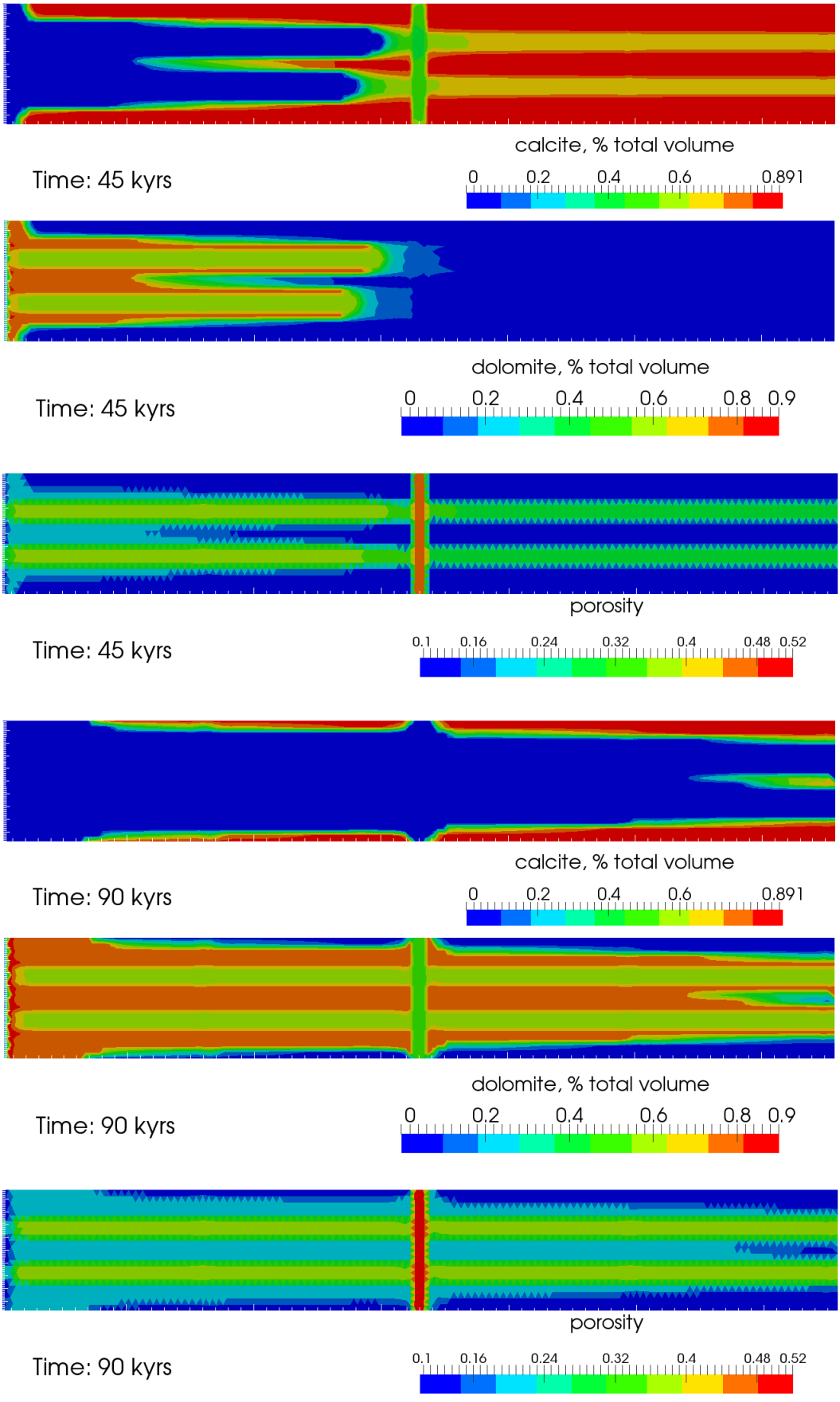

Figure 5.14: Layered rock properties: flow from left to right at 100 °C. Calcite and dolomite amounts and porosity distribution after 45kyrs and 90kyrs of simulation time

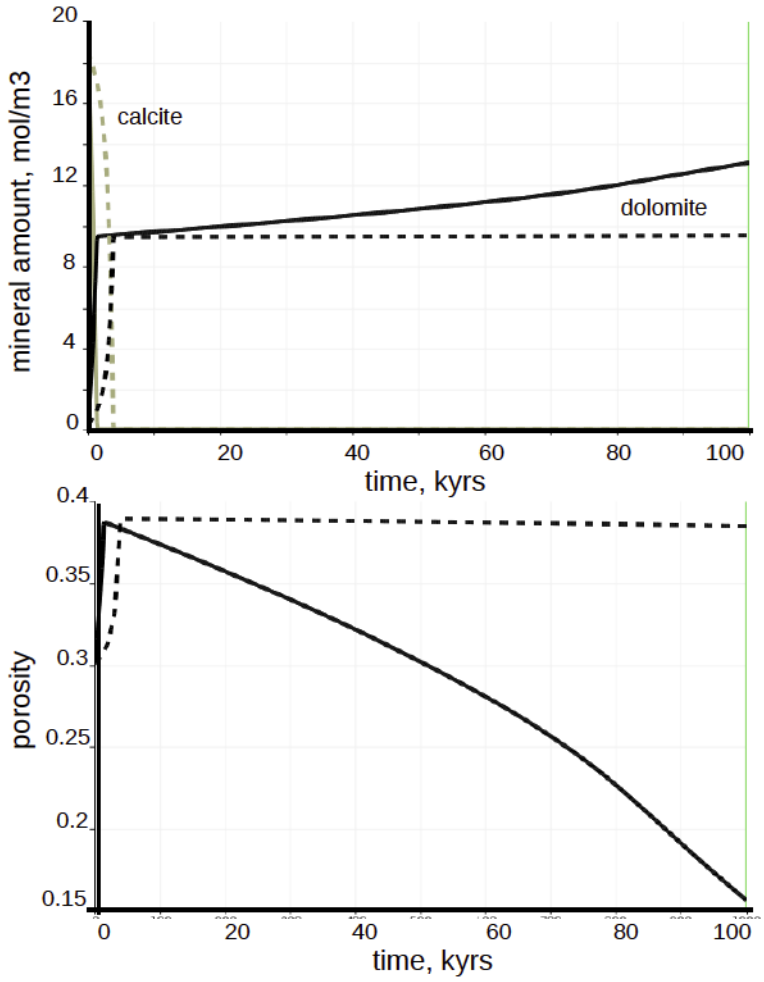

Figure 5.15: Plot over time in the high-permeability layer at the distance of $18.3\,m$ (solid lines) and $49.5\,m$ (dashed lines) from the left boundary and $65\,m$ from the top boundary: calcite and dolomite amounts (top) and porosity evolution (bottom)

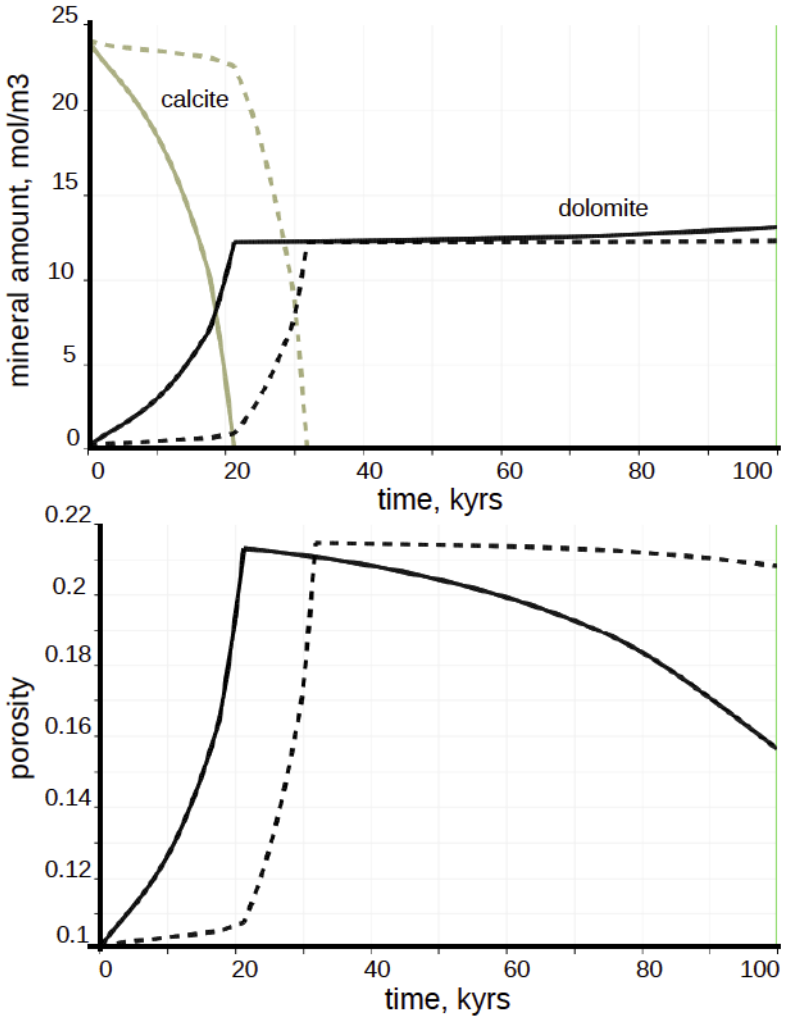

Figure 5.16: Plot over time in the low-permeability layer at the distance of $18.3\,m$ (solid lines) and $28\,m$ (dashed lines) from the left boundary and $25\,m$ from the top boundary: calcite and dolomite amounts (top) and porosity evolution (bottom)

the initial amount of calcite (69.3% of the total volume in the high-permeability layer and 89.1% in the low-permeability layer).

5.3.3 Dolomitising fluids entering through the fault

The results of the simulation with layered rock properties at 100 °C and the fault injection after 2.5kyrs are presented in Figure 5.17. First, calcite is replaced by dolomite in the fault, close to the injection region. The subsequent increase in porosity and permeability in the dolomitised region drives the flow upwards to the lower high-permeability layer where it starts to spread laterally, driving dolomitisation. Some dolomitisation also occurs in the lower low-permeability layer at the boundary with the fault, due to the porosity/permeability enhancement following dolomitisation along the fault.

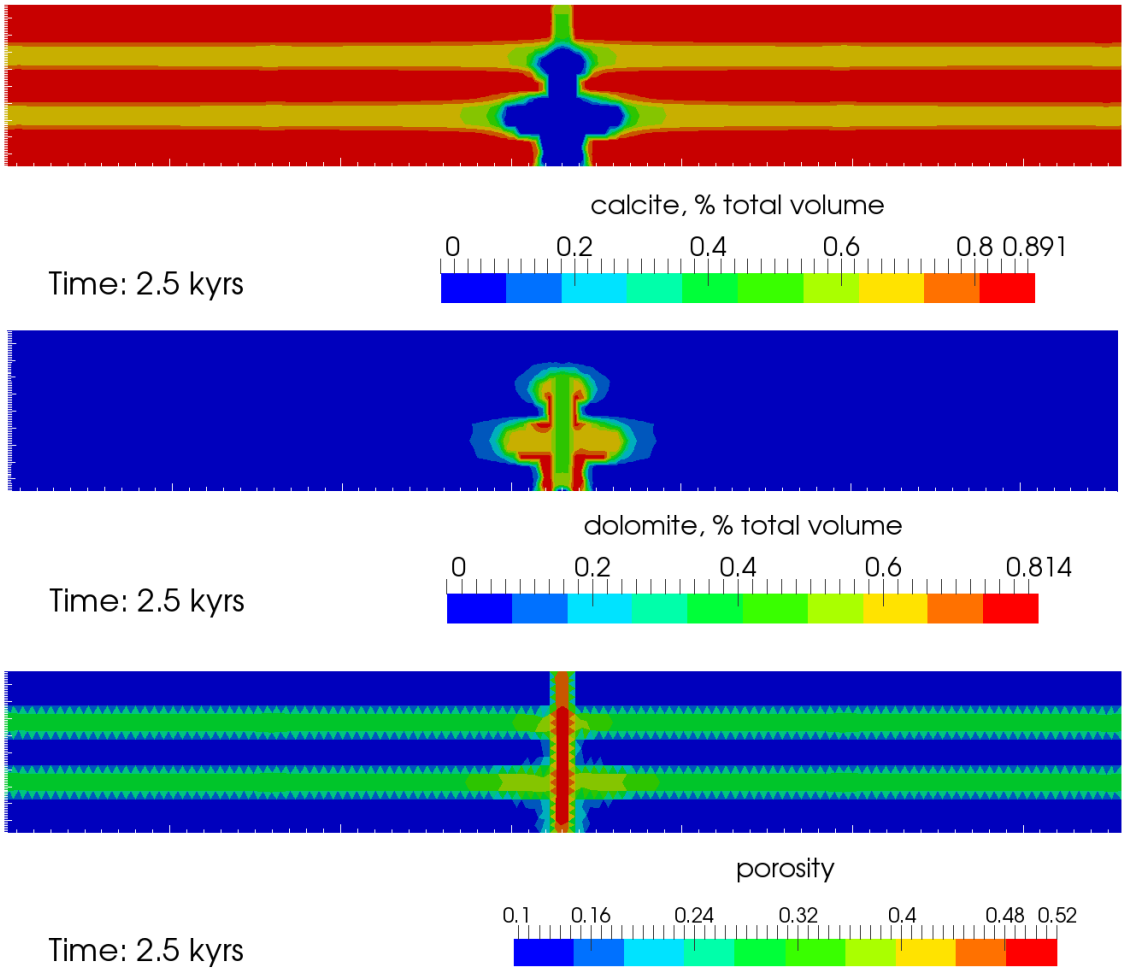

Figure 5.17: Fault injection at 100 °C: calcite and dolomite amounts and porosity distribution after 2.5kyrs of simulation time

Due to the fast calcite replacement in the fault, porosity is increased up to 52% by 2.5kyrs, however after 2kyrs, due to post-replacement dolomite cementation, porosity is reduced back to 45% close to injection region.

The obtained pattern of dolomitisation resembles true patterns observed in nature [79]. The dolomite distribution is more irregular and patchy close to the fluid-feeding fault where dolomitisation affects facies with different initial permeabilities differently. Stratabound dolomite bodies tend to develop in high permeability layers away from the fault. This pattern, known as "Christmas-tree" pattern, has been described in the Benicassim case study [43].

The "Christmas-tree" dolomitisation pattern is observed after the 2.5kyrs, however no predictions can be made whether this structure will be preserved later on, or all the layers will be eventually dolomitised like in the flow scenario above.

5.4 Conclusions

Reactive transport simulations of dolomite formation in the Benicàssim area were performed using the new CSMP++GEM coupled code. The results presented here show the capabilities of the code and give an insight into the dolomitisation process.

Unlike the software used in the previous simulations from Corbella et al. [43], our RTM code has a porosity and permeability evolution feedback. The permeability evolution during the replacement phase makes the dolomitisation a self-accelerating process. This feature could be one of the reasons why in our simulations it takes much less time to dolomitise the cross-section completely (90kyrs to dolomitise a 1300m cross-section vs. 0.5Myrs to dolomitise half of it).

Corbella et al. have assumed that both calcite and dolomite are kinetically controlled minerals, and in their simulations calcite dissolution and dolomite precipitation did not occur simultaneously: calcite was dissolved preferentially in the high-permeability beds whereas dolomite was mostly precipitated in the low-permeability beds. Our results show a behaviour that is more compatible with what has been observed in experiments: calcite dissolution is controlled by the slow rate of dolomite precipitation and the flow velocity as the dolomitising front advances faster in the high-permeability layers.

Fluid temperature is one of the main controls of the dolomitisation process. Results presented in this section agree with the results from Corbella et al. [43]. At

100 °C dolomitisation happens three times faster than at 70 °C. However, it is possible that at higher temperatures we need to use kinetics for calcite dissolution as the rates of calcite dissolution is almost as high as the rate of dolomite precipitation.

The simulation results with the fixed fluid pressure on the left and on the right model boundaries and no fault injection show stratabound dolomitisation during the first 45kyrs, with dolomite replacing preferentially the high permeability layers, later on all the layers get dolomitised. Fixed pressure on the bottom of the fault seems to be a more realistic fluid flow scenario that could predict the formation of stratabound dolomite bodies, however the corresponding simulations are computationally very costly.

We have used the same value for the dolomite specific reactive surface area for all the rock types and this value was constant during the simulation. Future work will include more realistic models of specific surface area evolution, that can address the changes in the size and shape of dolomite grains during the calcite replacement.

A more realistic boundary condition for the fluid flow would be the volumetric fluid source. Due to the node-based chemistry calculations the boundaries between different rocks were going through the middle of the cells. In the future the split boundaries could be used to represent the material boundaries more precisely. In order to achieve a better understanding of the hydrothermal dolomite formation process a complete flow system needs to be taken into account, including the hydrothermal convection, and the simulations need to be carried on in 3D.

Chapter 6

Conclusions

The work presented in this thesis contributes to the advancement of realistic simulations of non-isothermal fluid flow in porous media coupled with transport of solutes and chemical reactions. The developed codes – the Pressure-Temperature-Enthalpy scheme and the CSMP++GEM reactive transport code – share common features: sophisticated numerical methods for flow and transport, the use of accurate equations of state for pure and saline water, full account of fluid properties and their change with pressure, temperature and salinity, and the use of unstructured grids.

The Pressure-Temperature-Enthalpy scheme for the single phase non-isothermal flow in porous media implemented in CSMP++ was utilized to simulate the full working cycle of an underground hot water storage and assess its efficiency. Though neglected in many energy storage studies, groundwater flow influence on the energy losses through the storage walls was investigated. A sensitivity study for different injection temperatures showed minimal differences in the produced water temperature, that would have not been captured by a simpler solution scheme, thus highlighting the importance of precise flow and heat transport computations.

The new CSMP++GEMS coupled code is a powerful tool for reactive transport modelling. It stands out from RTM codes due to the combination of the finite element – finite volume method for the solution of flow and transport equations and the Gibbs energy minimisation method for chemical equilibrium calculations.

In codes like TOUGHREACT, the fluid properties are dependent on the brine mass fraction that changes due to the flow only and no feedback of chemical reactions is taken into account. Our simulations capture even slight changes in fluid salinity caused by mineral dissolution/precipitation that influence the fluid density and viscosity and therefore affect the flow.

The RTM simulator presented in this thesis can be used to check the existing conceptual models of dolomitisation, by trying to reproduce the patterns observed

in the case studies and outcrop analogues. Still, RTM simulations are constrained by the lack and uncertainty of the geochemical data, especially the rate constants at lower temperatures and mineral reactive surface area. The next big step in the dolomitisation simulations is to reproduce the experimental results.

The development of the CSMP++GEM code has just started, there are still many improvements to be made. These include the implementation of the Pitzer activity model and new realistic rules of porosity/permeability and grain size/shape evolution. The use of split boundaries would allow precise material boundaries representation.

The TOUGHREACT code has been widely used in reactive transport modelling applications over the last 20 years, even though it has well-known drawbacks, including results being strongly dependent on time stepping. The CSMP++GEM simulator on the other hand has demonstrated itself reliable through benchmarking and simulation results. However, it will take years of testing and application modelling to make changes in users minds and prove that there are alternative options for RTM. And this work is a small step on this way.

The computational power is growing at an exponential rate and unlike 20 years ago detailed numerical simulations of Thermal-Hydrological-Chemical processes are now possible. Nevertheless, these coupled simulations are still challenging and new. Two codes presented in this thesis allow for realistic numerical simulations of complex systems, but they are still very slow. New efficient numerical methods and parallelisation are needed to move towards the full-scale simulations.

Appendix A

Source code of the Law of Mass Action approach for reactive transport modelling implemented in CSMP++

This appendix contains the C++ code listings of the reactive transport code based on the Law of Mass Action approach (LMA-RTM), that was presented in Chapter 3. This code was used for 1D benchmarking and a 2D simulation of the geologically realistic cross-section. The LMA-RTM simulator is a verification case (CalciteDissolution_VVCase) within the CSMP++ framework and the chemical calculations part is implemented in ChemicalEquilibriumVisitor.

A.1 CalciteDissolution_VVCase.h

```
#ifndef CALCITEDISSOLUTION_VVCASE_H
#define CALCITEDISSOLUTION_VVCASE_H

#include "Test.h"

// Model construction
#include "Model1D.h"
#include "ANSYS_Model3D.h"

// File I/O and Initialization
#include "ComputationalSettings.h"
#include "InputDataManager.h"

// finite elements
#include "PDE_Integrator.h"
#include "NumIntegral_NT_op_N_dV.h"
#include "NumIntegral_dNT_op_dN_dV.h"
```

```
#include "NumIntegral_NT_lhsop_N_dV.h"
#include "NumIntegral_SetRHS_to_Zero.h"
#include "PointSource_rhsop.h"
#include "VelocityAndVolumeFlux.h"

// finite volumes
#include <ExplicitNodeCenteredFiniteVolumeTransport.h>

// visitors
#include "ConductivityVisitor.h"
#include "ChemicalEquilibriumVisitor.h"

// Solver
#include "SAMG_Solver.h"
#include "LUdcmp_Solver.h"

// Output
#include "VTU_Interface.h"

using namespace std;
using namespace csmp;


namespace csmp
  {

   class CalciteDissolution_VVCase : public Test
    {
      public:
        CalciteDissolution_VVCase(const char* prefix);
        void CalculateDispersion(Model<1U>& model);
        virtual void run();
    };

  } // csmp


#endif // CALCITEDISSOLUTION_VVCASE_H
```

A.2 CalciteDissolution_VVCase.cpp

```
#include "CalciteDissolution_VVCase.h"

using namespace std;

namespace csmp
{
  CalciteDissolution_VVCase::CalciteDissolution_VVCase
  (const char* prefix)
  {
    this->setName("CalciteDissolution_VVCase");
    prefix_=prefix;
  }
```

```
void CalciteDissolution_VVCase::CalculateDispersion(Model<1U>& model)
{
  Index velocity_key      = model.Database().StorageKey ("velocity");
  Index porosity_key      = model.Database().StorageKey ("porosity");
  Index dispersion_key    = model.Database().StorageKey ("dispersion");
  Index dispersivity_key  = model.Database().StorageKey
  ("dispersivity");

  VectorVariable<1U> velocity_vector;
  ScalarVariable porosity,dispersion, dispersivity;
  const vector<Element< 1U>*>::const_iterator modelElementsEnd
  ( model.Region("Model").ElementsEnd() );
  for( vector<Element<1U>*>::const_iterator it
  ( model.Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
    {
      (*it)->Read(velocity_key, velocity_vector);
      (*it)->Read(dispersivity_key, dispersivity);
      dispersion = dispersivity()*velocity_vector[0];
      (*it)->Store(dispersion_key, dispersion);
    }

}

void CalciteDissolution_VVCase::run()
{

  std::string regions_name (this->getName());
      std::string config_name (this->getName());
      std::string vars_name (this->getName()+".txt");

  double64 length(0.5), dx;
  size_t n_elements (50);
  dx = length/(double64)n_elements;

  Model1D model("Line", length, n_elements, vars_name.c_str());
  const PropertyDatabase<1U>& pd_ref(model.Database());
  //reference to the models property database.

  InputDataManager<1U> model_configuration;
  ComputationalSettings run_settings;
  model_configuration.ConfigureFromFile( model, config_name.c_str(),
  false, true, true, true, true, true, run_settings );

  static VTU_Interface<1U> vtu(model);
  // output properties
  list<string> output_props;
  list<string> region_names;

  output_props.push_back("fluid_pressure");
  output_props.push_back("velocity");
  output_props.push_back("dispersion");
  output_props.push_back("H_1p");
```

```
output_props.push_back("Ca_2p");
output_props.push_back("Mg_2p");
output_props.push_back("CO3_2m");
output_props.push_back("Cl_1m");
output_props.push_back("CaCO3");
output_props.push_back("CaMgCO32");

output_props.push_back("T_H_1p");
output_props.push_back("T_Ca_2p");
output_props.push_back("T_Mg_2p");
output_props.push_back("T_CO3_2m");
output_props.push_back("T_Cl_1m");

ConductivityVisitor<1U> conductivity_visitor( model, "conductivity",
"permeability", "fluid_viscosity" );
model.Accept(conductivity_visitor);

//SAMG_Solver solver;
LUdcmp_Solver solver;

//! steady state pressure
PDE_Integrator<1U, Region>  steady_state_pressure( &solver );

NumIntegral_dNT_op_dN_dV<1U>  p_conductance( pd_ref,
"conductivity", "fluid_pressure", "fluid_pressure" );
PointSource_rhsop <1U> nodal_fluid_src( pd_ref,
"nodal_fluid_source", "fluid_pressure" );

VelocityAndVolumeFlux<1U>  velocity( model,
"conductivity", "porosity", "fluid_pressure", false );

steady_state_pressure.Add( &p_conductance );
steady_state_pressure.Add ( &nodal_fluid_src );
steady_state_pressure.AddPostProcess( &velocity );

model.Apply (steady_state_pressure);

vtu.OutputDataToVTU( ( string(config_name) + "_Properties" ).c_str(),
output_props, model.Region("Model"), 0);


//! transport
ExplicitNodeCenteredFiniteVolumeTransport<1U,
ExplicitStencilProcessor>
transport1 ("Model", model, "porosity", "T_Ca_2p", "velocity",
"nodal_fluid_source", false);

ExplicitNodeCenteredFiniteVolumeTransport<1U,
ExplicitStencilProcessor>
transport2 ("Model", model, "porosity", "T_Mg_2p", "velocity",
"nodal_fluid_source", false);

ExplicitNodeCenteredFiniteVolumeTransport<1U,
ExplicitStencilProcessor>
```

```
transport3 ("Model", model, "porosity", "T_H_1p", "velocity",
"nodal_fluid_source", false);

ExplicitNodeCenteredFiniteVolumeTransport<1U,
ExplicitStencilProcessor>
transport4 ("Model", model, "porosity", "T_CO3_2m", "velocity",
"nodal_fluid_source", false);

ExplicitNodeCenteredFiniteVolumeTransport<1U,
ExplicitStencilProcessor>
transport5 ("Model", model, "porosity", "T_Cl_1m", "velocity",
"nodal_fluid_source", false);

//! diffusive transport
LUdcmp_Solver solver1;
//!Ca
PDE_Integrator<1U, Region> diff_transport1( &solver1 );
NumIntegral_dNT_op_dN_dV<1U> dispersion1( pd_ref, "dispersion",
"T_Ca_2p", "T_Ca_2p" );

NumIntegral_NT_lhsop_N_dV<1U> capacitance_lhs1( pd_ref, "unity",
"T_Ca_2p", "T_Ca_2p" );
capacitance_lhs1.LumpedFormulation(true);
capacitance_lhs1.MultiplyWithTimeIncrement (true);

NumIntegral_NT_op_N_dV<1U>    capacitance_rhs1( pd_ref, "unity",
"T_Ca_2p" );
capacitance_rhs1.MultiplyWithTimeIncrement (true);

diff_transport1.Add( &dispersion1 );
diff_transport1.Add( &capacitance_lhs1 );
diff_transport1.Add( &capacitance_rhs1 );

//!Mg
PDE_Integrator<1U, Region> diff_transport2( &solver1 );
NumIntegral_dNT_op_dN_dV<1U> dispersion2( pd_ref, "dispersion",
"T_Mg_2p", "T_Mg_2p" );

NumIntegral_NT_lhsop_N_dV<1U> capacitance_lhs2( pd_ref, "unity",
"T_Mg_2p", "T_Mg_2p" );
capacitance_lhs2.LumpedFormulation(true);
capacitance_lhs2.MultiplyWithTimeIncrement (true);

NumIntegral_NT_op_N_dV<1U>    capacitance_rhs2( pd_ref, "unity",
"T_Mg_2p" );
capacitance_rhs2.MultiplyWithTimeIncrement (true);

diff_transport2.Add( &dispersion2 );
diff_transport2.Add( &capacitance_lhs2 );
diff_transport2.Add( &capacitance_rhs2 );

//!H
PDE_Integrator<1U, Region> diff_transport3( &solver1 );
NumIntegral_dNT_op_dN_dV<1U> dispersion3( pd_ref, "dispersion",
```

```
"T_H_1p", "T_H_1p" );

NumIntegral_NT_lhsop_N_dV<1U> capacitance_lhs3( pd_ref, "unity",
"T_H_1p", "T_H_1p" );
capacitance_lhs3.LumpedFormulation(true);
capacitance_lhs3.MultiplyWithTimeIncrement (true);

NumIntegral_NT_op_N_dV<1U>    capacitance_rhs3( pd_ref, "unity",
"T_H_1p" );
capacitance_rhs3.MultiplyWithTimeIncrement (true);

diff_transport3.Add( &dispersion3 );
diff_transport3.Add( &capacitance_lhs3 );
diff_transport3.Add( &capacitance_rhs3 );

//!CO3
PDE_Integrator<1U, Region>  diff_transport4( &solver1 );
NumIntegral_dNT_op_dN_dV<1U>  dispersion4( pd_ref, "dispersion",
"T_CO3_2m", "T_CO3_2m" );

NumIntegral_NT_lhsop_N_dV<1U> capacitance_lhs4( pd_ref, "unity",
"T_CO3_2m", "T_CO3_2m" );
capacitance_lhs4.LumpedFormulation(true);
capacitance_lhs4.MultiplyWithTimeIncrement (true);

NumIntegral_NT_op_N_dV<1U>    capacitance_rhs4( pd_ref, "unity",
"T_CO3_2m" );
capacitance_rhs4.MultiplyWithTimeIncrement (true);

diff_transport4.Add( &dispersion4 );
diff_transport4.Add( &capacitance_lhs4 );
diff_transport4.Add( &capacitance_rhs4 );

//!Cl
PDE_Integrator<1U, Region>  diff_transport5( &solver1 );
NumIntegral_dNT_op_dN_dV<1U>  dispersion5( pd_ref, "dispersion",
"T_Cl_1m", "T_Cl_1m" );

NumIntegral_NT_lhsop_N_dV<1U> capacitance_lhs5( pd_ref, "unity",
"T_Cl_1m", "T_Cl_1m" );
capacitance_lhs5.LumpedFormulation(true);
capacitance_lhs5.MultiplyWithTimeIncrement (true);

NumIntegral_NT_op_N_dV<1U>    capacitance_rhs5( pd_ref, "unity",
"T_Cl_1m" );
capacitance_rhs5.MultiplyWithTimeIncrement (true);

diff_transport5.Add( &dispersion5 );
diff_transport5.Add( &capacitance_lhs5 );
diff_transport5.Add( &capacitance_rhs5 );

//! get nodal rock properties
model.ExtrapolateElementToNodeProperty("porosity", "nodal_porosity",
false); //false -- by volume, true -- by distance
```

```
    //! initial equilibration
    ChemicalEquilibriumVisitor<1U>  reactor( model );
    reactor.SetInitialProperties ();

    vtu.OutputDataToVTU( ( string(config_name) + "_Properties_equil" )
    .c_str(), output_props, model.Region("Model"), 0);
    cin.get();

    double64        time_increment;
    double64        total_time;
    double64        global_time = 0.;
    size_t          saved = 1;

  time_increment = 200.;
  total_time = 50000;
  size_t timestep = 0;
  while ( global_time <= total_time )
    {
      CalculateDispersion(model);

      diff_transport1.TimeIncrement (1./time_increment);
      diff_transport2.TimeIncrement (1./time_increment);
      diff_transport3.TimeIncrement (1./time_increment);
      diff_transport4.TimeIncrement (1./time_increment);
      diff_transport5.TimeIncrement (1./time_increment);
      model.Apply (diff_transport1);
      model.Apply (diff_transport2);
      model.Apply (diff_transport3);
      model.Apply (diff_transport4);
      model.Apply (diff_transport5);

      // advecting basis species
      // ----------------------------
      transport1.AdvectVariable(time_increment, 0.5, false, false);
      transport2.AdvectVariable(time_increment, 0.5, false, false);
      transport3.AdvectVariable(time_increment, 0.5, false, false);
      transport4.AdvectVariable(time_increment, 0.5, false, false);
      transport5.AdvectVariable(time_increment, 0.5, false, false);

      vtu.OutputDataToVTU( ( string(config_name) + "_Properties_before
_______react" ).c_str(), output_props, model.Region("Model"), timestep);

      reactor.Equilibrate();

      vtu.OutputDataToVTU( ( string(config_name) + "_Properties_after
_______react" ).c_str(), output_props, model.Region("Model"), timestep);
      timestep++;

       global_time += time_increment;
    }
  }

}//end csmp
```

A.3 ChemicalEquilibriumVisitor.h

```
#ifndef CHEMICAL_EQUILIBRIUM_VISITOR_H
#define CHEMICAL_EQUILIBRIUM_VISITOR_H

/** @file ChemicalEquilibriumVisitor.h
 *  @author Alina Yapparova
 *  @brief  Water-Calcite at equilibrium
 *  @details computes water-calcite equilibrium,
 all reactions are modeled using equilibrium constants,
 *  the resulting system of non-linear equations is solved
 using KINSOL from SUNDIALS software library
 *  @date 03.07.2013
 */

#include "Model.h"
#include "Visitor.h"
#include "ErrorHandler.h"

#include <kinsol/kinsol.h>
#include <kinsol/kinsol_dense.h>
#include <nvector/nvector_serial.h>
#include <sundials/sundials_types.h>
#include <sundials/sundials_math.h>

#define NEQ 6    // number of equations (NPRIM+NMIN)
#define NMIN 2   // number of minerals at equilibrium
#define NPRIM 4 // number of primary species
#define NSEC 2   // number of secondary species

#define ONE  RCONST(1.0)
#define ZERO RCONST(0.0)
#define TOL  1.e-20

namespace csmp {

typedef struct
{
    double H_1p;
    double n_cc, n_d;
    double T_H_1p, T_CO3_2m, T_Ca_2p, T_Mg_2p, T_Cl_1m;
    double a_Ca_2p, a_Mg_2p, a_H_1p, a_CO3_2m, a_OH_1m, a_HCO3_1m;
    bool flag_cc, flag_d;
} *UserData;

template<size_t dim>
class ChemicalEquilibriumVisitor: public Visitor<dim> {
  public:
    ChemicalEquilibriumVisitor( Model<dim>& sg );
    ~ChemicalEquilibriumVisitor();

    void Equilibrate ();
    void SetInitialProperties();
```

```
    virtual void Visit( Node<dim>* );

  private:
    ChemicalEquilibriumVisitor();
    Model<dim>& model_;
    bool equil_flag;

    Index nphi_key_, FV_key_, H_1p_key_, Ca_2p_key_, Mg_2p_key_,
    HCO3_1m_key_, Cl_1m_key_, T_H_1p_key_, T_CO3_2m_key_, T_Ca_2p_key_,
    T_Mg_2p_key_, T_HCO3_1m_key_, T_Cl_1m_key_, CaCO3_key_,
    CaMgCO32_key_, CO2_key_, CO3_2m_key_;

    ScalarVariable phi, FV, H_1p, Ca_2p, Mg_2p, HCO3_1m, OH_1m, Cl_1m,
    T_H_1p, T_Ca_2p, T_Mg_2p, T_HCO3_1m, T_CO3_2m, T_Cl_1m, CaCO3,
    CaMgCO32, CO2, CO3_2m;

    const double pure_water_density;
    const double calcite_density;
    const double calcite_molar_mass;

    //! KINSOL related variables
    N_Vector C, C_init; // concentrations of primary species
    N_Vector scale, scale_init; // scaling vector (=1 in our case)
    int mset, flag;
    void *kmem, *kmem_init;
    UserData data;

    //! KINSOL related functions
    int check_flag(void *flagvalue, char *funcname, int opt);
    void PrintOutput(N_Vector y);
    void PrintFinalStats(void *kmem);
};
} // csmp
#endif //CHEMICAL_EQUILIBRIUM_VISITOR_H
```

A.4 ChemicalEquilibriumVisitor.cpp

```
#include "ChemicalEquilibriumVisitor.h"

namespace csmp{

static int func(N_Vector y, N_Vector f, void *user_data);
static int func_init(N_Vector y, N_Vector f, void *user_data);
static int jac(long int N, N_Vector y, N_Vector f, DlsMat J,
void *user_data, N_Vector tmp1, N_Vector tmp2);
void TotalConcentrations(N_Vector C, std::vector<double>& result);
void TotalActivities(N_Vector C, std::vector<double>& a,
std::vector<double>& result);
void ActivityCoefficients(N_Vector C, std::vector<double>& a,
void *user_data);


template<size_t dim>
```

```
ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor
( Model<dim>& model ):
  Visitor<dim>( MODEL, NODE ),
  pure_water_density (1.e3),          // kg/m3
  calcite_density    (2710.),         // kg/m3
  calcite_molar_mass (100.09e-3),     // kg/mol
  model_ (model),

  nphi_key_      (model.Database().StorageKey    ( "nodal_porosity" ) ),
  FV_key_        (model.Database().StorageKey    ( "finite_volume") ),
  H_1p_key_      (model.Database().StorageKey    ( "H_1p"           ) ),
  CO3_2m_key_    (model.Database().StorageKey    ( "CO3_2m"         ) ),
  Ca_2p_key_     (model.Database().StorageKey    ( "Ca_2p"          ) ),
  Mg_2p_key_     (model.Database().StorageKey    ( "Mg_2p"          ) ),
  HCO3_1m_key_   (model.Database().StorageKey    ( "HCO3_1m"        ) ),
  Cl_1m_key_     (model.Database().StorageKey    ( "Cl_1m"          ) ),
  T_H_1p_key_      (model.Database().StorageKey    ( "T_H_1p"       ) ),
  T_CO3_2m_key_    (model.Database().StorageKey    ( "T_CO3_2m"     ) ),
  T_Ca_2p_key_     (model.Database().StorageKey    ( "T_Ca_2p"      ) ),
  T_Mg_2p_key_     (model.Database().StorageKey    ( "T_Mg_2p"      ) ),
  T_HCO3_1m_key_   (model.Database().StorageKey    ( "T_HCO3_1m"    ) ),
  T_Cl_1m_key_   (model.Database().StorageKey    ( "T_Cl_1m"        ) ),
  CaCO3_key_     (model.Database().StorageKey    ( "CaCO3"          ) ),
  CaMgCO32_key_     (model.Database().StorageKey     ( "CaMgCO32"    ) )
  {
    ErrorHandler &err(ErrorHandler::Instance());

    //! initialize KINSOL related variables
    // Create vector for a solution
    C = N_VNew_Serial (NEQ);
    if (check_flag((void *)C, "N_VNew_Serial", 0)) err.notice (ERROR,
       "ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
                            "solution vector bad memory allocation");
    scale = N_VNew_Serial(NEQ);
    if (check_flag((void *)scale, "N_VNew_Serial", 0)) err.notice (ERROR,
          "ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
                                    "scale vector bad memory allocation");
    // set scaling to one
    N_VConst_Serial(ONE, scale);

    // Initialize and allocate memory for KINSOL
    kmem = KINCreate();
    if (check_flag((void *)kmem, "KINCreate", 0)) err.notice (ERROR,
      "ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
                                    "solver bad memory allocation");

    flag = KINInit(kmem, func, C); /* C passed as a template */
    if (check_flag(&flag, "KINInit", 1)) err.notice (ERROR,
       "ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
                                  "error in solver initialization");

    // Attach dense linear solver
    flag = KINDense(kmem, NEQ);
    if (check_flag(&flag, "KINDense", 1)) err.notice (ERROR,
```

```
        "ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
                                "error in linear solver initialization");

    // Indicate exact Newton
    mset = 1;
    flag = KINSetMaxSetupCalls(kmem, mset);
    if (check_flag(&flag, "KINSetMaxSetupCalls", 1)) err.notice (ERROR,
"ChemicalEquilibriumVisitor<dim>::ChemicalEquilibriumVisitor",
"error in Newton solver initialization");

    // Allocate memory for UserData
    data = (UserData) malloc (sizeof *data);
}
template<size_t dim>
ChemicalEquilibriumVisitor<dim>::~ChemicalEquilibriumVisitor()
{
   N_VDestroy_Serial(C);
   N_VDestroy_Serial(scale);
   KINFree(&kmem);
   free(data);
}

template<size_t dim>
void ChemicalEquilibriumVisitor<dim>::SetInitialProperties ()
{
  const typename vector<Node<dim>*>::const_iterator
  modelNodesEnd( model_.Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator n(
  model_.Region("Model").NodesBegin() );
  n != modelNodesEnd; ++n )
  {
    (*n)->Read (T_H_1p_key_, T_H_1p);

    if ((*n)->Status(T_H_1p_key_) != DIRICH)
    {
      (*n)->Read (H_1p_key_, H_1p);
      (*n)->Read (T_Ca_2p_key_, T_Ca_2p);
      (*n)->Read (T_Mg_2p_key_, T_Mg_2p);
      (*n)->Read (T_CO3_2m_key_, T_CO3_2m);
      (*n)->Read (CaCO3_key_, CaCO3);

      double64 coeff = 1.8/0.32; // converting mol/kg of soil
      to mol/kg of water
      CaCO3()     *= coeff;


      C_init = N_VNew_Serial (NPRIM-1);
      scale_init = N_VNew_Serial(NPRIM-1);
      N_VConst_Serial(ONE, scale_init);
      kmem_init = KINCreate();
      KINInit(kmem_init, func_init, C_init); /* C passed as a template */
      KINDense(kmem_init, NPRIM-1);

      data->T_Ca_2p    = T_Ca_2p()   ;
```

```
data->H_1p       = log10(H_1p());
data->T_CO3_2m   = T_CO3_2m();
data->n_cc = CaCO3();

KINSetUserData(kmem_init, data);

NV_Ith_S(C_init, 0) = log10(T_Ca_2p());
NV_Ith_S(C_init, 1) = log10(T_CO3_2m());
NV_Ith_S(C_init, 2) = CaCO3();

flag = KINSol(kmem_init,           /* KINSol memory block */
              C_init,              /* initial guess on input;
              solution vector */
              KIN_LINESEARCH, /* global stragegy choice */
              scale_init,          /* scaling vector, for the
              variable cc */
              scale_init);         /* scaling vector for function
              values fval */

Ca_2p()      = NV_Ith_S(C_init, 0);
CO3_2m()     = NV_Ith_S(C_init, 1);
CaCO3()      = NV_Ith_S(C_init, 2);


realtype K1 = -8.47, K2 = -17.17, K3 = 14.01, K4 = -10.31,
K5 = -16.71;
H_1p() = log10(H_1p());
realtype I, A, a_Ca_2p, a_CO3_2m, a_HCO3_1m, a_OH_1m, a_H_1p;
OH_1m() =  log10(pow(10., - H_1p() - K3));
HCO3_1m() = log10(pow(10.,   H_1p() + CO3_2m() - K4));

I = 0.5*  ( pow(10., Ca_2p())*4 + pow(10.,H_1p()) +
pow(10.,CO3_2m())*4 +  pow(10., OH_1m()) + pow(10., HCO3_1m()) );
A = -0.5*(sqrt(I)/(1+sqrt(I)) - 0.3*I);
a_Ca_2p    = 4*A;
a_CO3_2m   = 4*A;
a_HCO3_1m  = A;
a_OH_1m    = A;
a_H_1p     = A;
                // 'cause H_1p is actually pH here
T_H_1p()       = pow(10., H_1p() - a_H_1p) - pow(10., - H_1p()
- K3 - a_OH_1m) + pow(10.,   H_1p() + CO3_2m() + a_CO3_2m - K4 -
a_HCO3_1m)+ 2*pow(10., 2*H_1p() + CO3_2m() + a_CO3_2m - K5);

Ca_2p()      = pow( 10., NV_Ith_S(C_init, 0)  );
CO3_2m()     = pow( 10., NV_Ith_S(C_init, 1 ) );

(*n)->Store (Ca_2p_key_,  Ca_2p);
(*n)->Store (CO3_2m_key_, CO3_2m);

// and back to mol/kg of soil
CaCO3()    /= coeff;
(*n)->Store (CaCO3_key_, CaCO3);
```

```
      (*n)->Store (T_H_1p_key_,  T_H_1p);

      N_VDestroy_Serial(C_init);
      N_VDestroy_Serial(scale_init);
      KINFree(&kmem_init);
    }
    else
      {
      (*n)->Read (H_1p_key_, H_1p);
      (*n)->Read (T_Cl_1m_key_, T_Cl_1m);
      (*n)->Read (T_Mg_2p_key_, T_Mg_2p);

      realtype K3 = 14.01;
      H_1p() = log10(H_1p());
      realtype I, A, a_OH_1m, a_H_1p;
      OH_1m() =  log10(pow(10., - H_1p() - K3));

      I = 0.5*  ( pow(10., T_Mg_2p())*4 + pow(10.,H_1p())
      + pow(10., OH_1m()) + pow(10., T_Cl_1m()) );
      A = -0.5*(sqrt(I)/(1+sqrt(I)) - 0.3*I);
      a_OH_1m    = A;
      a_H_1p     = A;

      T_H_1p()       = pow(10., H_1p() - a_H_1p) - pow(10., - H_1p()
      - K3 - a_OH_1m);
      (*n)->Store (T_H_1p_key_,  T_H_1p);

      }
  }
}
template<size_t dim>
void ChemicalEquilibriumVisitor<dim>::Equilibrate ()
 {
     printf("\n Equilibrating water with calcite\n");
     model_.Accept( *this );
 }
```

```
template<size_t dim>
void ChemicalEquilibriumVisitor<dim>::Visit( Node<dim>* n )
{
    ErrorHandler &err(ErrorHandler::Instance());

    n->Read (nphi_key_, phi);
    n->Read (H_1p_key_, H_1p);
    n->Read (Ca_2p_key_, Ca_2p);
    n->Read (Mg_2p_key_, Mg_2p);
    n->Read (CO3_2m_key_, CO3_2m);
    n->Read (CaCO3_key_, CaCO3);
    n->Read (CaMgCO32_key_, CaMgCO32);

    n->Read (T_H_1p_key_, T_H_1p);
    n->Read (T_Ca_2p_key_, T_Ca_2p);
    n->Read (T_Mg_2p_key_, T_Mg_2p);
    n->Read (T_CO3_2m_key_, T_CO3_2m);
```

```
n->Read (T_Cl_1m_key_, T_Cl_1m);
n->Store (Cl_1m_key_, T_Cl_1m);

equil_flag = false;
data->flag_cc = true;
data->flag_d  = true;

double64 coeff = 1.8/0.32;
// converting mol/kg of soil to mol/kg of water
CaCO3()     *= coeff;
CaMgCO32() *= coeff;

int iter = 0;
std::vector<double> a (NPRIM+NSEC, 0.);

if (n->Status(H_1p_key_) != DIRICH)
{
  while (!equil_flag && iter<3)
  {
    //! initial concentrations
    NV_Ith_S(C, 0) = log10(0.95*T_Ca_2p()); //! also possible Ca_2p
    NV_Ith_S(C, 1) = log10(0.99*T_Mg_2p()); //! also possible Mg_2p
    NV_Ith_S(C, 2) = log10(H_1p());     //! dont use T_H_1p as
    an initial guess, because it can be negative!
    NV_Ith_S(C, 3) = log10(CO3_2m());  //! T_CO3
    NV_Ith_S(C, 4) = CaCO3();
    NV_Ith_S(C, 5) = CaMgCO32();

    //printf("\nInitial guess:\n");
    //PrintOutput(C);

    //! UserData data, total concentrations of basic species
    and molar number of Calcite
    data->n_cc = CaCO3();
    data->n_d = CaMgCO32();

    data->T_Ca_2p   = T_Ca_2p()  ;
    data->T_Mg_2p   = T_Mg_2p()  ;
    data->T_H_1p    = T_H_1p()   ;
    data->T_CO3_2m  = T_CO3_2m();
    data->T_Cl_1m   = T_Cl_1m();

    KINSetUserData(kmem, data);

    // Call KINSol to solve problem
    flag = KINSol(kmem,           /* KINSol memory block */
                  C,              /* initial guess on input;
                  solution vector */
                  //KIN_NONE,
                  KIN_LINESEARCH, /* global stragegy choice */
                  scale,          /* scaling vector,
                  for the variable cc */
                  scale);         /* scaling vector
                  for function values fval */
```

```
    if (flag != KIN_SUCCESS && flag != KIN_INITIAL_GUESS_OK)
        err.notice (ERROR, "ChemicalEquilibriumVisitor<dim>::Visit",
                           "KINSOL_failed");

    //printf("\nComputed solution:\n");
    //PrintOutput(C);

    if ( NV_Ith_S(C, 4) < TOL && NV_Ith_S(C, 5) < TOL)
      {
        if ( NV_Ith_S(C, 4) <  NV_Ith_S(C, 5))
          data->flag_cc = false;
        else
          data->flag_d = false;
      }
    else
      {
        if ( NV_Ith_S(C, 4) < TOL )
          data->flag_cc = false;
        if ( NV_Ith_S(C, 5) < TOL)
          data->flag_d = false;
      }

    //Print final statistics and free memory
    //PrintFinalStats(kmem);

    if (NV_Ith_S(C, 4) > -TOL && NV_Ith_S(C, 5) > -TOL)
      equil_flag = true;

    iter++;
  }

// calculate activity coefficients
ActivityCoefficients(C, a, data);

std::vector<double> result (NPRIM, 0.);
//TotalConcentrations(C, result);
TotalActivities(C, a, result);

T_Ca_2p()   = result[0];
T_Mg_2p()   = result[1];
T_H_1p()    = result[2];
T_CO3_2m() = result[3];

n->Store (T_H_1p_key_,    T_H_1p);
n->Store (T_Ca_2p_key_,   T_Ca_2p);
n->Store (T_Mg_2p_key_,   T_Mg_2p);
n->Store (T_CO3_2m_key_, T_CO3_2m);

Ca_2p()     = pow( 10., NV_Ith_S(C, 0)  );
Mg_2p()     = pow( 10., NV_Ith_S(C, 1)  );
H_1p()      = pow( 10., NV_Ith_S(C, 2)  );
CO3_2m()    = pow( 10., NV_Ith_S(C, 3)  );
CaCO3()     = NV_Ith_S(C, 4);
```

```
    CaMgCO32()  = NV_Ith_S(C, 5);

    n->Store (H_1p_key_, H_1p);
    n->Store (Ca_2p_key_, Ca_2p);
    n->Store (Mg_2p_key_, Mg_2p);
    n->Store (CO3_2m_key_, CO3_2m);


    // and back to mol/kg of soil
    CaCO3()     /= coeff;
    CaMgCO32() /= coeff;

    n->Store (CaCO3_key_, CaCO3);
    n->Store (CaMgCO32_key_, CaMgCO32);

  }
}
```

```
/*
 * Check function return value...
 *     opt == 0 means SUNDIALS function allocates memory so check if
 *              returned NULL pointer
 *     opt == 1 means SUNDIALS function returns a flag so check if
 *              flag >= 0
 *     opt == 2 means function allocates memory so check if returned
 *              NULL pointer
 */
template<size_t dim>
int ChemicalEquilibriumVisitor<dim>::check_flag(void *flagvalue,
char *funcname, int opt)
{
  int *errflag;

  /* Check if SUNDIALS function returned NULL pointer -
  no memory allocated */
  if (opt == 0 && flagvalue == NULL) {
    fprintf(stderr,
            "\nSUNDIALS_ERROR: %s() failed - returned NULL pointer\n\n",
            funcname);
    return(1);
  }

  /* Check if flag < 0 */
  else if (opt == 1) {
    errflag = (int *) flagvalue;
    if (*errflag < 0) {
      fprintf(stderr,
              "\nSUNDIALS_ERROR: %s() failed with flag = %d\n\n",
              funcname, *errflag);
      return(1);
    }
  }
```

```
  /* Check if function returned NULL pointer - no memory allocated */
  else if (opt == 2 && flagvalue == NULL) {
    fprintf(stderr,
            "\nMEMORY_ERROR: %s() failed - returned NULL pointer\n\n",
            funcname);
    return(1);
  }

  return(0);
}

/*
 * System function
 */
static int func(N_Vector C, N_Vector f, void *user_data)
{
  UserData data;
  data = (UserData)user_data;

  realtype *cd, *fd;

  //! equilibrium constants
  //! log(K)s actually
  realtype K1 = -8.47, K2 = -17.17;

  //! basic(primary) species concentrations
  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, CaCO3, CaMgCO32;

  //! total initial concentrations of basic species
  realtype T_Ca_2p_init = data->T_Ca_2p,
           T_Mg_2p_init = data->T_Mg_2p,
           T_H_1p_init = data->T_H_1p,
           T_CO3_2m_init = data->T_CO3_2m;
  //! and mineral mole numbers
  realtype CaCO3_init = data->n_cc;
  realtype CaMgCO32_init = data->n_d;

  std::vector<double> result(NPRIM, 0.); // lenght, value
  std::vector<double> a (NPRIM+NSEC, 0.);
  ActivityCoefficients(C, a, data);

  realtype a_Ca_2p, a_Mg_2p, a_CO3_2m;
  a_Ca_2p  = a[0];
  a_Mg_2p  = a[1];
  a_CO3_2m = a[3];
  TotalActivities(C, a, result);


  realtype T_Ca_2p      = result[0],
           T_Mg_2p      = result[1],
           T_H_1p       = result[2],
           T_CO3_2m     = result[3];

  cd = NV_DATA_S(C);
```

```
  fd = NV_DATA_S(f);

  Ca_2p     = cd[0];
  Mg_2p     = cd[1];
  H_1p      = cd[2];
  CO3_2m    = cd[3];
  CaCO3     = cd[4];
  CaMgCO32 = cd[5];

  fd[0] = T_Ca_2p    + CaCO3 + CaMgCO32 - T_Ca_2p_init    -
  CaCO3_init - CaMgCO32_init;
  fd[1] = T_Mg_2p    + CaMgCO32    - T_Mg_2p_init    -
  CaMgCO32_init ;
  fd[2] = T_H_1p     - T_H_1p_init;
  fd[3] = T_CO3_2m + CaCO3 + 2*CaMgCO32 - T_CO3_2m_init -
  CaCO3_init - 2*CaMgCO32_init ;

  if (data->flag_cc)
    fd[4] = Ca_2p + a_Ca_2p  + CO3_2m + a_CO3_2m - K1;
  else
    fd[4] = CaCO3;

  if (data->flag_d)
    fd[5] = Ca_2p + a_Ca_2p + Mg_2p + a_Mg_2p + 2*CO3_2m
    +2*a_CO3_2m - K2;
  else
    fd[5] = CaMgCO32;

  return 0;
}

static int func_init(N_Vector C, N_Vector f, void *user_data)
{
  UserData data;
  data = (UserData)user_data;

  realtype *cd, *fd;

  //! equilibrium constants
  //! log(K)s actually
  realtype K1 = -8.47, K3 = 14.01, K4 = -10.31, K5 = -16.71, K6 = -3.23,
  K7 = -2.98;

  //! basic(primary) species concentrations
  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, CaCO3, CaMgCO32, T_Ca_2p, T_Mg_2p,
  T_CO3_2m;
  realtype OH_1m, HCO3_1m;

  //! total initial concentrations of basic species
  realtype T_Ca_2p_init = data->T_Ca_2p,
           T_CO3_2m_init = data->T_CO3_2m,
           CaCO3_init = data->n_cc;
           H_1p = data->H_1p;
```

```
  cd = NV_DATA_S(C);
  fd = NV_DATA_S(f);

  Ca_2p     = cd[0];
  CO3_2m    = cd[1];
  CaCO3      = cd[2];

  //!activity calculations
  realtype I, A, a_Ca_2p, a_CO3_2m, a_HCO3_1m;
  OH_1m =  log10(pow(10., - H_1p - K3));
  HCO3_1m = log10(pow(10.,   H_1p + CO3_2m - K4));

  I = 0.5*  ( pow(10., Ca_2p)*4 + pow(10.,H_1p) +
  pow(10.,CO3_2m)*4 +  pow(10., OH_1m) + pow(10., HCO3_1m) );
  A = -0.5*(sqrt(I)/(1+sqrt(I)) - 0.3*I);
  a_Ca_2p  = 4*A;
  a_CO3_2m = 4*A;
  a_HCO3_1m  = A;

  T_Ca_2p       = pow(10., Ca_2p) + pow(10., Ca_2p + a_Ca_2p +
  CO3_2m + a_CO3_2m - K6);
  T_CO3_2m      = pow(10., CO3_2m) + pow(10., H_1p + CO3_2m +
  a_CO3_2m - K4 - a_HCO3_1m) + pow(10., CO3_2m + a_CO3_2m +
  2*H_1p - K5) + pow(10., Ca_2p + a_Ca_2p + CO3_2m + a_CO3_2m
  - K6);

  fd[0] = CaCO3 + T_Ca_2p - CaCO3_init - T_Ca_2p_init;
  fd[1] = CaCO3 + T_CO3_2m - CaCO3_init - T_CO3_2m_init;
  fd[2] = Ca_2p + a_Ca_2p + CO3_2m + a_CO3_2m - K1;

  return 0;
}

static int jac(long int N, N_Vector C, N_Vector f, DlsMat J,
void *user_data, N_Vector tmp1, N_Vector tmp2)
{
  //! basic(primary) species concentrations
  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, CaCO3, CaMgCO32;

  realtype *cd, *fd;
  cd = NV_DATA_S(C);

  UserData data;
  data = (UserData)user_data;

  //! equilibrium constants
  //! log(K)s actually
  realtype K3 = 14.01, K4 = -10.31, K5 = -16.71, K6 = -3.23, K7 = -2.98;

  Ca_2p     = cd[0];
  Mg_2p     = cd[1];
  H_1p      = cd[2];
  CO3_2m    = cd[3];
  CaCO3     = cd[4];
```

```
  CaMgCO32 = cd[5];

  DENSE_ELEM(J, 0, 0) = log(10)*(pow(10., Ca_2p) + pow(10., Ca_2p
  + CO3_2m - K6));
  DENSE_ELEM(J, 0, 3) = log(10)*pow(10., Ca_2p + CO3_2m - K6);
  DENSE_ELEM(J, 0, 4) = 1.;
  DENSE_ELEM(J, 0, 5) = 1.;

  DENSE_ELEM(J, 1, 1) = log(10)*(pow(10., Mg_2p) + pow(10., Mg_2p
  + CO3_2m - K7));
  DENSE_ELEM(J, 1, 3) = log(10)*pow(10., Mg_2p + CO3_2m - K7);
  DENSE_ELEM(J, 1, 5) = 1.;

  DENSE_ELEM(J, 2, 2) = log(10)*(pow(10., H_1p) + pow(10., -H_1p - K3)
  + pow(10., H_1p + CO3_2m - K4) + 4*pow(10., 2*H_1p + CO3_2m - K5));
  DENSE_ELEM(J, 2, 3) = log(10)*(pow(10., H_1p + CO3_2m - K4)
  + pow(10., CO3_2m + 2*H_1p - K5));

  DENSE_ELEM(J, 3, 0) = log(10)*pow(10., Ca_2p + CO3_2m - K6);
  DENSE_ELEM(J, 3, 1) = log(10)*pow(10., Mg_2p + CO3_2m - K7);
  DENSE_ELEM(J, 3, 2) = log(10)*(pow(10., H_1p + CO3_2m - K4)
  + 2*pow(10., CO3_2m + 2*H_1p - K5));
  DENSE_ELEM(J, 3, 3) = log(10)*(pow(10., CO3_2m)
  + pow(10., H_1p + CO3_2m - K4) + pow(10., CO3_2m + 2*H_1p - K5)
  + pow(10., Ca_2p + CO3_2m - K6)+ pow(10., Mg_2p + CO3_2m - K7));
  DENSE_ELEM(J, 3, 4) = 1.;
  DENSE_ELEM(J, 3, 5) = 2.;

  if (data->flag_cc)
  {
    DENSE_ELEM(J, 4, 0) = 1.;
    DENSE_ELEM(J, 4, 3) = 1.;
  }
  else
    DENSE_ELEM(J, 4, 4) = 1.;


  if (data->flag_d)
  {
    DENSE_ELEM(J, 5, 0) = 1.;
    DENSE_ELEM(J, 5, 1) = 1.;
    DENSE_ELEM(J, 5, 3) = 2.;
  }
  else
    DENSE_ELEM(J, 5, 5) = 1.;


  return 0;
}

/*
 * Print solution
 */
template<size_t dim>
```

```
void ChemicalEquilibriumVisitor<dim>::PrintOutput(N_Vector C)
{
    printf("\nCa+2\t=\t%e",   NV_Ith_S(C,0));
    printf("\nMg+2\t=\t%e",   NV_Ith_S(C,1));
    printf("\nH+\t=\t%e",    NV_Ith_S(C,2));
    printf("\nCO3-2\t=\t%e", NV_Ith_S(C,3));
    printf("\nCaCO3 mole number = %e SI = %e", NV_Ith_S(C,4),
    NV_Ith_S(C,0)+NV_Ith_S(C,3));
    printf("\nCaMgCO32 mole number = %e SI = %e", NV_Ith_S(C,5),
    NV_Ith_S(C,0)+NV_Ith_S(C,1)+2*NV_Ith_S(C,3));
}

/*
 * Print final statistics
 */
template<size_t dim>
void ChemicalEquilibriumVisitor<dim>::PrintFinalStats(void *kmem)
{
  long int nni, nfe, nje, nfeD;
  int flag;

  flag = KINGetNumNonlinSolvIters(kmem, &nni);
  check_flag(&flag, "KINGetNumNonlinSolvIters", 1);
  flag = KINGetNumFuncEvals(kmem, &nfe);
  check_flag(&flag, "KINGetNumFuncEvals", 1);

  printf("\nFinal Statistics..\n");
  printf("nni    = %5ld    nfe   = %5ld \n", nni, nfe);

  realtype fnorm, steplength;
  flag = KINGetFuncNorm(kmem, &fnorm);
  printf("\nfnorm = %e", fnorm);
  flag = KINGetStepLength(kmem, &steplength);
  printf("\nsteplength = %e\n\n", steplength);

}

void TotalConcentrations(N_Vector C, std::vector<double>& result)
{
  realtype *cd;
  cd = NV_DATA_S(C);
  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, CaCO3, CaMgCO32;
  realtype T_Ca_2p, T_Mg_2p, T_H_1p, T_CO3_2m;

  // equilibrium constants, log(K)s actually
  realtype K1 = -8.47, K2 = -17.17, K3 = 14.01, K4 = -10.31, K5 = -16.71,
  K6 = -3.23, K7 = -2.98;

  //! reactions
  //  CaCO3    =            + Ca_2p +          CO3_2m          (K1)
  //  CaMgCO32 =            + Ca_2p + Mg_2p + 2CO3_2m          (K2)
  //  OH_1p    = H2O - H_1p                                    (K3)
  //  HCO3_1m  =       H_1p                 +  CO3_2m          (K4)
  //  CO2      =-H2O + 2*H_1p                +  CO3_2m          (K5)
```

```
  //  CaCO3_aq =              Ca_2p +          CO3_2m          (K6)
  //  MgCO3_aq =                       Mg_2p + CO3_2m          (K7)
  //  Cl       =       Cl_1p

  Ca_2p     = cd[0];
  Mg_2p     = cd[1];
  H_1p      = cd[2];
  CO3_2m    = cd[3];
  CaCO3     = cd[4];
  CaMgCO32  = cd[5];

  T_Ca_2p       = pow(10., Ca_2p) + pow(10., Ca_2p + CO3_2m - K6);
  T_Mg_2p       = pow(10., Mg_2p) + pow(10., Mg_2p + CO3_2m - K7);
  T_H_1p        = pow(10., H_1p) - pow(10., - H_1p - K3)
                                 + pow(10.,    H_1p + CO3_2m - K4)
                                 + 2*pow(10., 2*H_1p + CO3_2m - K5);
  T_CO3_2m      = pow(10., CO3_2m) + pow(10., H_1p + CO3_2m - K4)
                                   + pow(10., CO3_2m + 2*H_1p - K5)
                                   + pow(10., Ca_2p + CO3_2m - K6)
                                   + pow(10., Mg_2p + CO3_2m - K7);

  result[0] = T_Ca_2p;
  result[1] = T_Mg_2p;
  result[2] = T_H_1p;
  result[3] = T_CO3_2m;

}

void ActivityCoefficients(N_Vector C, std::vector<double>& a,
void *user_data)
{
  //Davies activity model
  // log gamma = -A*z^2*(sqrt(I)/(1+sqrt(I)) - 0.3*I)
  // A = 0.5

  realtype *cd;
  cd = NV_DATA_S(C);

  UserData data;
  data = (UserData)user_data;

  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, OH_1m, HCO3_1m, Cl_1m, I, A;
  Ca_2p     = cd[0];
  Mg_2p     = cd[1];
  H_1p      = cd[2];
  CO3_2m    = cd[3];

  realtype K3 = 14.01, K4 = -10.31;
  OH_1m =  log10(pow(10., - H_1p - K3));
  HCO3_1m = log10(pow(10.,    H_1p + CO3_2m - K4));
  Cl_1m = log10(data->T_Cl_1m);

  I = 0.5*(pow(10., Ca_2p)*4 + pow(10., Mg_2p)*4 + pow(10.,H_1p) +
  pow(10.,CO3_2m)*4 +  pow(10., OH_1m) + pow(10., HCO3_1m)
```

```
  + pow(10., Cl_1m));
  A = -0.5*(sqrt(I)/(1+sqrt(I)) - 0.3*I);

  a[0] = 4*A;   //a_Ca_2p
  a[1] = 4*A;   //a_Mg_2p
  a[2] = A;     //a_H_1p
  a[3] = 4*A;   //a_CO3_2m
  a[4] = A;     //a_OH_1m
  a[5] = A;     //a_HCO3_1m
}

void TotalActivities(N_Vector C, std::vector<double>& a,
std::vector<double>& result)
{
  realtype *cd;
  cd = NV_DATA_S(C);
  realtype H_1p, Ca_2p, Mg_2p, CO3_2m, CaCO3, CaMgCO32;
  realtype a_H_1p, a_Ca_2p, a_Mg_2p, a_CO3_2m, a_OH_1m, a_HCO3_1m;
  realtype T_Ca_2p, T_Mg_2p, T_H_1p, T_CO3_2m;

  // equilibrium constants, log(K)s actually
  realtype K1 = -8.47, K2 = -17.17, K3 = 14.01, K4 = -10.31, K5 = -16.71,
  K6 = -3.23, K7 = -2.98;

  //! reactions
  //  CaCO3    =            + Ca_2p +          CO3_2m          (K1)
  //  CaMgCO32 =            + Ca_2p + Mg_2p + 2CO3_2m          (K2)
  //  OH_1p    = H20 - H_1p                                    (K3)
  //  HCO3_1m  =       H_1p                 +  CO3_2m          (K4)
  //  CO2      =-H2O + H_1p + HCO3_1m                          (K5)
  //  H2CO3    =       H_1p + HCO3_1m                          (K6)
  //  CaCO3_aq =            Ca_2p +          CO3_2m          (K7)
  //  MgCO3_aq =                   Mg_2p +  CO3_2m          (K8)
  //  Cl       =       Cl_1p

  Ca_2p     = cd[0];
  Mg_2p     = cd[1];
  H_1p      = cd[2];
  CO3_2m    = cd[3];
  CaCO3     = cd[4];
  CaMgCO32  = cd[5];

  a_Ca_2p   = a[0];
  a_Mg_2p   = a[1];
  a_H_1p    = a[2];
  a_CO3_2m  = a[3];
  a_OH_1m   = a[4];
  a_HCO3_1m = a[5];

  T_Ca_2p      = pow(10., Ca_2p) + pow(10., Ca_2p + a_Ca_2p + CO3_2m +
  a_CO3_2m - K6);
  T_Mg_2p      = pow(10., Mg_2p) + pow(10., Mg_2p + a_Mg_2p + CO3_2m +
  a_CO3_2m - K7);
  T_H_1p       = pow(10., H_1p) - pow(10., - H_1p - a_H_1p - K3 -
```

```
  a_OH_1m) + pow(10.,    H_1p + a_H_1p + CO3_2m + a_CO3_2m - K4 -
  a_HCO3_1m) + 2*pow(10., 2*H_1p + 2*a_H_1p + CO3_2m + a_CO3_2m - K5);
  T_CO3_2m      = pow(10., CO3_2m) + pow(10., H_1p + a_H_1p + CO3_2m +
  a_CO3_2m - K4 - a_HCO3_1m) + pow(10., CO3_2m + a_CO3_2m + 2*H_1p
  +2*a_H_1p - K5);

  result[0] = T_Ca_2p;
  result[1] = T_Mg_2p;
  result[2] = T_H_1p;
  result[3] = T_CO3_2m;

}
template class ChemicalEquilibriumVisitor<1U>;
template class ChemicalEquilibriumVisitor<2U>;
template class ChemicalEquilibriumVisitor<3U>;

} //csmp
```

Appendix B

CSMP++GEM source code

This section contains the complete C++ code listing of the new CSMP++GEM reactive transport simulator described in detail in Chapter 4 and applied to the RTM simulations of the Benicàssim case study in Chapter 5. The KozenyCarmanVisitor class is created specifically for the calculation of the porosity/permeability update on the node-by-node basis and can be replaced by another visitor with the same function. The chemical equilibrium calculations are performed using the GEMS3K standalone code, with its function calls wrapped in the GEMS3K_Visitor within the CSMP++ library. The RTM_Simulator class contains all the auxiliary functions (e.g. fluid and rock properties update) as well as the main time loop with transient pressure and temperature, transport of solutes and chemical equilibrium calculations.

B.1 main.cpp

```
#include "GEMS3K_Visitor.h"
#include "RTM_Simulator.h"
#include <iostream>

using namespace std;
using namespace csmp;

int main(int argc, char **argv)
{
  try
  {
    RTM_Simulator<3U> rtm3d(argv);
    rtm3d.run();
  }
  catch (csmp::Exception& e)
  {
    e.Out();
    return -1;
  }
```

```
  return 0;
}
```

B.2 KozenyCarmanVisitor.h

```
#ifndef KOZENY_CARMAN_VISITOR_H
#define KOZENY_CARMAN_VISITOR_H

/** @file KozenyCarmanVisitor.h
 *  @author Alina Yapparova
 *  @brief Kozeny-Carman porosity/permeability corelation
 *  @details computes permeability from given porosity
 *  @date 27.04.2015
 */

#include "Visitor.h"
#include "Model.h"

namespace csmp{

template<size_t dim>
class KozenyCarmanVisitor: public Visitor<dim>
{
  public:
    KozenyCarmanVisitor( Model<dim>& model, Index poro_new_key,
    Index poro_key, Index perm_key );
    ~KozenyCarmanVisitor();

    virtual void Visit( Element<dim>* e);
    virtual void Visit( Model<dim>* m);

  private:
    KozenyCarmanVisitor();

    Model<dim>& model;
    Index poro_new_key, poro_key, perm_key;
    ScalarVariable phi_new, phi, k_new, k;

};

}//csmp
#endif
```

B.3 KozenyCarmanVisitor.cpp

```
#include "KozenyCarmanVisitor.h"


namespace csmp{

template<size_t dim>
```

```
KozenyCarmanVisitor<dim>::KozenyCarmanVisitor( Model<dim>& model,
Index poro_new_key, Index poro_key, Index perm_key):
  Visitor<dim>( MODEL, ELEMENT ),
  model (model),
  poro_new_key (poro_new_key),
  poro_key (poro_key),
  perm_key (perm_key)
  {

  }

template<size_t dim>
KozenyCarmanVisitor<dim>::~KozenyCarmanVisitor(){}

template<size_t dim>
void KozenyCarmanVisitor<dim>::Visit( Model<dim>* m ){}

template<size_t dim>
void KozenyCarmanVisitor<dim>::Visit( Element<dim>* e )
{
  k       = e->Read(perm_key);
  phi     = e->Read(poro_key);
  phi_new = e->Read(poro_new_key);
  if (phi_new() == 0.)
    k_new() = 0.;
  else if (phi() == 1.)
    k_new() = k();
  else
    k_new() = k()*(1-phi())*(1-phi())/(1-phi_new())/
    (1-phi_new())*phi_new()*phi_new()*phi_new()
    /phi()/phi()/phi();
  e->Store(perm_key, k_new);
}

template class KozenyCarmanVisitor<1U>;
template class KozenyCarmanVisitor<3U>;

}//csmp
```

B.4 GEMS3K_Visitor.h

```
#ifndef GEMS3K_VISITOR_H
#define GEMS3K_VISITOR_H

/** @file GEMS3K_Visitor.h
 *  @author Alina Yapparova
 *  @brief  Aqueous equilibrium calcuation
 *  @details computes equilibrium concentrations of aqueous ions,
 *  complexes and minerals using GEMS3K library
 *  @date 01.07.2014
 */

#include "Visitor.h"
```

```
#include "Model.h"

//GEMS3K
#include "../../../support_libraries/gems3k/source/nodearray.h"

namespace csmp{

template<size_t dim>
class GEMS3K_Visitor: public Visitor<dim>
{
  public:
    GEMS3K_Visitor( Model<dim>& model, const char* system_file_list_name,
    const char* recipes_file_list_name);
    ~GEMS3K_Visitor();

    void SetInitialProperties();
    virtual void Visit( Node<dim>* n);
    virtual void Visit( Model<dim>* m);

    int GetnIC();
    int GetnPH();
    int GetnPS();

    int SetTime(double timestep, double totaltime, int step);

    int NbyIdx(int idx);
    int IdxbyN(int idx);

  private:
    GEMS3K_Visitor();

    Model<dim>& model;
    const Index recipe_index_key;
    TNodeArray* na;

    int nIC, nPH, nPS;
    int nNodes;
    int node_index;
    long int* recipe_index;

    double dt, time;
    int step;

    Index p_key, T_key, X_key, nphi_key, inert_vol_key, vol_key,
    fv_key, Q_key, abPS_old_key, abPS_key, abSP_old_key, abSP_key,
    q_chem_key, q_chem_prev_key, axPH_key, avPH_key, amPH_key,
    pH_key, pe_key, Eh_key;

    ArrayVariable *abPS_node, *abSP_node, *bIC_node, *q_chem_node,
    *q_chem_prev_node, *axPH_node, *avPH_node, *amPH_node;

    ScalarVariable X, solution_volume, inert_volume, nphi, nphi_new,
    fv, Q, pH_node, pe_node, Eh_node;
```

```
};

}//csmp
#endif
```

B.5 GEMS3K_Visitor.cpp

```
#include "GEMS3K_Visitor.h"


namespace csmp{

template<size_t dim>
GEMS3K_Visitor<dim>::GEMS3K_Visitor( Model<dim>& model,
const char* system_file_list_name,
const char* recipes_file_list_name):
  Visitor<dim>( MODEL, NODE ),
  model (model),
  recipe_index_key (model.Database().StorageKey("recipe_index"))
  {
    model.Region("Model").RenumberNodes();

    nNodes = model.Region("Model").Nodes();
    na = new TNodeArray( nNodes );

    recipe_index = new long int[nNodes];
    ScalarVariable ind;

    const typename vector<Node<dim>*>::const_iterator modelNodesEnd(
    model.Region("Model").NodesEnd() );
    for( typename vector<Node<dim>*>::const_iterator n(
    model.Region("Model").NodesBegin() ); n != modelNodesEnd; ++n )
    {
        node_index = (*n)->Idx();
        (*n)->Read(recipe_index_key, ind);
        recipe_index[node_index] = static_cast<int>(ind());
    }

    //! Read GEMS input files
    if( na->GEM_init( system_file_list_name,
    recipes_file_list_name, recipe_index, false ) )
    {
          cout << "Error occured during reading the files";
          cin.get();
    }

    nIC = GetnIC();
    nPH = GetnPH();
    nPS = GetnPS();

    abPS_node          = new ArrayVariable(nIC-1, 0.);
    abSP_node          = new ArrayVariable(nIC-1, 0.);
    bIC_node           = new ArrayVariable(nIC-1, 0.);
```

```
    axPH_node          = new ArrayVariable(nPH-nPS, 0.);
    avPH_node          = new ArrayVariable(nPH-nPS, 0.);
    amPH_node          = new ArrayVariable(nPH-nPS, 0.);

  }

template<size_t dim>
GEMS3K_Visitor<dim>::~GEMS3K_Visitor()
{
  delete na;
  delete[] recipe_index;
  delete abPS_node;
  delete abSP_node;
  delete bIC_node;
  delete axPH_node;
  delete avPH_node;
  delete amPH_node;

}

template<size_t dim>
int GEMS3K_Visitor<dim>::GetnIC()
{
    return na->pCSD()->nICb; //! number of independent components;
}

template<size_t dim>
int GEMS3K_Visitor<dim>::GetnPH()
{
    return na->pCSD()->nPHb; //! number of phases;
}

template<size_t dim>
int GEMS3K_Visitor<dim>::GetnPS()
{
    return na->pCSD()->nPSb; //! number of phases solutions;
}

template<size_t dim>
int GEMS3K_Visitor<dim>::SetTime(double timestep, double totaltime,
int n)
{
    dt = timestep;
    time = totaltime;
    step = n;
    return 0;
}

template<size_t dim>
void GEMS3K_Visitor<dim>::SetInitialProperties ()
{
    p_key           = model.Database().StorageKey("fluid_pressure");
    T_key           = model.Database().StorageKey("temperature");
    X_key           = model.Database().StorageKey("fluid_salinity");
```

```
    nphi_key        = model.Database().StorageKey("nodal porosity");
    abPS_key        = model.Database().StorageKey("abPS");
    abSP_key        = model.Database().StorageKey("abSP");
    axPH_key        = model.Database().StorageKey("axPH");
    avPH_key        = model.Database().StorageKey("avPH");
    amPH_key        = model.Database().StorageKey("amPH");
    vol_key         = model.Database().StorageKey("solution volume");
    inert_vol_key   = model.Database().StorageKey("inert volume");
    fv_key          = model.Database().StorageKey("finite volume");
    Q_key           = model.Database().StorageKey("chemical source");
    pH_key           = model.Database().StorageKey("pH");
    pe_key           = model.Database().StorageKey("pe");
    Eh_key           = model.Database().StorageKey("Eh");

    //! Initial equilibration
    long int  NodeStatusCH_;
    long int Mode = NEED_GEM_AIA;
    //forcing AIA mode for initial equilibration

    cout<<"\nGEMS3K_Visitor::SetInitialProperties()
    initial equilibration"<<endl;

    const typename vector<Node<dim>*>::const_iterator modelNodesEnd(
    model.Region("Model").NodesEnd() );
    for( typename vector<Node<dim>*>::const_iterator n(
    model.Region("Model").NodesBegin() ); n != modelNodesEnd; ++n )
    {
        node_index = (*n)->Idx();

        //! setting up pressure and Temperature at each node
        na->pNodT1()[(node_index)]->TK = (*n)->Read(T_key) + 273.15;
        //!from Celsius to Kelvin
        na->pNodT1()[(node_index)]->P  = (*n)->Read(p_key);

        //! calculating equilibrium by calling GEMS3K,
        // getting the status back
        NodeStatusCH_ = na->RunGEM( node_index, Mode);
        if( !( NodeStatusCH_ == OK_GEM_AIA ||
        NodeStatusCH_ == OK_GEM_SIA ) )
        {
              cout << "Error occured during re-calculating equilibrium,
              node index = "<<node_index <<" Status = "<<
              NodeStatusCH_ <<endl;
              cin.get();
        }


        for (int i=0;i<nIC-1;i++)
        {
          (*abPS_node)(i) = node1_bPS(node_index, 0, i);
          //!phase 0 -- aqueous phase
          //! abSP is set up in GEMS in units of mass or moles
          per pore volume
          (*abSP_node)(i) = node1_bSP(node_index, i);
```

```
}

//! new solution volume, normalized. can be used to control
the scheme accuracy -- should be always around 1[l]
solution_volume() = node1_vPS(node_index, 0)*1000.;
//scaled from m3 to liters

(*n)->Store(abPS_key, *abPS_node);
(*n)->Store(abSP_key, *abSP_node);

nphi() = (*n)->Read(nphi_key);
inert_volume() = 1. - nphi();
//! store mineral amounts (in moles) and mineral volumes
for (int i=0;i<nPH-nPS;i++)
{
  (*axPH_node)(i) = node1_xPH(node_index, i+nPS)*nphi();
  (*avPH_node)(i) = node1_vPH(node_index, i+nPS)*nphi()*1000.;
  // volume fraction in liters per liter
  (*amPH_node)(i) = node1_mPH(node_index, i+nPS)*nphi();
  // minerals mass in kg per liter
  inert_volume() -= (*avPH_node)(i);
}
(*n)->Store(axPH_key, *axPH_node);
(*n)->Store(avPH_key, *avPH_node);
(*n)->Store(amPH_key, *amPH_node);
(*n)->Store(inert_vol_key, inert_volume);

//! Store pH, pe and Eh
pH_node() = node1_pH (node_index);
pe_node() = node1_pe (node_index);
Eh_node() = node1_Eh (node_index);
(*n)->Store(pH_key, pH_node);
(*n)->Store(pe_key, pe_node);
(*n)->Store(Eh_key, Eh_node);

//! calculate salinity and store it
//! X = 1 - mass_of_pure_water/mass_of_solution
//! salinity                    molar mass of water in kg/mol
X() =  1. - node1_xPA(node_index, 0)*
0.01801528/node1_mPS(node_index, 0);
(*n)->Store(X_key, X);

//! and solution volume (should be 1 liter)
(*n)->Store(vol_key, solution_volume);

//! chemical source
Q() = 0.;
(*n)->Store(Q_key, Q);
//model.Region("Model").N(NbyIdx(k))->Store(Q_key, Q);

if (recipe_index[node_index] == 0)
  {
    (*n)->Status(abPS_key, DIRICH);
    cout<<"\n node_index = "<<node_index;
```

```
        }
    }
}

template<size_t dim>
void GEMS3K_Visitor<dim>::Visit( Model<dim>* m ) {}

//! SNIA
template<size_t dim>
void GEMS3K_Visitor<dim>::Visit( Node<dim>* n )
{
    node_index = n->Idx();
    long int  NodeStatusCH_;
    //long int Mode = NEED_GEM_SIA;
    long int Mode = NEED_GEM_AIA;
    bool accept_GEMS_results;

    if (recipe_index[node_index])  // do not equilibrate boundary nodes
    {
        //! bIC = abPS + abSP
        n->Read(abPS_key, *abPS_node);
        n->Read(abSP_key, *abSP_node);

        //! setting up pressure and Temperature at each node
        na->pNodT1()[(node_index)]->TK = n->Read(T_key) + 273.15;
        //!from Celsius to Kelvin
        na->pNodT1()[(node_index)]->P  = n->Read(p_key);

        //! setting up initial bulk composition at each node
        (*bIC_node) = (*abPS_node) + (*abSP_node);
        for (int i=0;i<nIC-1;i++)
            node1_bIC(node_index, i) = (*bIC_node)[i];

        NodeStatusCH_ = na->RunGEM( node_index, Mode);

        if( !( NodeStatusCH_ == OK_GEM_AIA ||
        NodeStatusCH_ == OK_GEM_SIA ) )
         {
               cout << "Error occured during re-calculating
                equilibrium, node_index = "<<node_index<<" Status =
                "<< NodeStatusCH_ <<endl;
               accept_GEMS_results = false;
         }
         else
               accept_GEMS_results = true;

        if (accept_GEMS_results)
         {
              for (int i=0;i<nIC-1;i++)
              {
                  //! assign new values to abSP
                  (*abSP_node)(i) = node1_bSP(node_index, i);
                  //! and abPS
                  (*abPS_node)(i) = node1_bPS(node_index, 0, i);
```

```
}

//!read old amPH
n->Read(amPH_key, *amPH_node);
//! calculate chemical source Q =
//sum_i (mPH_old - mPH_new)_i
Q() = n->Read(Q_key);
fv = n->Read(fv_key);
for (int i=0;i<nPH-nPS;i++)
  Q() += (*amPH_node)(i)*fv();

//! read old nodal porosity
nphi = n->Read(nphi_key);
//! read inert volume fraction
inert_volume = n->Read(inert_vol_key);

//! calculate new nodal porosity
//nphi_new = 1 - inert_volume - sum_i (avPH_new_i)
nphi_new() = 1.;
for (int i=0;i<nPH-nPS;i++)
{
    (*avPH_node)(i) = node1_vPH(node_index, i+nPS)*1000.;
    nphi_new() += (*avPH_node)(i);
}
nphi_new() = (1.-inert_volume())/nphi_new();

for (int i=0;i<nPH-nPS;i++)
{
    //! mineral amounts (for output)
    (*axPH_node)(i) = node1_xPH(node_index, i+nPS)*
    nphi_new();
    //! new mineral volumes are used to re-calculate
    //porosity
    (*avPH_node)(i) *= nphi_new();
    (*amPH_node)(i) = node1_mPH(node_index, i+nPS)*
    nphi_new(); // minerals mass in kg per liter
    Q() -= (*amPH_node)(i)*fv();
 }

 if ( nphi_new() <= 0. )
    throw csmp::Exception (FATAL_ERROR,
    "GEMS3K_Visitor::Visit()",
    "Porosity is zero or negative");

 //! scale concentrations to the new porosity
 //(for mass conservation)
 for (int i=0;i<nIC-1;i++)
 {
      (*abSP_node)(i) *= nphi()/nphi_new();
      (*abPS_node)(i) *= nphi()/nphi_new();
 }

 //! store new values:
 //! abSP
```

```
                n->Store(abSP_key, *abSP_node);

                //! abPS
                n->Store(abPS_key, *abPS_node);

                //! mineral amounts
                n->Store(axPH_key, *axPH_node); //in mol/m3
                n->Store(avPH_key, *avPH_node); // volume fraction
                n->Store(amPH_key, *amPH_node); // mass
                n->Store(nphi_key, nphi_new);   // new porosity

                //! Store pH, pe and Eh
                pH_node() = node1_pH (node_index);
                pe_node() = node1_pe (node_index);
                Eh_node() = node1_Eh (node_index);
                n->Store(pH_key, pH_node);
                n->Store(pe_key, pe_node);
                n->Store(Eh_key, Eh_node);

                //! salinity    molar mass of water in kg/mol
                X() =  1. - node1_xPA(node_index, 0)*
                0.01801528/node1_mPS(node_index, 0);
                n->Store(X_key, X);

                //! store chemical source
                n->Store(Q_key, Q);

                //! new solution volume, normalized
                solution_volume() = node1_vPS(node_index, 0)*1000.;
                //scaled from m3 to liters
                n->Store(vol_key, solution_volume);

        }
    }
}
```

B.6 RTM_Simulator.h

```
#ifndef RTM_SIMULATOR_H
#define RTM_SIMULATOR_H

/** @file RTM_Simulator.h
 *  @author Alina Yapparova
 *  @brief  General Reactive Transport Modelling Simulator
 *  @details uses GEMS3K library for chemical speciation calculations and
 *  sequential iterative approach for chemistry-transport coupling
 *  @date 06.11.2014
 */

// model
#include "ANSYS_Model3D.h"
#include "Model1D.h"
```

```
// File I/O and Initialization
#include "ComputationalSettings.h"
#include "InputDataManager.h"

// finite elements
#include "PDE_Integrator.h"
#include "NumIntegral_NT_lhsop_N_dV.h"
#include "NumIntegral_NT_op_N_dV.h"
#include "NumIntegral_dNT_op_dN_dV.h"
#include "NumIntegral_dNT_op_dV.h"
#include "PointSource_rhsop.h"
#include "NumIntegral_SetRHS_to_Zero.h"
#include "VelocityAndVolumeFlux.h"

// finite volumes
#include <ExplicitNodeCenteredFiniteVolumeTransport.h>

// visitors
#include "ConductivityVisitor.h"
#include "ComputeGravityTermVisitor.h"
#include "GEMS3K_Visitor.h"
#include "KozenyCarmanVisitor.h"

// eos
#include "Brine.h"

// Solver
#include "SAMG_Solver.h"
//#include "LUdcmp_Solver.h"

// Output
#include "VTU_Interface.h"
#include <time.h>

#include "iostream"
using namespace std;

namespace csmp
{
  template<size_t dim>
  class RTM_Simulator
  {
    public:
          RTM_Simulator (char** argv );
          ~RTM_Simulator ();
          int run_old();
          int run();
          int test_transport();
          int test_pressure_temperature();
    private:
          const char* geometry_name_;
          const char* config_file_name_;
          const char* vars_name_;
          const char* system_file_list_name_;
```

```
const char* recipes_file_list_name_;
list<string> output_props_;

Model<dim>* model;
PropertyDatabase<dim>*  pd_ref;
VTU_Interface<dim>* vtu;
ComputationalSettings  run_settings;
//LUdcmp_Solver *p_solver,  *pt_solver,  *t_solver;
SAMG_Solver   *p_solver, *pt_solver,  *t_solver;

//! equation of state
double64  temperature, pressure, salinity;
Brine     seawater;

//! pde_integrator is used to solve the transient pressure
//equation using finite elements
PDE_Integrator<dim, Region>*  transient_pressure;
MathOperatorLHS<dim> *pt_capacitance_lhs, *pt_conductance,
*velocity;
MathOperatorRHS<dim> *pt_capacitance_rhs, *pt_gravity,
                     *nodal_fluid_src, *fluid_expansion_src,
                     *chemical_src;

//! an instance of a FV transport class is used to solve for
//the species transport
NodeCenteredFiniteVolumeTransport<dim> *transport_abPS;
//! and heat transport
NodeCenteredFiniteVolumeTransport<dim> *heat_transport;

//! time variables
double64        time_increment;
double64        total_time;
double64        global_time;
time_t          start,end;

size_t timestep, sia_counter;
bool accept_GEMS_results, tstep_accept;
double eps, min_err, max_err, err;

int nIC, nPH, nPS;

Index p_key, T_key, X_key;
Index velocity_key, dispersion_key, dispersivity_key;
Index phi_key, phi_old_key, nphi_key, fv_key, rho_key,
rho_TX_key, beta_fluid_key, beta_rock_key, beta_tot_key,
k_key, mu_key, lambda_key, lambda_tot_key, gravity_key,
mass_gravity_key, tx_source_key, chemical_source_key,
vol_key;

Index rho_rock_key, cp_fluid_key, cp_rock_key, cp_tot_key,
K_fluid_key, K_rock_key, K_tot_key, vel_mult_key;

Index abPS_old_key, abPS_key, q_chem_key, q_chem_prev_key,
abSP_old_key, abSP_key;
```

```
        ScalarVariable phi, nphi, fv, rho, rho_TX, beta_fluid,
        beta_rock, beta_tot, k, mu, lambda, lambda_tot, tx_source,
        chemical_source, T, rho_rock, cp_fluid, cp_rock, cp_tot,
        K_fluid, K_rock, K_tot;

        VectorVariable<dim> gravityVector, velocityVector;

        int SteadyStatePressureCalculator();
        int SteadyStatePressureWithGravityCalculator();
        int InitializeTransientPressureWithGravity(bool
        gravity=true);
        int UpdatePressure(double64 time_increment, bool
        gravity=true);

        int SteadyStateTemperatureCalculator();
        int InitializeTransientTemperature(bool implicit=true);
        int UpdateTemperature(double64 time_increment);

        int UpdateFluidProperties();

        int CalculateDispersion();
        int CalculateTotalCompressibility();
        int CalculateMassConductivity();
        int CalculateMassGravityTerm();
        int CalculateDensityTX();
        int CalculateFluidExpansionSource();
        int SetChemicalSourceToZero();
        int CalculateTotalHeatCapacity();
        int CalculateTotalThermalConductivity();
        int CalculateVelocityMultiplier();

        int CreateAndInitializeRTM_Variables(bool implicit=true);
        int UpdateTransport(double64 time_increment);

        int SetQChemToZero();
        int DivideQChemWithTimeIncrement(double time_increment);
        int CopyFromTo(Index from_key, Index to_key);
        int BackUp(Index new_key, Index old_key);

        double CalculateL2Error();
    };
} //end csmp
#endif //RTM_SIMULATOR_H
```

B.7 RTM_Simulator.cpp

```
#include "RTM_Simulator.h"


using namespace std;

namespace csmp
```

```
{
  template<size_t dim>
  RTM_Simulator<dim>::RTM_Simulator(char** argv )
      :geometry_name_(argv[1]), vars_name_(argv[2]),
      config_file_name_(argv[3]),
      system_file_list_name_(argv[4]),
      recipes_file_list_name_(argv[5]),
      eps(1.e-4),
      temperature(25.),
      pressure(1.e5),
      salinity(0.0),
      seawater(temperature, pressure, salinity)
    {
      model = new ANSYS_Model3D(geometry_name_, geometry_name_,
      vars_name_, true, true, true);

      pd_ref = &(model->Database());
      //reference to the models property database.

      InputDataManager<dim>  model_configuration;
      model_configuration.ConfigureFromFile(*model,
      config_file_name_, false,
      true, true, false, true, true, run_settings );

      vtu = new VTU_Interface<dim>(*model);

      //pt_solver = new LUdcmp_Solver();
      pt_solver = new SAMG_Solver();
      pt_solver->SolverSettings()->ExplicitSecondary(true);
      pt_solver->SolverSettings()->Set_iout1(-1);
      pt_solver->SolverSettings()->Set_idmp(-1);
      pt_solver->SolverSettings()->Set_mode_mess(-3);

      //! main variables
      p_key = model->Database ().StorageKey ("fluid_pressure");
      T_key = model->Database ().StorageKey ("temperature");
      X_key = model->Database ().StorageKey ("fluid_salinity");
      phi_key     = model->Database ().StorageKey ("porosity");
      phi_old_key = model->Database ().StorageKey ("porosity_old");
      nphi_key    = model->Database ().StorageKey ("nodal_porosity");
      k_key       = model->Database ().StorageKey ("permeability");

      //! fluid properties calculated from equation of state
      rho_key             = model->Database ().StorageKey ("fluid
______density");
      mu_key              = model->Database ().StorageKey ("fluid
______viscosity");
      beta_fluid_key      = model->Database ().StorageKey ("fluid
______compressibility");
      cp_fluid_key        = model->Database ().StorageKey ("fluid
______heat_capacity");

      //! output main variables
      output_props_.push_back("fluid_pressure");
```

```
    output_props_.push_back("velocity");
    output_props_.push_back("temperature");
    output_props_.push_back("fluid salinity");
    output_props_.push_back("fluid density");

  }

template<>
RTM_Simulator<1U>::RTM_Simulator(char** argv )
    :vars_name_(argv[1]), config_file_name_(argv[2]),
    system_file_list_name_(argv[3]), recipes_file_list_name_(argv[4]),
    eps(1.e-9),
    temperature(25.),
    pressure(1.e5),
    salinity(0.0),
    seawater(temperature, pressure, salinity)
  {
    double64 length(atof(argv[5])), dx; // 9.99, 10, 0.5
    size_t n_elements (atoi(argv[6])); //999, 1000, 50

    if (length == 0.0 || n_elements == 0.0)
      throw csmp::Exception(FATAL_ERROR,
      "RTM_Simulator<1U>::RTM_Simulator()",
          "invalid input parameters: model length or number of
          elements");

    dx = length/(double64)n_elements;

    model = new Model1D("Line", vars_name_, length, n_elements);

    pd_ref = &(model->Database());
    //reference to the models property database.

    InputDataManager<1U> model_configuration;
    model_configuration.ConfigureFromFile(*model, config_file_name_,
    false, true, true, false, true, true, run_settings );

    vtu = new VTU_Interface<1U> (*model);
    pt_solver = new SAMG_Solver();
    //pt_solver = new LUdcmp_Solver();
    pt_solver->SolverSettings()->ExplicitSecondary(true);
    pt_solver->SolverSettings()->Set_iout1(-1);
    pt_solver->SolverSettings()->Set_idmp(-1);
    pt_solver->SolverSettings()->Set_mode_mess(-3);

    //! main variables
    p_key = model->Database ().StorageKey ("fluid pressure");
    T_key = model->Database ().StorageKey ("temperature");
    X_key = model->Database ().StorageKey ("fluid salinity");
    phi_key     = model->Database ().StorageKey ("porosity");
    phi_old_key = model->Database ().StorageKey ("porosity old");
    nphi_key    = model->Database ().StorageKey ("nodal porosity");
    k_key       = model->Database ().StorageKey ("permeability");
```

```
      //! fluid properties calculated from the equation of state
      rho_key             = model->Database().StorageKey("fluid
      density");
      mu_key              = model->Database().StorageKey("fluid
      viscosity");
      beta_fluid_key      = model->Database().StorageKey("fluid
      compressibility");
      cp_fluid_key        = model->Database().StorageKey("fluid heat
      capacity");

      //! output main variables
      output_props_.push_back("fluid pressure");
      output_props_.push_back("velocity");
      output_props_.push_back("temperature");
      output_props_.push_back("fluid salinity");
      output_props_.push_back("fluid density");
    }

  template<size_t dim>
  RTM_Simulator<dim>::~RTM_Simulator()
    {
        //! clean memory here!
        delete model;
        delete vtu;

        delete transport_abPS;
        delete heat_transport;
        delete pt_solver;

        delete transient_pressure;
        delete pt_capacitance_lhs;
        delete pt_conductance;
        delete velocity;
        delete pt_capacitance_rhs;
        delete pt_gravity;
        delete nodal_fluid_src;
        delete fluid_expansion_src;
        delete chemical_src;
    }

  //! Steady State Pressure without gravity, can be used to run 1d
  // (or pseudo 1d) benchmarks
  template<size_t dim>
  int RTM_Simulator<dim>::SteadyStatePressureCalculator()
  {
    //! conductivity visitor
    ConductivityVisitor<dim> conductivity_visitor( *model,
    "conductivity", "permeability", "fluid viscosity" );
    (*model).Accept(conductivity_visitor);

    //! steady state pressure
    p_solver = new SAMG_Solver();
    // supresses SAMG output
    p_solver->SolverSettings()->ExplicitSecondary(true);
```

```
    p_solver->SolverSettings()->Set_iout1(-1);
    p_solver->SolverSettings()->Set_idmp(-1);
    p_solver->SolverSettings()->Set_mode_mess(-3);
    //p_solver = new LUdcmp_Solver();
    PDE_Integrator<dim, Region> steady_state_pressure( p_solver );
    steady_state_pressure.Verbose(false);

    NumIntegral_dNT_op_dN_dV <dim, Element<dim>> p_conductance(
    *pd_ref, "conductivity", "fluid_pressure", "fluid_pressure" );
    PointSource_rhsop <dim, Element<dim>>          nodal_fluid_src(
    *pd_ref, "nodal_fluid_source", "fluid_pressure" );

    VelocityAndVolumeFlux<dim, Element<dim>> velocity( *model,
    "conductivity", "nodal_porosity", "fluid_pressure", false );

    steady_state_pressure.Add( &p_conductance );
    steady_state_pressure.Add( &nodal_fluid_src );
    steady_state_pressure.AddPostProcess( &velocity );

    (*model).Apply (steady_state_pressure);

    delete p_solver;
    return 0;
  }

  //! Steady State Pressure with gravity
  //! used for initial state calculation
  template<size_t dim>
  int RTM_Simulator<dim>::SteadyStatePressureWithGravityCalculator()
  {
    lambda_key          = model->Database ().StorageKey
    ("conductivity");
    lambda_tot_key      = model->Database ().StorageKey ("mass
____conductivity");

    gravity_key         = model->Database().StorageKey ("gravity
____term");
    mass_gravity_key    = model->Database().StorageKey ("mass_gravity
____term");

    CalculateMassConductivity();
    CalculateMassGravityTerm();

    //! steady state pressure
    p_solver = new SAMG_Solver();
    // supresses SAMG output
    p_solver->SolverSettings()->ExplicitSecondary(true);
    p_solver->SolverSettings()->Set_iout1(-1);
    p_solver->SolverSettings()->Set_idmp(-1);
    p_solver->SolverSettings()->Set_mode_mess(-3);
    //p_solver = new LUdcmp_Solver();
    PDE_Integrator<dim, Region> steady_state_pressure( p_solver );
    steady_state_pressure.Verbose(false);
```

```
    NumIntegral_dNT_op_dN_dV <dim, Element<dim>> p_conductance(
    *pd_ref, "mass_conductivity", "fluid_pressure", "fluid_pressure" );
    NumIntegral_dNT_op_dV <dim, Element<dim>> p_gravity( *pd_ref,
    "mass_gravity_term", "fluid_pressure" );
    VelocityAndVolumeFlux<dim, Element<dim>> velocity( *model,
    "conductivity", "nodal_porosity", "fluid_pressure", "fluid
____density", false );

    steady_state_pressure.Add( &p_conductance );
    steady_state_pressure.Add( &p_gravity );
    steady_state_pressure.AddPostProcess( &velocity );
    (*model).Apply (steady_state_pressure);

    delete p_solver;
    return 0;
  }

  // transient pressure with gravity
  template<size_t dim>
  int RTM_Simulator<dim>::InitializeTransientPressureWithGravity(
  bool gravity)
  {
    rho_TX_key           = model->Database ().StorageKey ("fluid
____density_TX");

    beta_rock_key        = model->Database ().StorageKey ("rock
____compressibility");
    beta_tot_key         = model->Database ().StorageKey ("total
____compressibility");
    tx_source_key        = model->Database ().StorageKey ("fluid
____expansion_source");
    chemical_source_key  = model->Database ().StorageKey ("chemical
____source");
    vol_key              = model->Database ().StorageKey ("solution
____volume");
    fv_key               = model->Database ().StorageKey ("finite
____volume");

    transient_pressure = new PDE_Integrator<dim, Region>( pt_solver );
    transient_pressure->Verbose(false);

    pt_capacitance_lhs = new NumIntegral_NT_lhsop_N_dV<dim,Element<dim>>(
    *pd_ref, "total_compressibility", "fluid_pressure", "fluid_pressure" );

    pt_capacitance_rhs = new NumIntegral_NT_op_N_dV<dim,Element<dim>>(
    *pd_ref, "total_compressibility", "fluid_pressure" );

    pt_conductance = new NumIntegral_dNT_op_dN_dV <dim, Element<dim>>(
    *pd_ref, "mass_conductivity", "fluid_pressure", "fluid_pressure" );
    pt_conductance->MultiplyWithTimeIncrement(true);

    if (gravity)
    {
      pt_gravity = new NumIntegral_dNT_op_dV <dim, Element<dim>>( *pd_ref,
```

```
      "mass_gravity_term", "fluid_pressure" );
      pt_gravity->AddAccumulateLater();
      pt_gravity->MultiplyWithTimeIncrement(true);
    }

    nodal_fluid_src = new PointSource_rhsop <dim, Element<dim>>( *pd_ref,
    "nodal_fluid_source", "fluid_pressure" );
    nodal_fluid_src->MultiplyWithTimeIncrement(true);
    nodal_fluid_src->LumpedFormulation(true);
    nodal_fluid_src->AddAccumulateLater();

    //! important: this source is not multiplied by dt
    fluid_expansion_src = new PointSource_rhsop <dim, Element<dim>>(
    *pd_ref, "fluid_expansion_source", "fluid_pressure" );
    fluid_expansion_src->LumpedFormulation(true);
    fluid_expansion_src->AddAccumulateLater();

    //! important: this source is not multiplied by dt
    chemical_src = new PointSource_rhsop <dim, Element<dim>>( *pd_ref,
    "chemical_source", "fluid_pressure" );
    chemical_src->LumpedFormulation(true);
    chemical_src->AddAccumulateLater();

    if (gravity)
        velocity = new VelocityAndVolumeFlux<dim, Element<dim>>( *model,
        "conductivity", "nodal_porosity", "fluid_pressure", "fluid
_________density", false );
    else
        velocity = new VelocityAndVolumeFlux<dim, Element<dim>>( *model,
        "conductivity", "nodal_porosity", "fluid_pressure", false );

    transient_pressure->Add( pt_conductance );
    transient_pressure->Add( pt_capacitance_lhs );
    transient_pressure->Add( pt_capacitance_rhs );
    if (gravity)
        transient_pressure->Add( pt_gravity );
    transient_pressure->Add( nodal_fluid_src );
    transient_pressure->Add( fluid_expansion_src );
    transient_pressure->Add( chemical_src );
    transient_pressure->AddPostProcess( velocity );

    //! for the initial calculation of fluid expansion source
    BackUp(rho_key, rho_TX_key);

    return 0;
  }

  template<size_t dim>
  int RTM_Simulator<dim>::UpdatePressure(double64 time_increment,
  bool gravity)
  {
    CalculateTotalCompressibility();
    CalculateMassConductivity();
    if (gravity)
```

```
        CalculateMassGravityTerm();

    CalculateFluidExpansionSource();

    transient_pressure->TimeIncrement(time_increment);
    (*model).Apply (*transient_pressure);

    SetChemicalSourceToZero();

    //! unless set up in the config file
    //CalculateDispersion();
    return 0;
  }

  template<size_t dim>
  int RTM_Simulator<dim>::SteadyStateTemperatureCalculator()
  {
    t_solver = new SAMG_Solver();
    //t_solver = new LUdcmp_Solver();

    K_fluid_key  = model->Database().StorageKey("fluid_thermal
____conductivity");
    K_rock_key   = model->Database().StorageKey("rock_thermal
____conductivity");
    K_tot_key    = model->Database().StorageKey("total_thermal
____conductivity");
    CalculateTotalThermalConductivity();

    PDE_Integrator<dim, Region> steady_state_temperature( t_solver );
    NumIntegral_dNT_op_dN_dV<dim, Element<dim> >   t_conductance(
    *pd_ref, "total_thermal_conductivity", "temperature",
    "temperature" );
    NumIntegral_SetRHS_to_Zero<dim> t_rhs(*pd_ref, "temperature");

    steady_state_temperature.Add(&t_conductance);
    steady_state_temperature.Add(&t_rhs);

    (*model).Apply (steady_state_temperature);

    delete t_solver;
    return 0;
  }

  template<size_t dim>
  int RTM_Simulator<dim>::InitializeTransientTemperature(bool implicit)
  {
    rho_rock_key = model->Database().StorageKey("rock_density");
    cp_rock_key  = model->Database().StorageKey("rock_heat_capacity");
    cp_tot_key   = model->Database().StorageKey("total_heat_capacity");

    K_fluid_key  = model->Database().StorageKey("fluid_thermal
____conductivity");
    K_rock_key   = model->Database().StorageKey("rock_thermal
____conductivity");
```

```
    K_tot_key     = model->Database().StorageKey("total_thermal
____conductivity");

    velocity_key = model->Database().StorageKey("velocity");
    vel_mult_key = model->Database().StorageKey("velocity_multiplier");

    if (!implicit)
    {
      //! for explicit transport
      heat_transport = new ExplicitNodeCenteredFiniteVolumeTransport<dim,
      ExplicitStencilProcessor>("Model", *model, "total_heat_capacity",
      "total_thermal_conductivity", "temperature", "heat_transfer_velocity",
      "nodal_heat_source", false);
    }
    else
    {
      heat_transport = new NodeCenteredFiniteVolumeTransport<dim>("Model",
      *model, "total_heat_capacity", "total_thermal_conductivity",
      "temperature", "velocity", "nodal_heat_source", false, false, NULL,
      "velocity_multiplier");
      heat_transport->GetSolverSettings().SetSolverInstance(1);
      // supresses SAMG output
      heat_transport->GetSolverSettings().ExplicitSecondary(true);
      heat_transport->GetSolverSettings().Set_iout1(-1);
      heat_transport->GetSolverSettings().Set_idmp(-1);
      heat_transport->GetSolverSettings().Set_mode_mess(-3);
    }

    // supresses csmp transport output
    heat_transport->Verbose(false);

    // if p,T are tested without transport, uncomment this
    //heat_transport->FiniteVolume("finite volume");

    return 0;
  }

  template<size_t dim>
  int RTM_Simulator<dim>::UpdateTemperature(double64 time_increment)
  {
    CalculateTotalHeatCapacity();
    CalculateTotalThermalConductivity();
    CalculateVelocityMultiplier();

    bool update_pore_volumes=true;
    heat_transport->AdvectVariable(time_increment, 0.9, true,
    update_pore_volumes);

    return 0;
  }

  template<size_t dim>
  int RTM_Simulator<dim>::UpdateFluidProperties()
  {
```

```
  const typename vector<Node< dim>*>::const_iterator modelNodesEnd(
  model->Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator it(
  model->Region("Model").NodesBegin() );
       it != modelNodesEnd; ++it )
  {
      temperature     = (*it)->Read(T_key);
      pressure        = (*it)->Read(p_key);
      salinity        = (*it)->Read(X_key);

      rho()           = seawater.Density();
      mu()            = seawater.Viscosity();
      beta_fluid()    = seawater.Compressibility();
      cp_fluid()      = seawater.HeatCapacity();

      (*it)->Store(rho_key, rho);
      (*it)->Store(mu_key, mu);
      (*it)->Store(beta_fluid_key, beta_fluid);
      (*it)->Store(cp_fluid_key, cp_fluid);

  }

  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateDispersion()
{
  //! caclulate dispersion from dispersivity
  Index velocity_key          = model->Database().StorageKey ("velocity");
  Index porosity_key          = model->Database().StorageKey ("porosity");
  Index dispersion_key        = model->Database().StorageKey ("dispersion");
  Index dispersivity_key      = model->Database().StorageKey
  ("dispersivity");
  VectorVariable<dim> velocity_vector;
  ScalarVariable porosity,dispersion, dispersivity;
  const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
  model->Region("Model").ElementsEnd() );
  for(typename vector<Element<dim>*>::const_iterator it(
  model->Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
    {
      (*it)->Read(velocity_key, velocity_vector);
      (*it)->Read(porosity_key, porosity);
      (*it)->Read(dispersivity_key, dispersivity);
      dispersion = dispersivity()*velocity_vector[0]*porosity();
      (*it)->Store(dispersion_key, dispersion);
    }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateTotalCompressibility()
{
```

```
  //! beta_tot()      = rho * phi * (beta_fluid(p, T) + beta_rock);
  const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
  model->Region("Model").ElementsEnd() );

  for( typename vector<Element<dim>*>::const_iterator it(
  model->Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
  {
    beta_rock()   = (*it)->Read(beta_rock_key);
    for (size_t ip=0;ip<(*it)->IntegrationPoints (); ++ip)
    {
      (*it)->PropertyValueAtIntegrationPoint( nphi_key, ip, nphi );
      (*it)->PropertyValueAtIntegrationPoint( rho_key, ip, rho );
      (*it)->PropertyValueAtIntegrationPoint( beta_fluid_key, ip,
      beta_fluid );
      beta_tot() = rho() * nphi() * ( beta_fluid() + beta_rock() );
      (*it)->Store( ip, beta_tot_key, beta_tot );
    }
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateMassConductivity()
{
  //! lambda          = k / mu;
  //! lambda_tot      = rho * k / mu;
  const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
  model->Region("Model").ElementsEnd() );
  for( typename vector<Element<dim>*>::const_iterator it(
  model->Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
  {
    k() = (*it)->Read(k_key);
    for (size_t ip=0;ip<(*it)->IntegrationPoints (); ++ip)
    {
      (*it)->PropertyValueAtIntegrationPoint( rho_key, ip, rho );
      (*it)->PropertyValueAtIntegrationPoint( mu_key, ip, mu );
      lambda() = k()/mu();
      lambda_tot() = rho()*k()/mu();
      (*it)->Store( ip, lambda_key, lambda );
      (*it)->Store( ip, lambda_tot_key, lambda_tot );
    }
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateMassGravityTerm()
{
  //! gravity term visitor
  ComputeGravityTermVisitor<dim> gravity_visitor( *model, "gravity_term",
  "permeability", "fluid_viscosity", "fluid_density" );
  (*model).Accept (gravity_visitor);
```

```
  const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
  model->Region("Model").ElementsEnd() );
  for( typename vector<Element<dim>*>::const_iterator it(
  model->Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
  {
    for (size_t ip=0;ip<(*it)->IntegrationPoints (); ++ip)
    {
      (*it)->Read(ip, gravity_key, gravityVector);
      (*it)->PropertyValueAtIntegrationPoint( rho_key, ip, rho );
      (*it)->Store( ip, mass_gravity_key, gravityVector*rho() );
    }
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateDensityTX()
{
  //! update density (p_new, T_old, X_old)
  const typename vector<Node< dim>*>::const_iterator modelNodesEnd(
  model->Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator it(
  model->Region("Model").NodesBegin() );
       it != modelNodesEnd; ++it )
  {
      temperature     = (*it)->Read(T_key);
      pressure        = (*it)->Read(p_key);
      salinity        = (*it)->Read(X_key);
      rho_TX()        = seawater.Density();
      (*it)->Store(rho_TX_key, rho_TX);
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateFluidExpansionSource()
{
  //! q_tx = - phi*SectorVolume*(rho(p_new,T_new,X_new) -
  rho_TX(p_new,T_old,X_old))
  const typename vector<Node< dim>*>::const_iterator modelNodesEnd(
  model->Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator it(
  model->Region("Model").NodesBegin() );
       it != modelNodesEnd; ++it )
  {
      (*it)->Read(nphi_key, nphi);
      (*it)->Read(fv_key, fv);
      (*it)->Read(rho_key, rho);
      (*it)->Read(rho_TX_key, rho_TX);

      tx_source() = -nphi()*(rho() - rho_TX())*fv();
```

```
        (*it)->Store(tx_source_key, tx_source);
    }

    return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::SetChemicalSourceToZero()
{
    //! q_chemical = 0.
    const typename vector<Node< dim>*>::const_iterator modelNodesEnd(
    model->Region("Model").NodesEnd() );
    for( typename vector<Node<dim>*>::const_iterator it(
    model->Region("Model").NodesBegin() );
         it != modelNodesEnd; ++it )
    {
        chemical_source() = 0.;
        (*it)->Store(chemical_source_key, chemical_source);
    }

    return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateTotalHeatCapacity()
{
    //! cp_tot = phi*rho*cp_fluid + (1-phi)*rho_rock*cp_rock
    const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
    model->Region("Model").ElementsEnd() );
    for( typename vector<Element<dim>*>::const_iterator it(
    model->Region("Model").ElementsBegin() );
         it != modelElementsEnd; ++it )
    {
      rho_rock()     = (*it)->Read(rho_rock_key);
      cp_rock()      = (*it)->Read(cp_rock_key);
      for (size_t iSector=0;iSector<(*it)->FV_Stencil()->Sectors();
      iSector++)
      {
        nphi()       = (*it)->PropertyValueAtSectorIntegrationPoint(
        iSector, 0U, nphi_key );
        rho()        = (*it)->PropertyValueAtSectorIntegrationPoint(
        iSector, 0U, rho_key );
        cp_fluid()   = (*it)->PropertyValueAtSectorIntegrationPoint(
        iSector, 0U, cp_fluid_key );
        cp_tot()     = (1-nphi())*rho_rock()*cp_rock() +
        nphi()*rho()*cp_fluid();
        (*it)->Store( iSector, 0U, cp_tot_key, cp_tot );
      }
    }
    return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateTotalThermalConductivity()
```

```
{
  //! K = phi*K_f + (1-phi)*K_r
  const typename vector<Element< dim>*>::const_iterator modelElementsEnd(
  model->Region("Model").ElementsEnd() );
  for( typename vector<Element<dim>*>::const_iterator it(
  model->Region("Model").ElementsBegin() );
       it != modelElementsEnd; ++it )
  {
    // this is a simple version, as K_fluid is constant and read
    //from the config input file
    (*it)->PropertyValueAtBaryCenter( nphi_key, nphi );
    K_rock()  = (*it)->Read(K_rock_key);
    (*it)->PropertyValueAtBaryCenter(K_fluid_key, K_fluid);
    K_tot()   = nphi()*K_fluid() + (1-nphi())*K_rock();
    (*it)->Store(K_tot_key, K_tot);
    // the more sophisticated version can be used:
    //for (size_t ip=0;ip<(*it)->IntegrationPoints (); ++ip)
    //{
    //  (*it)->PropertyValueAtIntegrationPoint( K_fluid_key, ip,
    //K_fluid );
    //  (*it)->Store( ip, K_tot_key, K_tot + phi()*K_fluid());
    //}
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CalculateVelocityMultiplier()
{
  //! vel_mult = rho*cp_fluid
  const typename vector<Node< dim>*>::const_iterator modelNodesEnd(
  model->Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator it(
  model->Region("Model").NodesBegin() );
       it != modelNodesEnd; ++it )
  {
    (*it)->Read(rho_key, rho);
    (*it)->Read(cp_fluid_key, cp_fluid);
    (*it)->Store(vel_mult_key, rho*cp_fluid());
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CreateAndInitializeRTM_Variables(bool implicit)
{
  model->CreateProperty ("abPS", "mol/l", ARRAY, NODE, nIC-1);
  // the last IC is charge
  model->CreateProperty ("q_chem", "mol/l", ARRAY, NODE, nIC-1);
  model->CreateProperty ("abSP", "mol/l", ARRAY, NODE, nIC-1);
  //! this property is used for output only (mineral amounts)
  model->CreateProperty ("axPH", "mol/l", ARRAY, NODE, nPH-nPS);
  // first nPS components are phases-solutions
  model->CreateProperty ("avPH", "X", ARRAY, NODE, nPH-nPS);
```

```
    //volumes of solid phases
    model->CreateProperty ("amPH", "kg/l", ARRAY, NODE, nPH-nPS);
    //masses of solid phases

    abPS_key          = model->Database ().StorageKey ("abPS");
    q_chem_key        = model->Database ().StorageKey ("q_chem");
    abSP_key          = model->Database ().StorageKey ("abSP");

    if (!implicit)
    {
      //! for explicit transport
      transport_abPS = new ExplicitNodeCenteredFiniteVolumeTransport<dim,
      ExplicitStencilProcessor> ("Model", *model, "nodal_porosity",
      "dispersion", "abPS", "velocity", "q_chem", false);
    }
    else
    {
      transport_abPS = new NodeCenteredFiniteVolumeTransport<dim>
      ("Model", *model, "nodal_porosity", "dispersion",
      "abPS", "velocity", "q_chem", false, false);

      transport_abPS->GetSolverSettings().SetSolverInstance(2);
      // supresses SAMG output
      transport_abPS->GetSolverSettings().ExplicitSecondary(true);
      transport_abPS->GetSolverSettings().Set_iout1(-1);
      transport_abPS->GetSolverSettings().Set_idmp(-1);
      transport_abPS->GetSolverSettings().Set_mode_mess(-3);
      //transport_abPS->GetSolverSettings().Set_nxtyp(0);
    }

    // supresses csmp transport output
    transport_abPS->Verbose(false);

    // calculate the volumes of finite volumes
    transport_abPS->FiniteVolume("finite_volume");

    output_props_.push_back("abPS");
    output_props_.push_back("axPH");
    output_props_.push_back("avPH");
    output_props_.push_back("amPH");

    return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::SetQChemToZero()
{
    //! mineral source = 0;
    ArrayVariable q_chem_node(nIC-1, 0.);
    const typename vector<Node<dim>*>::const_iterator modelNodesEnd(
    model->Region("Model").NodesEnd() );
    for( typename vector<Node<dim>*>::const_iterator n(
    model->Region("Model").NodesBegin() ); n != modelNodesEnd; ++n )
    {
```

```
      (*n)->Store(q_chem_key, q_chem_node);
    }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::CopyFromTo(Index from_key, Index to_key)
{
  ArrayVariable temp(nIC-1, 0.);
  const typename vector<Node<dim>*>::const_iterator modelNodesEnd(
  model->Region("Model").NodesEnd() );
  for( typename vector<Node<dim>*>::const_iterator n(
  model->Region("Model").NodesBegin() ); n != modelNodesEnd; ++n )
  {
      (*n)->Read(from_key, temp);
      (*n)->Store(to_key, temp);
  }
  return 0;
}

template<size_t dim>
int RTM_Simulator<dim>::BackUp(Index new_key, Index old_key)
{
  ScalarVariable temp;
  if (new_key.place == NODE)
  {
    const typename vector<Node<dim>*>::const_iterator modelNodesEnd(
    model->Region("Model").NodesEnd() );
    for( typename vector<Node<dim>*>::const_iterator n(
    model->Region("Model").NodesBegin() ); n != modelNodesEnd; ++n )
    {
        (*n)->Read(new_key, temp);
        (*n)->Store(old_key, temp);
    }
  }
  else if (new_key.place == ELEMENT)
  {
    const typename vector<Element<dim>*>::const_iterator
    modelElementsEnd( model->Region("Model").ElementsEnd() );
    for( typename vector<Element<dim>*>::const_iterator n(
    model->Region("Model").ElementsBegin() );
    n != modelElementsEnd; ++n )
    {
        (*n)->Read(new_key, temp);
        (*n)->Store(old_key, temp);
    }
  }

  return 0;
}

//! main function. SNIA for transport and chemistry
template<size_t dim>
int RTM_Simulator<dim>::run()
```

```
{
  //! create an instance of gems3k visitor
  GEMS3K_Visitor<dim> gems3k_visitor(*model,
  this->system_file_list_name_,
  this->recipes_file_list_name_);
  nIC = gems3k_visitor.GetnIC();
  nPH = gems3k_visitor.GetnPH();
  nPS = gems3k_visitor.GetnPS();
  //! create and initialize variables for chemical species
  bool implicit_transport = true; //default=true
  CreateAndInitializeRTM_Variables(implicit_transport);

  //! porosity interpolated to the elements
  model->InterpolateNodeToElementProperty("nodal_porosity", "porosity");

  //! Kozeny-Carman visitor fro porosity/permeability feedback
  KozenyCarmanVisitor<dim> KozenyCarman_visitor(*model, phi_key,
  phi_old_key, k_key );

  output_props_.push_back("recipe_index");

  vtu->OutputDataToVTU( ( string(config_file_name_) +
  "_Default_Properties" ).c_str(), output_props_,
  model->Region("Model"), 0);

  bool transient_temperature = false;
  bool constant_temperature = true;

  bool transient_pressure = true; //true
  bool pressure_with_gravity = true; //true

  //! initialization
  //! get fluid properties from eos
  UpdateFluidProperties();
  //! calculate steady state temperature if necessary
  if (!constant_temperature && !transient_temperature )
      SteadyStateTemperatureCalculator();
  //! calculate steady state pressure using those fluid properties
  if (pressure_with_gravity)
      SteadyStatePressureWithGravityCalculator();
  else
      SteadyStatePressureCalculator();

  //! calculate initial equilibrium state at initial
  // pressure/temperature conditions
  gems3k_visitor.SetInitialProperties();

  //! and recalculate fluid properties
  UpdateFluidProperties();
  //! and pressure
  if (pressure_with_gravity)
      SteadyStatePressureWithGravityCalculator();
  else
      SteadyStatePressureCalculator();
```

```
//! and chemical equilibrium
gems3k_visitor.SetInitialProperties();

//! debug output, comment out if needed
output_props_.push_back("fluid expansion source");
output_props_.push_back("chemical source");
output_props_.push_back("solution volume");
output_props_.push_back("nodal porosity");
output_props_.push_back("porosity");
output_props_.push_back("permeability");
output_props_.push_back("inert volume");
output_props_.push_back("fluid viscosity");
output_props_.push_back("nodal fluid source");
output_props_.push_back("cfl");
output_props_.push_back("pH");
output_props_.push_back("pe");
output_props_.push_back("Eh");

//! initial output
vtu->OutputDataToVTU( ( string(config_file_name_) +
"_Initial_Properties" ).c_str(), output_props_,
model->Region("Model"), 0);

//! initialize transient pressure/temperature calculators
if (transient_pressure)
    InitializeTransientPressureWithGravity(pressure_with_gravity);
if (transient_temperature)
    InitializeTransientTemperature(implicit_transport);

//! q_chem is the mineral dissolution/precipitation source,
// in SNIA is always equal to zero
SetQChemToZero();

time_increment = run_settings.NearestOutputTime(0.0);
total_time = run_settings.Duration();

cout<<"\n Running RTM simulation for a total_time = "<<total_time<<
" with a time_increment = "<<time_increment<<endl;

double64 dt, time;
global_time     = 0.;
timestep        = 1;

while ( global_time < total_time )
{
  if (transient_pressure)
  {
    //1. update pressure
    UpdatePressure(time_increment, pressure_with_gravity);
    //2. calculate rho(p_new, T_old, X_old) for the fluid
    //expansion source
    CalculateDensityTX();
  }
  else
```

```
    {
      if (pressure_with_gravity)
        SteadyStatePressureWithGravityCalculator();
      else
          SteadyStatePressureCalculator();
    }

    // 3. update temperature, if necessary
    if (transient_temperature)
      UpdateTemperature(time_increment);

    // 4. transport+chemistry are calculated with smaller dt=cfl
    dt = 0.9*transport_abPS->AnisotropicCourantIncrement();
    cout<<"\n CFL transport= "<<dt;

    time = 0.;
    while(time < time_increment)
    {
        if (time_increment - time < dt)
          dt = time_increment - time;

        // 4.1 calculate transport of chemical species
        transport_abPS->AdvectVariableSingleStep(dt, true, true);

        // 4.2 calculate chemical equilibrium with GEMS
        gems3k_visitor.SetTime(dt, time, timestep);
        model->Accept(gems3k_visitor);

        time += dt;

    }

    // 5. porosity interpolated to the elements
    //(only used for the output)
    BackUp(phi_key, phi_old_key);
    //back up the porosity for Kozeny-Carman Visitor
    model->InterpolateNodeToElementProperty("nodal_porosity",
    "porosity");

    // 6. permeability feedback
    model->Accept(KozenyCarman_visitor);

    //if (transient_pressure || transient_temperature)
    // 7. update fluid properties from equation of state
    UpdateFluidProperties();

    // 8. output
    vtu->OutputDataToVTU( ( string(config_file_name_) +
        "_Properties" ).c_str(), output_props_, model->Region("Model"),
        timestep);
    timestep++;
    global_time += time_increment;

  }
```

```
    return 0;
  }

  template class RTM_Simulator<1U>;
  template class RTM_Simulator<3U>;
}// end csmp
```

References

[1] Renewable energy: Share of renewable energy up to 13% of energy consumption in the EU27 in 2011, Eurostat News Releases, 65/2013 - 26 April 2013. http://ec.europa.eu/eurostat/

[2] Dickinson, J.S., Bulik, N., Matthews, M.C., Snijders, A, Aquifer thermal energy storage: theoretical and operational analysis, Geotechnique, 59, No.3, Pages 249-260

[3] H.O. Paksoy, Z. Grbz, B. Turgut, D. Dikici, H. Evliya, Aquifer thermal storage (ATES) for air-conditioning of a supermarket in Turkey, Renewable Energy, Volume 29, Issue 12, October 2004, Pages 1991-1996

[4] H.O Paksoy, O Andersson, S Abaci, H Evliya, B Turgut, Heating and cooling of a hospital using solar energy coupled with seasonal thermal energy storage in an aquifer, Renewable Energy, Volume 19, Issues 12, JanuaryFebruary 2000, Pages 117-122

[5] Stefan Kranz, Stephanie Frick, Efficient cooling energy supply with aquifer thermal energy storages, Applied Energy, Volume 109, September 2013, Pages 321-327

[6] Thomas E. Dwyer, Yoram Eckstein, Finite-element simulation of low-temperature, heat-pump-coupled, aquifer thermal energy storage, Journal of Hydrology, Volume 95, Issues 12, 15 November 1987, Pages 19-38

[7] B. Goyeau, J. Gounot, P. Fabrie, Numerical Modeling of Hot Water Storage in Aquifer by Finite Element Method, In: M.A. Celia, L.A. Ferrand, C.A. Brebbia, W.G. Gray and G.F. Pinder, Editor(s), Developments in Water Science, Elsevier, 1988, Volume 36, Pages 325-330

[8] C. Forkel, H. Daniels, Finite element simulation of circulation in large scale thermal energy storage basins, Advances in Water Resources, Volume 18, Issue 3, 1995, Pages 147-158

[9] Nathalie Courtois, Ariane Grisey, Dominique Grasselly, André Menjoz, Yves Noël, Vincent Petit, Dominique Thiéry, Application of aquifer thermal energy storage for heating and cooling of greenhouses in France: a pre-feasibility study, Proceedings European Geothermal Congress 2007, Unterhaching, Germany, 30 May-1 June 2007

[10] Alexander Vandenbohede, Thomas Hermans, Frdric Nguyen, Luc Lebbe, Shallow heat injection and storage experiment: Heat transport simulation and sensitivity analysis, Journal of Hydrology, Volume 409, Issues 12, 28 October 2011, Pages 262-272

[11] Jongchan Kim, Youngmin Lee, Woon Sang Yoon, Jae Soo Jeon, Min-Ho Koo, Youngseuk Keehm, Numerical modeling of aquifer thermal energy storage system, Energy, Volume 35, Issue 12, December 2010, Pages 4955-4965.

[12] Rui Fan, Yiqiang Jiang, Yang Yao, Deng Shiming, Zuiliang Ma, A study on the performance of a geothermal heat exchanger under coupled heat conduction and groundwater advection, Energy, Volume 32, Issue 11, November 2007, Pages 2199-2209

[13] Dim Coumou, Thomas Driesner, Sebastian Geiger, Christoph A. Heinrich, Stephan Matthäi, The dynamics of mid-ocean ridge hydrothermal systems: Splitting plumes and fluctuating vent temperatures, Earth and Planetary Science Letters, Volume 245, Issues 12, 15 May 2006, Pages 218-231

[14] Geiger , S., Driesner, T., Matthäi, S.K., and Heinrich, C. A. (2006), Multiphase Thermohaline Convection in the Earth's Crust: I. A Novel Finite element - Finite Volume Solution Technique Combined with a New Equation of State for NaCl-H2O. Transport in Porous Media 63(3), 399-434.

[15] Matthäi, S. K., Geiger, S. and Roberts, S.: 2001, Complex Systems Platform: Users Guide.

[16] SAMG User's Manual, Release 25a1, December 2010, Fraunhofer Institute SCAI, St. Augustin, Germany.

[17] Coumou, D. (2008), Numerical simulation of fluid flow in mid-ocean ridge hydrothermal systems, Ph.D. thesis, ETH Zurich, Zurich, Switzerland.

[18] Pruess, K. TOUGH2 - A General Purpose Numerical Simulator for Multiphase Fluid and Heat Flow, Lawrence Berkeley Laboratory Report LBL-29400, Berkeley, CA, 1991a.

[19] Haar, L., J. S. Gallagher, and G. S. Kell (1984), NBS/NRC Steam Tables Thermodynamic and Transport Properties and Computer Programs for Vapor and Liquid States of Water in SI Units, Hemisphere, New York.

[20] M.A. Rosen, Second-law analysis of aquifer thermal energy storage systems, Energy, Volume 24, Issue 2, February 1999, Pages 167-182

[21] Dincer, M. A., Rosen, I., Thermal Energy Storage Systems and Applications, 2nd Ed., Wiley, 2011

[22] Digitale Obersterreichische Raum-Informations-System http://doris.ooe.gv.at/fachinfo/wasser/

[23] Sanner, B., Shallow Geothermal Energy, Geo-Heat Center Quarterly Bulletin, Vol. 22, No. 2, June 2001, Pages 19-25.

[24] P. Engesgaard, K., Kipp, A geochemical transport model for redox-controlled movement of mineral fronts in groundwater flow systems: A case of nitrate removal by oxidation of pyrite, Water Resources Research, 28, pp. 2829-2843, 1992.

[25] Lichtner, P.: Continuum model for simultaneous chemical reactions and mass transport in hydrothermal systems. Geochimica et Cosmochimica Acta 49, 779-800 (1985)

[26] Reed, M.: Calculation of multicomponent chemical equilibria and reaction processes in systems involving minerals, gases and an aqueous phase. Geochimica et Cosmochimica Acta 46, 513-528 (1982)

[27] Helgeson H. C. (1968) Evaluation of irreversible reactions in geochemical processes involving minerals and aqueous solutions. I. Thermodynamic relations. Geochim. Cosmochim. Acta 32, 853-877.

[28] Steefel, C.I., and Lasaga, A.C.: A coupled model for transport of multiple chemical species and kinetic precipitation/dissolution reactions with application to reactive flow in single phase hydrothermal systems. American Journal of Science 294, 529-592 (1994)

[29] Steefel, C.I. and MacQuarrie, K.T.B.: Approaches to modeling of reactive transport in porous media. Reviews in Mineralogy and Geochemistry 34, 85-129 (1996)

[30] Steefel, C.I., Appelo, C.A.J., Arora, B., Jacques, D., Kalbacher, T., Kolditz, O., Lagneau, V., Lichtner, P.C., Mayer, K.U., Meeussen, J.C.L., Molins, S., Moulton, D., Shao, H., Simunek, J., Spycher, N., Yabusaki, S.B., Yeh, G.T. Reactive transport codes for subsurface environmental simulation(2015). Computational Geosciencies 19(3), 445478.

[31] C. de Dieuleveult, J. Erhel, M. Kern, A global strategy for solving reactive transport equations, Journal of Computational Physics, 228, pp. 6395-6410, 2009.

[32] User Documentation for KINSOL v2.7.0.

http://computation.llnl.gov/casc/sundials/documentation/documentation.html

[33] Bethke, C.: Geochemical and biogeochemical reaction modeling. Cambridge : Cambridge University Press, (2008)

[34] Putnis, A.: Mineral replacement reactions: from macroscopic observations to microscopic mechanisms. Mineralogical Magazine 66(5), 689-708 (2002)

[35] Gregg, J. M., 2004, Basin fluid flow, base metal mineralization, and the development of dolomite petroleum reservoirs, in Rizzi, G., Darke, G, and Braithwaite, C. eds., The Geometry and Petrogenesis of Dolomite Hydrocarbon Reservoirs: London, Geological Society Special Publication 235, p. 157-175.

[36] Jones, G., Xiao, Y. (2005) Dolomitization, anhydrite cementation, and porosity evolution in a reflux system: Insights from reactive transport models. AAPG Bulletin 89(5), 577 601.

[37] Al-Helal, A., Whitaker, F., Xiao, Y. (2012) Reactive transport modeling of brine reflux: dolomitization, anhydrite precipitation, and porosity evolution. Journal of Sedimentary Research 82, 196-215.

[38] Gabellone, T., Whitaker, F. (2015), Secular variations in seawater chemistry controlling dolomitisation in shallow reflux systems: insights from reactive transport modelling. Sedimentology (in revision).

[39] Lu, P., Cantrell, D.: Reactive transport modelling of reflux dolomitization in the Arab-D reservoir, Ghawar field, Saudi Arabia, Sedimentology (accepted) (2015)

[40] Wilson, A., Sanford, W., Whitaker, F., Smart, P. (2001) Spatial patterns of diagenesis during geothermal circulation in carbonate platforms. American Journal of Science, 301, 727-752.

[41] Whitaker, F., Xiao, Y.: Reactive transport modeling of early burial dolomitization of carbonate platforms by geothermal convection. AAPG Bull. 94(6), 889–917 (2010)

[42] Consonni, A., Ronchi, P., Geloni, C., Battistelli, A., Grigo, D., Biagi, S., Gherardi, F., Gianelli, G.: Application of numerical modelling to a case of compaction-driven dolomitization: a Jurassic palaeohigh in the Po Plain, Italy. Sedimentology 57(1), 209-231 (2010)

[43] Corbella, M., Gomez-Rivas, E., Martn-Martn, J.D., Stafford, S.L., Teixell, A., Griera, A., Trav, A., Cardellach, E., Salas, R.: Insights to controls on dolomitization by means of reactive transport models applied to the Benicssim case study (Maestrat Basin, eastern Spain). Petroleum Geoscience 20(1), 41-54 (2014)

[44] Machel, H. G.: Concepts and models of dolomitization, a critical reappraisal. In Braithwaite, C.J.R., Rizzi, G. and Darke, G. (eds.) The geometry and petrogenesis of dolomite hydrocarbon reservoirs, p. 7-63. Geological Society (London) Special Publication 235 (2004)

[45] Arvidson, R.S., Mackenzie, F.T.: The dolomite problem; control of precipitation kinetics by temperature and saturation state. American Journal of Science 299 (4), 257-288 (1999)

[46] Palandri, J. and Kharaka, Y.: A compilation of rate parameters of water-mineral interaction kinetics for application to geochemical modelling. U.S.G.S. Open File Report 2004-1068, Menlo Park CA (2004)

[47] Davies G.R. and Smith L.B.J. (2006) Structurally controlled hydrotermal dolomite reservoir facies: An overview. AAPG Bulletin, 90 (11), 1641-1690.

[48] Karpov, I.K., Chudnenko, K.V., Kulik, D.A.: Modeling chemical mass-transfer in geochemical processes: Thermodynamic relations, conditions of equilibria and numerical algorithms. American Journal of Science 297, 767-806 (1997)

[49] Kulik, D.A., Wagner, T., Dmytrieva, S.V., Kosakowski, G., Hingerl, F.F., Chudnenko, K.V., Berner, U.: GEM-Selektor geochemical modeling package: revised algorithm and GEMS3K numerical kernel for coupled simulation codes. Computational Geosciences 17, 1-24 (2013)

[50] Wagner, T., Kulik, D.A., Hingerl, F.F., Dmytrieva, S.V.: GEM-Selektor geochemical modeling package: TSolMod library and data interface for multicomponent phase models. The Canadian Mineralogist 50, 1173-1195 (2012)

[51] Shao, H., Dmytrieva, S.V., Kolditz, O., Kulik, D.A., Pfingsten, W., Kosakowski, G. (2009), Modeling reactive transport in non-ideal aqueoussolid solution system. Applied Geochemistry 24(7), 1287-1300.

[52] Fowler, S.J., Kosakowski, G, Driesner, T., Kulik,D.A., Wagner,T., Wilhelm, S., Masset, O.: Numerical simulation of reactive fluid flow on unstructured meshes. Transport in Porous Media (in revision) (2015)

[53] Adrian Bejan, Allan D. Kraus, Heat transfer handbook. John Wiley & Sons, 2003.

[54] Driesner, T. (2007), The system H2O-NaCl. Part II: Correlations for molar volume, enthalpy, and isobaric heat capacity from 0 to 1000oC, 1 to 5000 bar, and 0 to 1 XNaCl. Geochimica et Cosmochimica Acta 71, 4902-4919.

[55] Karpov, I.K., Chudnenko, K.V., Kulik, D.A., Avchenko, O.V., Bychinskii, V.A.: Minimization of Gibbs free energy in geochemical systems by convex programming. Geochemistry International 39, 1108-1119 (2001)

[56] Thien, B.M.J., Kulik, D.A., Curti, E.: A unified approach to model uptake kinetics of trace elements in complex aqueous-solid solution systems. Applied Geochemistry 41, 135-150 (2014)

[57] Marini, L., Ottonello, G., Canepa, M., and Cipolli, F.: Water-rock interaction in the Bisagno valley (Genoa, Italy): Application of an inverse approach to model spring water chemistry. Geochimica et Cosmochimica Acta 64, 2617-2635 (2000)

[58] Mironenko, M.V. and Zolotov, M.Yu.: Equilibriumkinetic model of waterrock interaction. Geochemistry International 50, 1-7 (2012)

[59] Scislewski, A. and Zuddas, P.: Estimation of reactive mineral surface area during water-rock interaction using fluid chemical data. Geochimica et Cosmochimica Acta 74, 6996-7007 (2010)

[60] Schott, J., Oelkers, E.H., Benezeth, P. Godderis, Y., and Francois, L.: Can accurate kinetic laws be created to describe chemical weathering? Comptes Rendus Geoscience 344, 568-585 (2012)

[61] Kulik, D.A., Thien, B., and Curti, E.: Partial-equilibrium concepts to model trace element uptake. Goldschmidt2012 Conference Abstract, Montreal (2012)

[62] Wolthers, M., Nehrke, G., Gustafsson, J. P., and Van Cappellen, P.: Calcite growth kinetics: Modeling the effect of solution stoichiometry. Geochimica et Cosmochimica Acta 77, 121-134 (2012)

[63] Nielsen, L.C., De Yoreo, J.J., and DePaolo, D.J.: General model for calcite growth kinetics in the presence of impurity ions. Geochimica et Cosmochimica Acta 115, 100-114 (2013)

[64] Geiger, S., Roberts, S., Matthai, S.K., Zoppou, C., Burri, A., (2004) Combining finite element and finite volume methods for efficient multiphase flow simulations in highly heterogeneous and structurally complex geologic media. Geofluids (4), 284-299.

[65] Bear J., Dynamics of fluids in porous media. Elsevier, 1972. 764 pp.

[66] Matthäi, S.K., Nick, H.M., Pain, C., Neuweiler, I., (2010), Simulation of Solute Transport Through Fractured Rock: A Higher-Order Accurate Finite-Element Finite-Volume Method Permitting Large Time Steps. Transport in Porous Media 83(2), 289-318.

[67] Nordstrom, D.K., Plummer, L.N., Wigley, T.M.L., Wolery, T.J., Ball, J.W., Jenne, E.A., Bassett, R.L., Crerar, D.A., Florence, T.M., Fritz, B., Hoffman, M., Holdren, G.R.Jr., Lafon, G.M., Mattigod, S.V., McDuff, R.E., Morel, F., Reddy, M.M., Sposito, G., and Thrailkill, J. (1979) A comparison of computerized chemical models for equilibrium calculations in aqueous systems. In: Chemical modeling in aqueous systems - Speciation, sorption, solubility, and kinetics: Series 93 (Ed E.A. Jenne), American Chemical Society, 857-892.

[68] Thoenen,T., Hummel, W., Berner, U., Curti E. (2014) The PSI/Nagra Chemical Thermodynamic Database 12/07, PSI Bericht Nr. 14-04, 1-416.

[69] Blanc, P., Lassin, A., Piantone, P. (2007) THERMODDEM a database devoted to waste minerals.BRGM (Orlans, France). http://thermoddem.brgm.fr

[70] Helgeson, H. C., Kirkham, D. H., Flowers, D. C., Theoretical prediction of the thermodynamic behavior of aqueous electrolytes at high pressures and temperatures: IV. Calculation of activity coefficients, osmotic coefficients, and apparent molal and standard and relative partial molal properties to 600 C and 5 kb. Am. J. Sci ., v. 281, p. 12491516, 1981.

[71] Pruess, K. EOS7, An Equation-of-State Module for the TOUGH2 Simulator for Two-Phase Flow of Saline Water and Air, Lawrence Berkeley Laboratory Report LBL-31114, Berkeley, CA, 1991b.

[72] Leal, A.M.M., Blunt, M.J., LaForce, T.C. A chemical kinetics algorithm for geochemical modelling. Applied Geochemistry 55, 46-61 (2015)

[73] Machel H.G. and Lonnee J. (2002) Hydrothermal dolomite A product of poor definition and imagination. Sedimentary Geology, 152, 163-171.

[74] Lucia, E.J. (2004) Origin and petrophysics of dolostone pore space. In: Braithwaite, C.J.R., Rizzi, G. and Darke, G. (eds) The Geometry and Petrogenesis of Dolomite Hydrocarbon Reservoirs. Geological Society, London, Special Publications, 235, 141-155.

[75] Jones, G.D., I. Gupta and E. Sonnenthal (2011) Reactive transport models of structurally controlled hydrothermal dolomite: Implications for Middle East carbonate reservoirs. 9th Middle East Geosciences Conference, GEO 2010. GeoArabia, v. 16, no. 2, p. 194-195.

[76] Gomez-Rivas, E., Corbella, M., Martin-Martin, J.D., Stafford, S.L., Teixell, A., Bons, P.D., Griera, A., Cardellach, E. Reactivity of dolomitizing fluids and Mg source evaluation of fault-controlled dolomitization at the Benicssim outcrop analogue (Maestrat basin, E Spain). Marine and Petroleum Geology 55, 26-42 (2014)

[77] Martin-Martin, J.D., Gomez-Rivas, E., Bover-Arnal, T., Trave, A., Salas, R., Moreno-Bedmar, J.A., Tomas, S., Corbella, M., Teixell, A., Verges, J., Stafford,

S.L. (2013) The Upper Aptian to Lower Albian syn-rift carbonate succession of the southern Maestrat Basin (Spain): Facies architecture and fault-controlled stratabound dolostones. Cretaceous Research, 41, 217236.

[78] Gomez-Rivas, E., Stafford, S.L., Lee, A.G.K., Corbella, M., Martin-Martin, J.D., Teixell, A. (2010) Flow patterns of dolomitizing solutions in a buried carbonate ramp – the Benicassim case study (Maestrat Basin, NE Spain). Paper SPE 39522, presented at the 72nd European Association of Geoscientists and Engineers Conference and Exhibition. Barcelona, June (Incorporating SPE EUROPEC 2010, 4, 29542959).

[79] Sharp, I., Gillespie, P., Morsalnezhad, D., Taberner, C., Karpuz, R., Verges, J, Horbury, A., Pickard, N., Garland, J., Hunt, D., (2010) Stratigraphic architecture and fracture-controlled dolomitization of the Cretaceous Khami and Bangestan groups: an outcrop case study, Zagros Mountains, Iran., in: van Buchem, F.S.P., Gerdes, K.D., Esteban, M., (Eds.), Mesozoic and Cenozoic Carbonate Systems of the Mediterranean and the Middle East: Stratigraphic and Diagenetic Reference Models, Geol. Soc, Lond. Sp. Publ. 329, London, pp. 343-396.