DISSERTATION

submitted
to the

Combined Faculty for the
Natural Sciences and Mathematics

of

Heidelberg University, Germany

for the degree of

Doctor of Natural Sciences

Put forward by

Dipl.-Math. techn. Martin Wlotzka
born in Essen, Germany

Date of oral examination:
September 25, 2017

Parallel Numerical Methods for Model Coupling in Nutrient Cycle Simulations

Advisor: Prof. Dr. Vincent Heuveline
Co-Advisor: Prof. Dr. Klaus Butterbach-Bahl

Abstract

We present a new approach to ecosystem nutrient cycle simulations. Nutrient conversion and fluxes between ecosystem compartments are driven by highly complex biogeochemical and hydrological processes. Nutrient cycles in soils supply vegetation and crop growth, affect groundwater and surface water eutrophication, and release greenhouse gases into the atmosphere. Simulating nutrient cycles is regarded a grand challenge due to the multi-scale properties and involved multiphysics of the considered ecosystems. Common approaches are restricted on several levels, often suffering from limited temporal and spatial extent, resolution or accuracy, model simplifications, or inability to use high performance computing effectively.
In order to cope with their inherent complexity, we consider nutrient cycle simulations as a multiphysics problem by means of a coupling of dedicated biogeochemical and hydrological models. We formulate the model coupling problem in an abstract multiphysics setup to manage the complexity. We propose a new variant of operator splitting schemes for the time propagation of the coupled models. Our scheme employs local model propagators such that accuracy is maintained on the global level. Furthermore, the scheme features an inherent parallelism on the coupling level which is independent of possible parallelizations of the models. We present advances in a software technique which facilitates the model coupling on high performance computing platforms. Our development allows for dynamically changing the parallel configuration of models and their data distribution during runtime. We validate our approach to nutrient cycle simulations by means of numerical experiments using a greenhouse gas emission scenario and a nitrate leaching scenario. The results show the benefits of the proposed scheme in terms of an improved coverage of indirect and local effects. Furthermore, we present performance tests showing the superior parallel efficiency of our methodology over common approaches, which is due to its parallelism with respect to the coupling strategy.

Zusammenfassung

Wir präsentieren einen neuen Ansatz für die numerische Simulation von Nährstoffkreisläufen. Hochkomplexe biogeochemische und hydrologische Prozesse bestimmen die Umsetzung von Nährstoffen und Flüsse zwischen Bestandteilen von Ökosystemen. Kreisläufe in Böden versorgen Vegetation mit Nährstoffen, tragen zur Eutrophisierung von Gewässern bei, und lassen Treibhausgase in die Atmosphäre entweichen. Ihre numerische Simulation ist aufgrund der Multi-Skalen- und Multi-Physik-Eigenschaften der betrachteten Ökosysteme eine große Herausforderung. Übliche Vorgehensweisen sind oft auf mehreren Ebenen Einschränkungen unterworfen, etwa durch beschränkte zeitliche oder räumliche Ausdehnung der Simulation, durch beschränkte Auflösung oder Genauigkeit, durch Vereinfachungen der Modelle, oder durch mangelnde Eignung zur effektiven Nutzung von Hochleitsungsrechnern.
Wir betrachten die Simulation von Nährstoffkreisläufen im Sinne einer Kopplung von dedizierten biogeochemischen und hydrologischen Modellen. Um die Komplexität zu bewältigen, führen wir die Kopplung der Modelle in eine abstrakte Formulierung über. Wir schlagen eine Variante von Operator-Splitting-Methoden vor, mit deren Hilfe konsistente und konvergente Zeitschrittverfahren für das gekoppelte System definiert werden können. Darüber hinaus präsentieren wir eine Weiterentwicklungen einer Kopplungssoftware, mit deren Hilfe Multiphysikprobleme effizient auf Hochleistungsrechnern implementiert werden können. Wir zeigen die Stichhaltigkeit unserer Vorgehensweise anhand eines Szenarios mit Treibhausgasemissionen und eines Szenarios mit Nitratausspülungen. Die Ergebnisse zeigen die Vorteile unseres Ansatzes, mit dessen Hilfe lokale und indirekte Effekte in Nährstoffkreisläufen wesentlich genauer erfasst werden können als mit üblichen Vorgehensweisen.

Mathematical contributions

We consider a coupling of dedicated biogeochemical and hydrological models for nutrient cycle simulations. For the time propagation of the coupled models, we propose the use of composed one step schemes which employ local model propagators to form time integration schemes for the global system. The main mathematical contributions are the proofs of first order in time consistency and convergence for the composed one step schemes. The proofs use assumptions on continuity, boundedness and Lipschitz conditions. A family of ordinary differential equations which relates the single models to the global system allows to convey the proof techniques known from one step methods to the composed schemes. We complement the theorems with numerical experiments in a natural convection scenario showing the expected first order in time convergence. Furthermore, we present developments of advanced parallel communication routines for the OpenPALM software coupler tool to implement the composed one step schemes in model coupling applications. This enables the effective use of high performance computing for the considered nutrient cycle simulations.

Acknowledgements

I am taking this opportunity to say thanks to the numerous people who accompanied and supported me during my doctoral studies.

First of all, I am deeply grateful to my advisor Prof. Vincent Heuveline. You guided my work with trust and freedom, beginning at Karlsruhe Institute of Technology, then at Heidelberg University at the Interdisciplinary Center for Scientific Computing and at the Faculty of Mathematics and Computer Science, and finally at the University Computing Center.
Not a bit less am I grateful to my co-advisor Prof. Klaus Butterbach-Bahl. You warmly welcomed me in the collaboration and opened a new scientific field for me.
Enormous thanks are dedicated to all my past and present colleagues in Karlsruhe, Heidelberg and Garmisch-Partenkirchen. Thanks for the uncountable discussions, coffees, Mensa lunches, meetings and Klausurtagungen, which I always enjoyed with you and which you made my daily joy and motivation.
I also thank very much Prof. Olivier Thual and his group at Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique in Toulouse. You made my research stay a great experience.
Furthermore, I acknowledge the support of the German Research Foundation (DFG) under grant no. HE 4760/4-1.

Finally, I sincerely thank my family for their support outside of the office. You backed me up with encouragement and patience. In particular my nearest loved ones Kristina, Sophia and Jonathan, thank you so much!

Contents

1 Introduction

Major global concerns such as alimentation of the world population, freshwater quality, and climate change are affected by ecological processes related to the nutrient cycle. Nutrients are those chemical elements and compounds which are essential for the growth of living organisms. Among the most important nutrients are carbon and nitrogen. Nutrient cycles in soils are particularly important for agriculture, for water quality management, and for climate. They supply vegetation and crop growth, affect groundwater and surface water eutrophication, and release greenhouse gases such as carbon dioxide, methane or nitrous oxide into the atmosphere. Understanding nutrient cycles is thus crucial for assessing nutrient budgets, improving agricultural land management practices, maintaining water quality, and mitigating greenhouse gas emissions from soils.

Nutrient cycles involve highly complex biological, geochemical and hydrological processes. Microbial activity, soil and vegetation properties, plant uptake, land management, water availability and transport are among the primary drivers of nutrient conversion and fluxes between ecosystem compartments. It is often a difficult task to quantify their impact on nutrient cycles due to the complexity of ecosystems. Moreover, intricate interactions between the involved processes give rise to indirect and local effects which can contribute significantly to nutrient cycles, but are even harder to capture and predict. Measurement techniques like closed chambers, fast boxes, or eddy covariance techniques are limited both in the temporal and in the spatial scale. Indirect effects can occur with such temporal and spatial extent that it is practically impossible to measure and trace them back to their origin. Furthermore, capturing local effects like emission hot spots and hot moments requires a high temporal and spatial measurement resolution. This may lead to prohibitive costs for equipment and effort for operation. Therefore, predicting ecosystem feedback on changes in land use or climate, and evaluating mitigation strategies for reducing greenhouse gas emissions and water eutrophication can not solely rely on measurements, which are not always feasible for the needed time and space scales.

The ecosystem research community has responded to this issue with the development of numerical simulation models. The genesis of terrestrial nutrient cycle simulations dates back to the 1980s, when the foundations of process-oriented biogeochemical models were laid. Seeking to capture the complex ecosystem dynamics, biogeochemical models have evolved to a state where many relevant processes are incorporated. Today, the "DeNitrification / DeComposition"(DNDC) model system is widely adopted for carbon and nitrogen cycle simulations. However, biogeochemical models are typically one-dimensional in space, representing a single soil column. Lateral fluxes are ignored in such models, although they contribute significantly to nutrient cycles. In particular, topographical effects on hydrology and lateral transport of nutrients are neglected. Furthermore, many biogeochemical models lack of a precise resolution of aerobic and anaerobic soil conditions, and transitions between them, which are also largely affected by hydrology. This limits the accuracy of simulations severely since oxygen availability has an essential impact on biogeochemical processes. Overall, the ecosystem research community realized that hydrological processes need to be incorporated into biogeochemical simulations in a much more profound way as it has been done so far.

Efforts to overcome these shortcomings mainly aim in two directions. On the one hand, hydrological models are extended with simplified biogeochemical process descriptions. On the other hand,

biogeochemical models are coupled in a one-way fashion to a hydrological model which serves as a transport module. However, the common approaches to improve nutrient cycle simulations are facing major challenges:

1. The issue of different temporal and spatial scales.
   Models need to resolve short-term and localized events at the site scale occurring within days, hours or even minutes to capture hot spots and hot moments. But they also need to account for long-term and large-scale processes on the landscape scale over years or even decades to assess nutrient budgets and indirect effects.

2. The issue of different modeling approaches and numerical treatment.
   Hydrological models are often given in terms of partial differential equations, e.g. by means of the Richards equation for subsurface flow and kinematic wave equations for overland flow. The hydrological model domain is usually three-dimensional, with fully coupled model equations for all space dimensions. In contrast, the one-dimensional biogeochemical models are typically given in terms of functional relations of nutrient availability, microbial activity and other factors. Domains with lateral extent are covered by placing several independent model instances next to each other. Therefore, the models demand for different numerical treatment. In particular, the definition of biogeochemical models has usually not a continuous, but a discrete character in the first place, both with respect to space in terms of soil layers and with respect to time in terms of predefined time steps.

3. The issue of different demands for computational resources.
   The biogeochemical model computations for one time step usually require only moderate computing resources due to their site-oriented nature and plain mathematical structure. Moreover, the parallelization is straight forward since the model instances can be calculated independently from each other. In contrast, hydrological model computations can be much more demanding, and harder to parallelize, due to their fully coupled three-dimensional nature. Also the time step sizes often need to be much smaller for hydrological models than for biogeochemical ones, thus requiring more steps to be computed.

4. The issue of software complexity and reuse.
   Already the biogeochemical and hydrological simulation codes by themselves are often complex software packages. Incorporating one model into the other, or coupling them, results in additional complexity. Refactoring existing, or even developing new software for nutrient cycle simulations might require prohibitive effort and costs. It is thus a desire to reuse existing simulation codes with as little modifications as possible. However, there are often no interfaces available to facilitate interaction and data exchange, since biogeochemical and hydrological models have mainly been developed in separate communities.

Biogeochemistry and hydrology simulations are already complex matters on their own, and it is even more challenging to combine them properly. The road to further progress in nutrient cycle simulations is blocked by the sketched issues.

1.1 A new approach to nutrient cycle simulations

As we outlined above, difficulties are encountered on multiple levels. Therefore, a new approach is needed to respond to the severe challenges in nutrient cycle simulations. First of all, we acknowledge the fact that well-established models and simulation codes are available for biogeochemistry as well as for hydrology as the starting position of our work. It is desirable to reuse existing

codes for both scientific and practical reasons. In this thesis we present a new methodology for combining existing biogeochemical and hydrological models with minimal code modifications to improve nutrient cycle simulations. The key novelty is a model coupling technique which maintains the capabilities, the accuracy, and the computational performance of the models which are used as building blocks of the overall system. The basis of our approach is an abstraction of the model coupling for nutrient cycle simulations to a more general multiphysics setup. On this basis, we develop the tools to address all four issues mentioned above, both on the mathematical level and on the implementation level. The abstraction of nutrient cycle simulations to general multiphysics setups is the key to reduce the complexity. We present a new variant of operator splitting methods for the time propagation of coupled models on that abstract multiphysics level. Furthermore, we present new software developments to facilitate model coupling and interaction. The proposed software coupling technique is the key tool to implement our new operator splitting variant for parallel platforms and high performance computing systems. This is crucial to meet the demand for computational resources of complex applications. Finally, we demonstrate the feasibility of our approach to nutrient cycle simulations by means of numerical experiments on a greenhouse gas emission scenario, and on a nutrient leaching scenario. In the following subsections, we introduce further details of our approach and highlight the main thesis contributions.

1.1.1 Abstraction to a general multiphysics setup

The core idea of our approach is to couple dedicated models for the biogeochemical and for the hydrological processes. The desire to include all relevant phenomena in numerical simulations is clearly not new, but rather natural. Indeed, "*simulations that couple multiple physical phenomena are as old as simulations themselves*" [52, p. 1]. Our approach is however new in that it can significantly improve the accuracy and the computational efficiency of nutrient cycle simulations, and it enables to capture local and indirect effects. Furthermore, our model coupling technique is applicable to general multiphysics problems, not only to nutrient cycle simulations. This is a benefit of our measure to manage the complexity of nutrient cycle simulations by using an abstraction to a general multiphysics model coupling setup. The term multiphysics denotes systems which consist of more than one physical component or quantity governed by its own principle for evolution or equilibrium. In our case, these are the biogeochemical and the hydrological processes. The classical approach to multiphysics is to form a system of equations which incorporates all considered phenomena. Such systems may in general involve algebraic equations, ordinary differential equations (ODE), as well as partial differential equations (PDE). Solving the multiphysics system as a whole is called a "*monolithic*" scheme. Other types of coupling and solution schemes exists, which can be described using the terminology of Keyes et. al. (2011). They introduce the notion of "*strong*", "*weak*", "*tight*" and "*loose*" coupling in the following sense [52, p. 73]. The attribution of strong versus weak coupling refers to the physics represented by the models. A coupling is called strong or weak if the physics of the models have a strong or weak influence on each other. On the mathematical level, strongly coupled models may lead to stiff problems. In contrast, the attribution of tight versus loose coupling refers to numerical or algorithmic aspects of multiphysics simulations. Tight coupling schemes attempt to keep the state variables synchronized across models at all times, whereas loose coupling schemes allow for non-synchronous model states. A typical example of a loose coupling is the shift of one model by one time step against another model.
Our abstraction from nutrient cycle simulations to a general multiphysics setup consists in the assumption that models are given in terms of a functional representation. We focus on the case of time-dependent models which can be represented by a function describing their temporal vari-

ation. Such functional representation may result from models with purely temporal variability, or from spatially discretized PDE, possibly using the method of lines [94]. This abstract setup opens a broad scope for multiphysics applications. It includes all kinds of models defined by ODE, and all kinds of models defined by PDE as long as they are accessible in a spatially discretized form. The latter requirement is mostly not a limitation, since numerical simulations of models defined by PDE usually involve a spatial discretization. The generality on the coupling level is the key for addressing the four issues we identified above. Based on this generic principle, the properties of the solvers used for the separate models can usually be leveraged in a convergent and efficient scheme for the global problem. Furthermore, it allows to effectively use high performance computing even if individual models demand for different computational resources, and it allows to reuse existing simulation codes in the coupling.

1.1.2 Composed one step schemes

As indicated above, we assume in our abstract view on multiphysics that models are given in terms of a functional representation describing their temporal variation, i.e. models are given as the right hand side functions of ODE. We already pointed out that this may also include models with spatial variation, e.g. PDE in spatially discretized form. As part of our novel model coupling technique we propose the notion of "*composed one step schemes*" for propagating the coupled models in time. The characteristic properties of our composed one step schemes are that they represent one step methods on a global time grid, and that the global solution for a given time step is composed of individual model contributions. The concept of composed one step schemes is derived from classical operator splitting methods. We present two variants of composed one step schemes which we call "*consecutive*" and "*concurrent*" operator splitting, respectively. In the consecutive variant, the individual model contributions are computed one after another, while the concurrent variant features an inherent parallelism among models. Both variants allow for internal parallelization in the individual model implementations. On top of that, the concurrent variant offers an additional level of parallelism which is independent of possible internal model parallelism.
We already published the notion of consecutive and concurrent operator splitting schemes in our previous works [108, 107]. However, their rigorous definition is an original thesis contribution. The consecutive operator splitting is equivalent to the well-known Lie(-Trotter) splitting [89] in our setup, whereas there is no classical operator splitting counterpart for the concurrent scheme. The major thesis contributions on composed one step schemes are our proofs of first order in time consistency and discrete stability, and thus convergence, of both the consecutive and the concurrent variant. By these proofs, we re-establish the known first order convergence of the Lie(-Trotter) splitting in our setup, and we show the first order convergence of our new concurrent variant. Our proofs rely solely on the basic assumptions of continuity, boundedness and Lipschitz conditions of the functional model representations without requiring any further knowledge of possible internal mathematical structure of the models.
We supplement our theoretical convergence results with numerical experiments. We investigate the numerical convergence of our composed one step schemes in a natural convection scenario comprising a fluid dynamics model and a temperature model. The results show the expected first order in time convergence in different flow regimes.

1.1.3 Dynamic OpenPALM distributors

We address multiphysics on the mathematical level by means of a composed one step scheme. On the software level, the issue of different demand for computational resources, and the issue

of software complexity and reuse require dedicated techniques for efficiently implementing multiphysics systems. Assuming the proposed modeling, the coupling of existing biogeochemical and hydrological models implies important modifications of the underlying model codes. We employ a dedicated software coupler tool named OpenPALM to realize our multiphysics approach to nutrient cycle simulations. OpenPALM is a general purpose software coupler tool which serves as the link between the coupled models. It controls the execution of the models and manages data transfer between them. A core feature of OpenPALM is its ability to support two levels of parallelism, both on the coupling level and on the model level. OpenPALM's coupling level parallelism allows to execute multiple models at the same time. This feature on the implementation side is the exact pendant of the concurrent operator splitting parallelism on the mathematical side. OpenPALM's model level parallelism allows to manage data transfer between models which are internally parallelized. The coupler is able to transfer distributed data between parallelized models, including the case of differing data distributions on the two end points of a transfer. OpenPALM maintains the integrity of global data objects by properly arranging the transfer of corresponding local object parts between models. This feature is essential for allocating adequate computational resources to the biogeochemical and hydrological models in order to meet their different demands.

In OpenPALM such data transfer between parallelized models is however restricted to the case where the data distribution is known a priori. It has to be specified in a configuration which is processed during the compilation of the multiphysics application. Moreover, the data distribution cannot be changed during runtime of the application. A further restriction is that only data objects of a priori known and fixed size can be transferred in parallel. These restrictions have major implications on the usage of OpenPALM. The size of data objects is often not determined a priori, but during runtime in the setup phase of an application. For instance, vector sizes may depend on various factors like mesh size or discretization parameters, which are only known at runtime. Also the distribution of data among the processes of parallelized models is often determined at runtime, e.g. through a domain decomposition. OpenPALM must however be supplied with data sizes and distributions prior to application startup. The usual way to determine them is to perform offline test runs of the separated models using the exact same input data, parameters and parallelization as later in the multiphysics model coupling, and to note down the resulting data sizes and distributions. This may result in substantial overhead for setting up the OpenPALM configuration. Furthermore, the configuration is then fixed throughout the whole simulation. This restriction prevents from common practices like refinement or load balancing which would require to change data sizes and distributions during runtime. The usual way using OpenPALM is to shut down the application, determine the new configuration with offline tests, and then restart the application with the new configuration.

We present a new development in the framework of OpenPALM which allows to overcome these restrictions by means of "*dynamic distributors*". These distributors drop the problematic requirement of specifying data sizes and distributions a priori, and thus remove the restrictions described above. This new feature allows to change the size of data objects as well as their distribution during runtime of the application. This eases the usage of OpenPALM greatly since neither the preliminary offline testing is needed anymore, nor the shutdown, reconfigure, restart procedure. Instead it is possible to temporize the configuration of the transfer mechanism until data sizes and distributions are known after the setup phase of applications, and also to change the configuration any time during runtime without interrupting the execution. Based on our development OpenPALM thus fully supports the usual course of scientific computing applications where vector and matrix sizes and their distribution are not determined a priori, but only at runtime in a setup phase, and where data sizes and distributions may change during runtime due to refinement or rebalancing.

1.1.4 Ecosystem scenarios

With the adequate mathematical methods and software tools in hand, we demonstrate the effectiveness of our approach to nutrient cycle simulations by coupling dedicated biogeochemical and hydrological models. We present a greenhouse gas emission scenario and a nutrient leaching scenario, where indirect and local effects contribute significantly to nutrient cycles in both cases. In order to capture these effects, simulations need to resolve the biogeochemical and the hydrological processes accurately enough. We show that our model coupling approach can substantially improve the ability to cover indirect and local effects compared to common nutrient cycle simulation techniques. Furthermore, we exemplify the parallelization and efficient use of high performance computing for the simulations even though the individual models exhibit different demand for computational resources. This allows for sufficiently small time steps on an hourly or even sub-hourly basis, while at the same time considering ample simulation times of multiple years or even decades. The proper combination of our composed one step schemes with our dynamic distributors feature is an enabling technique for nutrient cycle simulations, and for multiphysics model coupling in general.

1.2 Thesis outline

Chapter 2 is dedicated to the derivation of the composed one step schemes. To lay a basis for the derivation, we briefly recall some basic results from the classical theory of ordinary differential equations and their numerical treatment. The main emphasis in Chap. 2 is the definition of the composed one step schemes and proofs of their first order in time consistency, discrete stability and convergence. Chapter 3 presents numerical experiments on the convergence of the composed one step schemes using a natural convection scenario. We investigate the numerical accuracy of our first order composed one step schemes and compare with a second order in time monolithic scheme. The expected convergence orders are accurately achieved even for large time step sizes in a wide range of flow regimes. Chapter 4 is devoted to the development of the dynamic distributors, i.e. the advanced parallel communication capabilities of OpenPALM. We introduce the concepts and functionalities of OpenPALM as a general purpose dynamic parallel software coupler tool. We add further details on the previously existing parallel communication routines which we use as the baseline for our developments. We complete the chapter with the presentation of our new dynamic parallel communication features. Chapter 5 presents the multiphysics ecosystem nutrient cycle simulations. We demonstrate the use of our composed one step schemes in conjunction with our new OpenPALM dynamic distributors data transfer feature for coupling a biogeochemistry model and a hydrology model. We introduce the basic processes which affect nutrient cycles in soils and the corresponding modeling approaches. To overcome the limitations of traditional modeling approaches in assessing indirect and local effects, we present our approach of coupling dedicated models for biogeochemical and hydrological processes. We reuse existing model implementations and realize the coupling and data exchange with OpenPALM, which allows to allocate high performance computing resources adequately for both models to balance their different computational demands. We demonstrate the feasibility of our approach by means of numerical experiments comprising a nitrous oxide emission scenario and a nitrate leaching scenario, and we present parallel performance tests. Chapter 6 concludes this thesis with a summary of the main results and perspective remarks.

2 Abstract multiphysics setup and composed one step schemes

This chapter introduces the mathematical basis of our approach to nutrient cycle simulations. We consider a coupling of dedicated biogeochemical and hydrological models in a multiphysics application. Coupling dedicated models allows to address the issues identified in the introduction in several respects. Models and their numerical treatment may be tailored to adequately match the needed temporal and spatial scales, and existing model implementations may be reused in the coupling. A coupling methodology is then needed which yields a well-defined global system, and which at the same time maintains the properties of the models and their solvers on the local level. Moreover, the coupling methodology is desired to allow to allocate suitable computational resources to the models, and to deal with software complexity.
As we outlined in the introduction, we accept a bottom up view considering existing models as the starting point of our work. The specific modeling and implementation of the biogeochemical and hydrological processes which we use for nutrient cycle simulations are discussed in Chapter 5. By reusing existing models and codes, we intend to leverage the capabilities on the model level as described above. We complement this bottom up view with a top down view on the coupling level. In the top down view, we propose a coupling methodology to form a consistent global multiphysics system using existing models as building blocks. We assume the availability of models in an abstract mathematical formulation which serves as the interface between bottom up model view and top down coupling view.

2.1 Abstract multiphysics problem formulation

We consider the following very basic multiphysics situation: two models interact by simultaneously affecting a common quantity y. This might include spatially and temporally variable quantities, model interfaces, boundary values, or other quantities. In nutrient cycle simulations, the common quantity may include soil water content or nutrient concentrations. From the bottom up view, models might be given in terms of equations involving y, or in terms of software which implements actions on y. We assume that the common quantity is time dependent, i.e. $y = y(t)$, which suits our ecosystem models as well as any model with temporal variability in general. In particular, models given in terms of ODE comply with this assumption. Note that possible space dependency of y is not explicitly considered on the coupling level in our approach. Instead, we assume that any possible space dependency of y is treated on the model level, and only the time dependency is exposed to the coupling level. This does mostly not prevent from considering models with space dependence, e.g. models given in terms of PDE, provided that only time dependence is exposed. Space dependence may be treated by means of a discretization, e.g. using the method of lines for PDE.
Our abstraction from the processes related to nutrient cycles to a general setup consists in the assumption that models are given in terms of functions describing the temporal variation $\dot{y}$ of the common quantity. For two models, the individual model action is formulated as $\dot{y} = f(t, y)$ and $\dot{y} = g(t, y)$, respectively. Insofar, the functions f and g represent the effects of the individual

models on the common quantity y. The coupled multiphysics system can then be stated in the form of the following initial value problem

$$\begin{aligned} \dot{y} &= f(t,y) + g(t,y) \ , \ t \in I \ , \\ y(0) &= y_0 \ , \end{aligned} \tag{2.1}$$

where $I = [0,T]$ with some final time $T > 0$ is the time interval under consideration, and y_0 is a given initial value of the common quantity. The setup can obviously be extended to any number of models acting on the same common quantity y. Also systems of the form

$$\begin{pmatrix} \dot{y_1} \\ \dot{y_2} \end{pmatrix} = \begin{pmatrix} \hat{f}(t,y_1,y_2) \\ \hat{g}(t,y_1,y_2) \end{pmatrix} , \tag{2.2}$$

can be reformulated in the sense of (2.1) using

$$y = (y_1,y_2)^\top \ , \quad f(t,y) = \begin{pmatrix} \hat{f}(t,y_1,y_2) \\ 0 \end{pmatrix} , \quad g(t,y) = \begin{pmatrix} 0 \\ \hat{g}(t,y_1,y_2) \end{pmatrix} .$$

Equation (2.1) is the abstract global problem formulation which we use in our approach to multiphysics model coupling. It is the interface between the bottom up model view and the top down coupling view. It abstracts from the specific models at hand to a formulation which is reduced to the role of the common quantity y as the mathematical medium of the coupling.
The ODE form of the abstract setup describes the temporal evolution of the common quantity y under the simultaneous influence of both models. One might in principal consider any numerical method for solving (2.1), such as one step or multi step methods, Galerkin methods, extrapolation methods, splitting methods, or others. The application of an ODE solver to the combined right-hand-side function $f + g$ is a standard approach and results in a monolithic scheme. However, embracing $f + g$ in a monolithic scheme is opposed to our approach of coupling dedicated models. On the conceptual side, a monolithic scheme treats both models with the same solver which can result in exactly those issues identified in the introduction. On the practical side, the individual models represented by f and g might only be separately accessible, e.g. if they are implemented in separate codes and applications.
We adopt the idea of splitting methods [89], or operator splitting, for our approach to model coupling. Splitting methods, such as the well-known Lie(-Trotter) splitting or the Strang splitting, treat the constituents of the coupling separately by means of individual model propagators and combine the split solutions to yield a global solution. We use this idea to establish our own notion of coupled model propagators, which we call composed one step schemes. Our aim is to compose global propagators from one step methods on the model level. The composed one step schemes yield numerical solutions of the global multiphysics problem on a global time grid. Naturally, the global time steps are synchronization points for the models. Note that this does not prevent individual models from computing local sub-steps, possibly using other types of time stepping schemes such as multistep methods, as long as the models are synchronized at the global time steps. We propose two variants of composed one step schemes which we call consecutive and concurrent, respectively. The consecutive variant resembles the Lie(-Trotter) scheme in our setup, while the concurrent variant is dissimilar from the known splitting schemes. Both schemes are first order in time consistent and convergent.

In the first section, we recall some basic results from the classical theory of ODE which are relevant for the proofs of consistency and convergence of the proposed schemes. The second section recalls the basic definition of one step methods, and the concepts of consistency, discrete stability

and convergence. The last section is devoted to the presentation of splitting and composition schemes, our proposed schemes for coupled models, and the proofs on their consistency, discrete stability and convergence.

2.2 Classical theory of ordinary differential equations

In this section, we outline some results form the classical theory on the existence, regularity, uniqueness and stability of solutions for systems of ordinary differential equations. To state the results which are relevant for the derivation of the composed one step schemes in Sec. 2.4.1, we use a first order ODE of the form

$$\dot{y}(t) = f(t, y(t)) \, , \, t \in I \tag{2.3}$$

with an independent variable $t \in I \subset \mathbb{R}$, a right-hand-side function $f(t,y) = \big(f_1(t,y), ..., f_d(t,y)\big)^\top$, and the unknown function $y(t) = \big(y_1(t), ..., y_d(t)\big)^\top$. The independent variable is often associated with time and, unless otherwise indicated, we chose without loss of generality the interval $I = [0, T]$ with some $T > 0$. We assume that f is defined on

$$D = I \times \Omega \subset \mathbb{R} \times \mathbb{R}^d$$

with some set $\Omega \subset \mathbb{R}^d$. For a given initial value $y_0 \in \Omega$, we seek a solution of (2.3) according to the following definition [84, p. 13].

Definition 2.1 (Initial Value Problem)
The problem of finding a continuously differentiable function $y : I \longrightarrow \mathbb{R}^d$ which satisfies

$$Graph(y) = \big\{(t, y(t)) \, , \, t \in I\big\} \subset D \, , \tag{2.4a}$$
$$\dot{y}(t) = f(t, y(t)) \, , \, t \in I \, , \tag{2.4b}$$
$$y(0) = y_0 \, , \tag{2.4c}$$

is called an initial value problem (IVP) with problem data f, $D = I \times \Omega$, and y_0.

In the following, we use the notation for the p-Norms ($1 \leq p < \infty$) and the maximum norm

$$\|y\|_p = \Big(\sum_{i=1}^{d} |y_i|^p \Big)^{1/p} \, , \, \|y\|_\infty = \max_{1 \leq i \leq d} |y_i| \, , \, y \in \mathbb{R}^d \, ,$$

and for the Euclidean norm and the scalar product

$$\|y\| = \|y\|_2 \, , \, (x, y) = \sum_{i=1}^{d} x_i y_i \, , \, x, y \in \mathbb{R}^d \, .$$

We first state the classical existence theorem by Peano. The Peano theorem assumes continuity of the right-hand-side function f of the initial value problem and guaranties the existence of a local solution.

Theorem 2.1 (Local Existence Theorem by Peano)
Let $f : D \longrightarrow \mathbb{R}^d$ be continuous on

$$D = \big\{(t, y) \in \mathbb{R} \times \mathbb{R}^d : |t| \leq \alpha, \|y - y_0\| \leq \beta\big\}$$

where $\alpha, \beta > 0$ are constants. Then there is a solution y of the initial value problem (2.4) *on the interval $I = [-T, T]$ with*

$$T = \min\left\{\alpha, \frac{\beta}{M}\right\}\ ,\ M = \max\left\{\|f(t,x)\| \ :\ (t,x) \in D\right\}\ .$$

Proofs based on the uniform convergence of equidistant Euler polygons to a solution can be found in [84, p. 14] and [37, p. 42].

Remark:
The constant $M = \max\left\{\|f(t,x)\| \ :\ (t,x) \in D\right\}$ in Theorem 2.1 surely exists, since f is assumed to be continuous and therefore attains its maximum on the compact set D. In the case of $M = 0$, i.e. $f \equiv 0$, we have the obvious solution $y \equiv y_0$ on $I = [-\alpha, \alpha]$.

The next two theorems give statements on the continuation of local solutions.

Theorem 2.2 (Continuation Theorem)
Let f be continuous on a closed set $D = J \times \Omega \subset \mathbb{R} \times \mathbb{R}^d$ containing the point $(0, y_0)$, and let y be a solution of the initial value problem (2.4) *on some interval $I = [-T, T] \subset J$. Then the local solution y can be continued up to the boundary of D.*

Proofs can be found in [84, p. 17] and [37, p. 40].

Theorem 2.3 (Global Existence Theorem)
Let f be continuous on $\mathbb{R} \times \mathbb{R}^d$, and let y be a solution of the initial value problem (2.4) *on some interval I. If there is a continuous function $\beta : \mathbb{R} \longrightarrow \mathbb{R}$ such that*

$$\|y(t)\| \leq \beta(t)\ ,\ t \in I\ ,$$

then y can be continued to a global solution on $\mathbb{R}$.

A proof can be found in [84, p. 18].

The following theorem guaranties the regularity of the solution depending on the regularity of the right-hand-side function of the initial value problem.

Theorem 2.4 (Regularity Theorem)
Let y be a solution of the initial value problem (2.4) *on some interval I. If $f \in C^m(D)$ for some $m \geq 1$, then $y \in C^{m+1}(I)$.*

A proof can be found in [84, p. 19].

The theorems on the existence of solutions stated above only assumed continuity of f. However, to address stability and uniqueness of a solution, the additional assumption of a Lipschitz condition is necessary.

Definition 2.2 (Lipschitz Condition)
(1) The function $f : D \longrightarrow \mathbb{R}^d$ satisfies a Lipschitz condition on its domain $D \subset \mathbb{R} \times \mathbb{R}^d$, if there is a positive and continuous function $L : \mathbb{R} \longrightarrow \mathbb{R}$ with

$$\|f(t,x_1) - f(t,x_2)\| \leq L(t)\|x_1 - x_2\|\ ,\ (t,x_1),(t,x_2) \in D\ .$$

In this case, L is called the Lipschitz constant.
(2) The function $f : D \longrightarrow \mathbb{R}^d$ satisfies a local Lipschitz condition on its domain $D \subset \mathbb{R} \times \mathbb{R}^d$, if f satisfies a Lipschitz condition on any bounded subset of D. In this case, the Lipschitz constants may depend on the subsets.

Remark:
Note that a Lipschitz condition implies continuity of f with respect to the second argument, since

$$\|f(t, x_1) - f(t, x_2)\| \leq L(t)\|x_1 - x_2\| \to 0 \quad (x_1 \to x_2) \, .$$

On the other hand, continuity does in general not imply a Lipschitz condition. As a counter example, consider the function $f(t, x) = f(x) = \sqrt{x}$ for $x \in [0, 1]$, which is continuous. With the sequence $x_n := \frac{1}{n}$ $(n \in \mathbb{N})$ we have $x_n \to 0$ $(n \to \infty)$,$|x_n - 0| = \frac{1}{n}$ and $|f(x_n) - f(0)| = \sqrt{\frac{1}{n}}$. Thus for any fixed $L > 0$

$$|f(x_n) - f(0)| > L|x_n - 0| \quad \Longleftrightarrow \quad n > L^2 \, ,$$

therefore no Lipschitz condition holds.

The following theorem guarantees the stability of solutions under the assumption of a Lipschitz condition.

Theorem 2.5 (Local Stability Theorem)
Let f, g be continuous on $D = I \times \Omega \subset \mathbb{R} \times \mathbb{R}^d$, and let two initial value problems be given as

$$\dot{u} = f(t, u) \, , \, t \in I \, , \, u(0) = u_0 \, , \tag{2.5a}$$

$$\dot{v} = g(t, v) \, , \, t \in I \, , \, v(0) = v_0 \, . \tag{2.5b}$$

Furthermore, let f satisfy a Lipschitz condition on D with a Lipschitz constant $L(t)$ and $L := \sup_{t \in I} L(t) < \infty$. Then for any solution u of (2.5a) *and any solution v of* (2.5b) *holds*

$$\|u(t) - v(t)\| \leq e^{Lt} \Big[\|u_0 - v_0\| + \int_0^t \epsilon(s) \, ds \Big] \, , \, t \in I \, ,$$

where $\epsilon(s) = \sup_{x \in \Omega} \|f(s, x) - g(s, x)\|$.

A proof based on the Gronwall Lemma can be found in [84, p. 23].

A direct consequence of the stability is the uniqueness of the solution.

Theorem 2.6 (Local Uniqueness Theorem)
Let f be continuous and satisfy a Lipschitz condition. Then there is a unique local solution y of the initial value problem (2.4).

A proof can be found in [84, p. 24].

We finally state a theorem on global existence and uniqueness.

Theorem 2.7 (Global Existence and Uniqueness Theorem)
Let f be continuous on $D = \mathbb{R} \times \mathbb{R}^d$ and satisfy

$$\|f(t, x)\| \leq \alpha(t)\|x\| + \beta(t) \, , \, (t, x) \in D$$

with non-negative continuous functions $\alpha, \beta : \mathbb{R} \longrightarrow \mathbb{R}$. Then there is a global solution y of the initial value problem (2.4) *on $\mathbb{R}$. Furthermore, if f satisfies a Lipschitz condition, then the global solution is unique.*

A proof based on the generalized Gronwall Lemma can be found in [84, p. 26].

We close this section with the existence theorem by Picard-Lindelöf, which can be proved independently from the Peano theorem. In contrast to the Peano theorem, the Picard-Lindelöf theorem assumes not only continuity, but also a Lipschitz condition on f. Therefore, it guaranties existence and uniqueness of the solution.

Theorem 2.8 (Existence Theorem by Picard-Lindelöf)
Let $f : D \longrightarrow \mathbb{R}^d$ be continuous and satisfy a local Lipschitz condition. Then, for any $(t_0, y_0) \in D$, there is a $T > 0$ and a local solution $y : [t_0 - T, t_0 + T] \longrightarrow \mathbb{R}^d$ of the initial value problem (2.4) Moreover, this local solution is unique.

A proof based on the Banach Fixed Point Theorem can be found in [84, p. 27].

2.3 One step methods

In practice it is often not possible to construct analytical solutions to initial value problems. This might be the result of failing to meet the requirements of the known construction methods like the separation of variables [37, p. 14] or the variation of constants [37, p. 18], lack of knowledge of a construction method which suits the problem at hand, or even incomplete information on the problem data, e.g. if the function f is not entirely known but only given as a computer program. Nevertheless, in many situations where no analytical solution is known, one can compute approximations to the solution by means of numerical methods for solving ordinary differential equations. In this section, we briefly recall the concept of one step methods. These methods are based on a discretization of the time interval $I = [0, T]$ into a grid of $N + 1$ distinct instants of time $0 = t_0 < t_1 < ... < t_N = T$ with time step sizes $h_n = t_n - t_{n-1}$ $(n = 1, ..., N)$ and $h := \max_{1 \leq n \leq N} h_n$. Starting from the given initial value y_0, these methods successively compute approximations $y_1 \approx y(t_1), y_2 \approx y(t_2), ..., y_N \approx y(t_N)$ to the solution y of the initial value problem (2.4). Therefore they are also called time stepping schemes. We call a family of vectors $y^h = (y_n)_{0 \leq n \leq N}$ $(y_n \in \mathbb{R}^d)$ a grid function on the time grid $I^h = \{t_0, ..., t_N\}$.

One step methods use the approximation y_{n-1} from the last time instance t_{n-1} to compute the approximation y_n for the next time instance t_n. One step methods can be stated as

$$y_n = y_{n-1} + h_n F(h_n, t_{n-1}, y_n, y_{n-1}) \,,\; n = 1, ..., N \tag{2.6}$$

or equivalently by means of the difference operator L_h^F as

$$(L_h^F y^h)_n := \frac{y_n - y_{n-1}}{h_n} - F(h_n, t_{n-1}, y_n, y_{n-1}) = 0 \,,\; n = 1, ..., N \tag{2.7}$$

where F is the defining function of the method. Since a one step method is entirely characterized by means of its defining function F, we may call it "method F" for short. To illustrate the role of F, we give some examples of well-known one step methods which are widely used in practice:

- explicit or forward Euler method, also known as Euler polygon method [37, p. 36]:

$$y_n = y_{n-1} + h_n f(t_{n-1}, y_{n-1}) \,, \tag{2.8}$$

  i.e. $F(h_n, t_{n-1}, y_n, y_{n-1}) = f(t_{n-1}, y_{n-1})$

- implicit or backward Euler method [37, p. 204]:

$$y_n = y_{n-1} + h_n f(t_n, y_n)\,, \tag{2.9}$$

  i.e. $F(h_n, t_{n-1}, y_n, y_{n-1}) = f(t_n, y_n)$

- Trapezoidal rule [84, p. 140]:

$$y_n = y_{n-1} + \frac{h_n}{2}\Big[f(t_{n-1}, y_{n-1}) + f(t_n, y_n)\Big]\,, \tag{2.10}$$

  i.e. $F(h_n, t_{n-1}, y_n, y_{n-1}) = \frac{1}{2}\Big[f(t_{n-1}, y_{n-1}) + f(t_n, y_n)\Big]$

- general s-stage Runge Kutta methods [38, p. 40]:

$$g_i = y_{n-1} + h_n \sum_{j=1}^{s} a_{ij} f(t_{n-1} + c_j h_n, g_j)\,,\ i = 1, ..., s \tag{2.11a}$$

$$y_n = y_{n-1} + h_n \sum_{j=1}^{s} b_j f(t_{n-1} + c_j h_n, g_j) \tag{2.11b}$$

  i.e. $F(h_n, t_{n-1}, y_n, y_{n-1}) = \sum_{j=1}^{s} b_j f(t_{n-1} + c_j h_n, g_j)$, where the $g_i\,(i = 1, ..., s)$ are determined through (2.11a). The parameters a_{ij}, b_j and $c_j\,(i, j = 1, ..., s)$ define the specific method at hand. The method is explicit in case of $a_{ij} = 0$ for $i \leq j$, otherwise implicit. Runge-Kutta methods can be represented in a Butcher table as

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}\,,$$

  where $A = (a_{ij})_{1 \leq i,j \leq s}$ is the coefficient matrix and $c = (c_1, \ldots, c_s)^\top$ and $b = (b_1, \ldots, b_s)$ are the coefficient vectors. Note that the one step methods stated above represent Runge-Kutta methods with the following Butcher tables:

0	0
	1

explicit Euler

1	1
	1

implicit Euler

0	0	0
1	0	1
	1/2	1/2

trapezoidal rule

2.3.1 Consistency, discrete stability and convergence

We briefly outline the concepts of consistency, stability and convergence. First we recall the definition of consistency, which formulates the relation of a one step method with the solution of the initial value problem.

Definition 2.3 (Consistency)
Let y be a solution of the initial value problem (2.4)*. The local discretization error, or truncation error, of a one step method* (2.6) *is defined as*

$$\tau_n^h := \frac{y(t_n) - y(t_{n-1})}{h_n} - F(h_n, t_{n-1}, y(t_n), y(t_{n-1}))\ ,\ n = 1, ..., N\ . \tag{2.12}$$

The one step method is called consistent with the initial value problem if

$$\max_{t_n \in I} \|\tau_n^h\| \to 0 \quad (h \to 0)\,,$$

and it is called consistent with order $p > 0$ if for sufficiently smooth y

$$\max_{t_n \in I} \|\tau_n^h\| = O(h^p)\,.$$

Definition 2.4 (Lipschitz Continuity of One Step Methods)
A one step method is called Lipschitz continuous, if its defining function F satisfies a uniform Lipschitz condition with Lipschitz constant $L > 0$ such that

$$\|F(h,t,x,y) - F(h,t,\bar{x},\bar{y})\| \leq L\Big(\|x - \bar{x}\| + \|y - \bar{y}\|\Big)$$

for all $t \in I$ and $x, \bar{x}, y, \bar{y} \in \Omega$.

Theorem 2.9 (Discrete Stability Theorem for One Step Methods)
Let F be a Lipschitz continuous one step method with Lipschitz constant L according to Definition 2.4, and let $y^h = (y_n)_{0 \leq n \leq N}$ and $z^h = (y_n)_{0 \leq n \leq N}$ be grid functions. Then for $h < \frac{1}{2L}$

$$\|y_n - z_n\| \leq e^{\kappa L t_n} \left(\|y_0 - z_0\| + \sum_{i=1}^{n} h_i \|(L_h y^h - L_h z^h)_i\| \right),\ 0 \leq n \leq N \tag{2.13}$$

with the constant $\kappa = 4$ for implicit methods. For explicit methods, the constant is $\kappa = 1$ and the condition on the step size can be omitted.

A proof can be found in [84, p. 52].

Definition 2.5 (Convergence)
A one step method is called convergent, if for any initial value problem (2.4)

$$\max_{t_n \in I} \|y_n - y(t_n)\| \to 0 \quad (h \to 0)\,,$$

where y is the solution of the initial value problem and $y^h = \big(y_n\big)_{0 \leq n \leq N}$ is the approximation computed by the one step method.

In case of one step methods, convergence is a direct consequence of consistency and discrete stability, as stated by the following theorem.

Theorem 2.10 (Convergence Theorem for One Step Methods)
Let F be a Lipschitz continuous one step method with Lipschitz constant L according to Definition 2.4, and let F be consistent with the initial value problem (2.4)*. If $\|y_0 - y(0)\| \to 0$, then the one step method is convergent*

$$\max_{t_n \in I} \|y_n - y(t_n)\| \to 0 \quad (h \to 0)\,,$$

and for sufficiently small step sizes $h < \frac{1}{2L}$ the a priori error estimate

$$\|y_n - y(t_n)\| \leq e^{4L t_n} \left(\|y_0 - y(0)\| + \sum_{i=1}^{n} h_i \|\tau_i^h\| \right),\ 0 \leq n \leq N\,,$$

holds. The condition on the step size can be omitted for explicit methods.

Proof. Since $L_h y^h = 0$ and $L_h y = \tau^h$, the claim follows directly from the Discrete Stability Theorem 2.9 [84, p. 54]. □

2.4 Operator splitting methods and composed one step schemes

Now we turn our attention back to the abstract multiphysics problem stated in (2.1). In this section, we derive the composed one step schemes which we propose for the time integration of the coupled models using the idea of operator splitting methods. The literature defines the term "splitting" or "splitting method" for a given initial value problem $\dot{y} = X(y), y(0) = y_0$ by means of the following three steps [89, p. 343]:

1. choosing a set of vector fields X_i such that $X = \sum_i X_i$;
2. integrating either exactly or approximately each X_i;
3. combining these solutions to yield an integrator for X.

The third step mentions the complement of splitting, namely combining the split parts to form a solution or approximation to the original problem. Combining the split parts is usually denoted a composition method in the literature. As an example, writing the exact solution by means of the flow $\varphi(t) = \exp(tX)$, the composition

$$\hat{\varphi}(t) = \exp(tX_1)\exp(tX_2)...\exp(tX_n) \tag{2.14}$$

is first order accurate, i.e. $\hat{\varphi}(t) = \varphi(t) + O(t^2)$.
The analysis of splitting methods is often studied by means of a classification of dynamical systems, in particular for flows of X which belong to some group of diffeomorphisms. Three main categories were identified [88]: dynamical systems, where the flow lies in a semi-group, in a symmetric space, or in a group. There is a rich literature on the analysis of splitting methods. In particular splittings have been studied which preserve certain properties of the system, like Hamiltonian systems [86, 10, 95], Poisson systems [85], volume-preserving systems [30, 66, 31], conformal volume-preserving systems, conformal Hamiltonian systems [65], contact systems [31], foliate systems with integrals [67, 46], general foliate systems [71], systems with symmetries [25], systems with reversing symmetries [58, 68], polynomial vector fields [19, 96], or trigonometric vector fields [82].
Widely known composition methods for flows $X = A + B$ include the first order Lie(-Trotter) method

$$\varphi_{\mathrm{LT}}(t) = \exp(tA)\exp(tB)\,,$$

which is the example (2.14), and the second order Strang method [100]

$$\varphi_{\mathrm{S}}(t) = \exp(\frac{t}{2}A)\exp(tB)\exp(\frac{t}{2}A)\,.$$

The prevalent composition methods present in the literature are derived from the general non-symmetric composition

$$\varphi = \mathrm{e}^{a_m tA}\mathrm{e}^{b_m tB}...\mathrm{e}^{a_1 tA}\mathrm{e}^{b_1 tB}\mathrm{e}^{a_0 tA}\,,$$

also called exponential operator splitting [48, 63], which is denoted an m-stage method with coefficients $a_0, ..., a_m, b_1, ..., b_m$. The choice of the number of stages and of the coefficients determines the convergence properties of the methods. Order conditions have been derived which ensure that methods have certain order p. The main methodologies for determining order conditions are direct methods based on [101] and [110], expansions by means of the Baker-Campbell-Hausdorff formula [39], extensions of the rooted tree approach known from the analysis of Runge-Kutta

methods [74], and symmetrized products of non-commutating operators [103].
A typical field of application for splitting methods is the numerical solution of partial differential equations, where the differential operators are split with respect to the spatial variables. Known splitting methods of this type include the alternating direction implicit (ADI) or Peaceman-Rachford scheme [79], which is a modification of the Crank-Nicolson scheme, and generalizations thereof like the Douglas-Rachford scheme [26], or fractional θ-schemes [21], which are based on the idea by Glowinski and Periaux [35].

2.4.1 Composed one step schemes

Analogously to splitting methods which are based on approximations to the flow $\varphi(t) = \exp(tX)$ of the vector field X as outlined above, we propose integration schemes for coupled models based on the numerical flow of one step methods. We use the following notation for numerical flows of one step methods.

Definition 2.6
For the initial value problem (2.4)*, let y_{n-1} be a given approximation to the solution at the time instant t_{n-1}, and let F be the defining function of a one step method as outlined in Sec. 2.3. We denote by Φ^F the numerical flow of the one step method F, i.e. the function that maps a tuple $(f, t_n, t_{n-1}, y_{n-1})$ to the approximation y_n at time instant t_n which results from the application of the one step method according to*

$$\Phi^F : (f, t_n, t_{n-1}, y_{n-1}) \mapsto y_n \quad \textit{such that} \quad y_n = y_{n-1} + h_n F(h_n, t_{n-1}, y_n, y_{n-1}) \; . \tag{2.15}$$

Remarks:

- For explicit one step methods, the numerical flow Φ^F is clearly well-defined, since computing the approximation for the next time step requires only function evaluations with known arguments from the previous time step.

- However, the application of an implicit one step method possibly requires to solve a nonlinear algebraic system of equations. In this case, Φ^F represents the implicitly defined function for y_n in (2.6), which is the system of equations resulting from the application of the method. Obviously, Φ^F is well-defined if and only if (2.6) is well-defined. This is surely the case if the problem data complies with the assumptions of the Implicit Function Theorem. In turn, this holds true if the system of equations resulting from the implicit method fulfills the assumptions of the Convergence Theorem for the Newton iteration, since these imply the assumptions from the Implicit Function Theorem. Precise statements of the Implicit Function Theorem and of the Convergence Theorem for the Newton iteration can be found in the Appendix of this chapter.

Let one step methods F and G with corresponding numerical flows Φ^F and Φ^G according to Definition 2.6 be given. Our idea is to address the initial value problem (2.1) by means of an operator splitting scheme, where F and G are used to treat the constituents f and g of the right-hand-side individually. We compose time stepping schemes for solving the coupled problem (2.1) while keeping the access of F to f separate from the access of G to g.

Definition 2.7 (Composed One Step Schemes)
Let one step methods F and G with corresponding numerical flows Φ^F and Φ^G according to Definition 2.6 be given. The composed one step schemes $\Phi^{F \circ G}$ and Φ^{F+G} for (2.1) *are defined*

by

$$\Phi^{F\circ G}(f,g,t_n,t_{n-1},y_{n-1}) := \Phi^F\Big(f,t_n,t_{n-1},\Phi^G(g,t_n,t_{n-1},y_{n-1})\Big)\,, \tag{2.16}$$

$$\Phi^{F+G}(f,g,t_n,t_{n-1},y_{n-1}) := \Phi^F(f,t_n,t_{n-1},y_{n-1}) + \Phi^G(g,t_n,t_{n-1},y_{n-1}) - y_{n-1}\,. \tag{2.17}$$

We call $\Phi^{F\circ G}$ a consecutive operator splitting scheme because Φ^F is applied to the result of Φ^G. In contrast, we call Φ^{F+G} a concurrent operator splitting scheme because Φ^F and Φ^G are applied independently and their individual results are combined to form the global result.

Remarks:

- Note that the consecutive scheme $\Phi^{F\circ G}$ represents the application of the Lie(-Trotter) splitting by consecutively executing the one step method G, then F.
- In contrast, the concurrent scheme Φ^{F+G} is different from the operator splitting schemes outlined above, which all share the consecutive nature.
- Both $\Phi^{F\circ G}$ and Φ^{F+G} obviously represent numerical flows of one step methods, since

$$y_n^{F\circ G} = \Phi^{F\circ G}(f,g,t_n,t_{n-1},y_{n-1}) \Longleftrightarrow \left.\begin{aligned} v_n &= y_{n-1} + h_n G(h_n,t_{n-1},v_n,y_{n-1}) \\ y_n^{F\circ G} &= y_{n-1} + h_n F(h_n,t_{n-1},y_n^{F\circ G},v_n) \end{aligned}\right\}\,,$$

  thus $y_n^{F\circ G} = y_{n-1} + h_n F\Big(h_n,t_{n-1},y_n,y_{n-1}+h_n G(h_n,t_{n-1},v_n,y_{n-1})\Big)$, and

$$y_n^{F+G} = \Phi^{F+G}(f,g,t_n,t_{n-1},y_{n-1}) \Longleftrightarrow \left.\begin{aligned} u_n &= y_{n-1} + h_n F(h_n,t_{n-1},u_n,y_{n-1}) \\ v_n &= y_{n-1} + h_n G(h_n,t_{n-1},v_n,y_{n-1}) \\ y_n^{F+G} &= u_n + v_n - y_{n-1} \end{aligned}\right\}\,,$$

  thus $y_n^{F+G} = y_{n-1} + h_n\Big[F(h_n,t_{n-1},u_n,y_{n-1}) + G(h_n,t_{n-1},v_n,y_{n-1})\Big]$.

We now prove theorems on the consistency, discrete stability and convergence of the composed one step schemes. We surely expect the consecutive scheme to be of order 1 since it resembles the Lie(-Trotter) splitting in our setup. However, our proofs emphasize the role of the one step methods used in the composition. Furthermore, we show that also the new concurrent scheme is of order 1.
For later reference, we state two families of initial value problems based on a solution y of the initial value problem (2.1). For $\nu \in [0,1]$, we define

$$\begin{aligned} \dot{u}^{(\nu)} &= f(t,u^{(\nu)}) \quad (\nu T \le t \le T)\,, \\ u^{(\nu)}(\nu T) &= y(\nu T)\,, \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} \dot{v}^{(\nu)} &= g(t,v^{(\nu)}) \quad (\nu T \le t \le T)\,, \\ v^{(\nu)}(\nu T) &= y(\nu T)\,. \end{aligned} \tag{2.19}$$

These families of initial value problems can be seen as attaching to any point on the trajectory y of the coupled system (2.1) trajectories of the separate models.

Theorem 2.11 (Consistency Theorem for the Consecutive Operator Splitting Scheme)
In the initial value problem (2.1)*, let f and g be continuous and satisfy a Lipschitz condition, and let $M_g := \sup_{\epsilon\in I, x\in\Omega} \|g(\epsilon, x)\| < \infty$. Let F be a consistent one step method for* (2.18) *and Lipschitz continuous with Lipschitz constant L_F. Moreover, let G be a one step method. Then the consecutive operator splitting scheme $\Phi^{F\circ G}$ as defined in* (2.16) *is consistent for* (2.1)*. Moreover, if F is consistent of order one or higher, then $\Phi^{F\circ G}$ is consistent of order one.*

Proof. Since f and g are assumed to be continuous and to satisfy a Lipschitz condition, according to Theorem 2.6 the initial value problems (2.1) and (2.19) have unique solutions y and $v^{(\nu)}$ for all $\nu \in [0, 1]$, respectively. The truncation error of $\Phi^{F\circ G}$ with respect to (2.1) is

$$\begin{aligned}\tau_n^h(F\circ G) &= \frac{1}{h_n}\Big[y(t_n) - y(t_{n-1})\Big] - F(h_n, t_{n-1}, y(t_n), v^{(\nu_{n-1})}(t_n)) \\ &= \tau_n^h(F) + F(h_n, t_{n-1}, y(t_n), y(t_{n-1})) - F(h_n, t_{n-1}, y(t_n), v^{(\nu_{n-1})}(t_n))\,,\end{aligned}$$

where $\tau_n^h(F)$ is the truncation error of F with respect to (2.18) for $\nu = \nu_{n-1} := t_{n-1}/T$. Using

$$v^{(\nu_{n-1})}(t_n) = y(t_{n-1}) + \int_{t_{n-1}}^{t_n} g(s, v^{(\nu_{n-1})}(s))\,ds\,,$$

due to the Lipschitz condition we have

$$\begin{aligned}\|\tau_n^h(F\circ G)\| &\le \|\tau_n^h(F)\| + \|F(h_n, t_{n-1}, y(t_n), y(t_{n-1})) - F(h_n, t_{n-1}, y(t_n), v^{(\nu_{n-1})}(t_n))\| \\ &\le \|\tau_n^h(F)\| + L_F\|y(t_{n-1}) - v^{(\nu_{n-1})}(t_n)\| \\ &\le \|\tau_n^h(F)\| + L_F\int_{t_{n-1}}^{t_n} \|g(s, v^{(\nu_{n-1})}(s))\|\,ds \\ &\le \|\tau_n^h(F)\| + h_n L_F M_g\,.\end{aligned}$$

Since F is assumed to be consistent for (2.18), the consecutive operator splitting scheme $\Phi^{F\circ G}$ is consistent for (2.1), and even consistent of order one if F is consistent of order one or higher. □

Theorem 2.12 (Consistency Theorem for the Concurrent Operator Splitting Scheme)
In the initial value problem (2.1)*, let f and g be continuous and satisfy a global Lipschitz condition with Lipschitz constants L_f and L_g, respectively. Moreover, let $M_f := \sup_{\epsilon\in I, x\in\Omega} \|f(\epsilon, x)\| < \infty$ and $M_g := \sup_{\epsilon\in I, x\in\Omega} \|g(\epsilon, x)\| < \infty$. Let F and G be consistent one step methods for* (2.18) *and* (2.19)*, respectively. Then the concurrent operator splitting scheme Φ^{F+G} as defined in* (2.17) *is consistent for* (2.1)*. Moreover, if both F and G are consistent of order one or higher, then Φ^{F+G} is consistent of order one.*

Proof. Since f and g are assumed to be continuous and to satisfy a global Lipschitz condition, according to Theorem 2.6 the initial value problems (2.1), (2.18) and (2.19) have unique solutions y, $u^{(\nu)}$ and $v^{(\nu)}$ for all $\nu \in [0, 1]$, respectively. Using

$$\begin{aligned}u^{(\nu_{n-1})}(t_n) &= y(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(s, u^{(\nu_{n-1})}(s))\,ds\,, \\ v^{(\nu_{n-1})}(t_n) &= y(t_{n-1}) + \int_{t_{n-1}}^{t_n} g(s, v^{(\nu_{n-1})}(s))\,ds\,, \\ y(t_n) &= y(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(s, y(s)) + g(s, y(s))\,ds\,,\end{aligned}$$

the truncation error of Φ^{F+G} with respect to (2.1) is

$$\begin{aligned}
\tau_n^h(F+G) &= \frac{1}{h_n}\Big[y(t_n) - y(t_{n-1})\Big] \\
&\quad - F(h_n, t_{n-1}, u^{(\nu_{n-1})}(t_n), y(t_{n-1})) - G(h_n, t_{n-1}, v^{(\nu_{n-1})}(t_n), y(t_{n-1})) \\
&= \frac{1}{h_n}\Big[u^{(\nu_{n-1})}(t_n) - y(t_{n-1})\Big] - F(h_n, t_{n-1}, u^{(\nu_{n-1})}(t_n), y(t_{n-1})) \\
&\quad + \frac{1}{h_n}\Big[v^{(\nu_{n-1})}(t_n) - y(t_{n-1})\Big] - G(h_n, t_{n-1}, v^{(\nu_{n-1})}(t_n), y(t_{n-1})) \\
&\quad + \frac{1}{h_n}\Big[y(t_n) + y(t_{n-1}) - u^{(\nu_{n-1})}(t_n) - v^{(\nu_{n-1})}(t_n)\Big] \\
&= \tau_n^h(F) + \tau_n^h(G) \\
&\quad + \frac{1}{h_n}\int_{t_{n-1}}^{t_n} f(s, y(s)) - f(s, u^{(\nu_{n-1})}(s)) + g(s, y(s)) - g(s, v^{(\nu_{n-1})}(s))\, ds\,,
\end{aligned}$$

where $\tau_n^h(F)$ and $\tau_n^h(G)$ are the truncation errors of F and G with respect to (2.18) and (2.19) for $\nu = \nu_{n-1} := t_{n-1}/T$, respectively. Due to the Lipschitz condition and owing to the Local Stability Theorem 2.5 we have

$$\begin{aligned}
&\Big\|\frac{1}{h_n}\int_{t_{n-1}}^{t_n} f(s, y(s)) - f(s, u^{(\nu_{n-1})}(s)) + g(s, y(s)) - g(s, v^{(\nu_{n-1})}(s))\, ds\Big\| \\
&\quad \le \frac{L_f}{h_n}\int_{t_{n-1}}^{t_n} \|y(s) - u^{(\nu_{n-1})}(s)\|\, ds + \frac{L_g}{h_n}\int_{t_{n-1}}^{t_n} \|y(s) - v^{(\nu_{n-1})}(s)\|\, ds \\
&\quad \le \frac{L_f e^{L_f h_n}}{h_n}\int_{t_{n-1}}^{t_n}\int_{t_{n-1}}^{s} \sup_{x\in\Omega} \|g(\epsilon, x)\|\, d\epsilon ds + \frac{L_g e^{L_g h_n}}{h_n}\int_{t_{n-1}}^{t_n}\int_{t_{n-1}}^{s} \sup_{x\in\Omega} \|f(\epsilon, x)\|\, d\epsilon ds \\
&\quad \le L_f h_n e^{L_f h_n} M_g + L_g h_n e^{L_g h_n} M_f = O(h_n)\,.
\end{aligned}$$

Thus we have $\|\tau_n^h(F+G)\| \le \|\tau_n^h(F)\| + \|\tau_n^h(G)\| + O(h_n)$. Since by assumption F is consistent for (2.18) and G is consistent for (2.19), the concurrent operator splitting scheme Φ^{F+G} is consistent for (2.1), and even consistent of order one if both F and G are consistent of order one or higher. □

Theorem 2.13 (Discrete Stability Theorem for the Consecutive Operator Splitting Scheme) *Let F and G be Lipschitz continuous one step methods according to Definition 2.4 with Lipschitz constants L_F and L_G, respectively, and $L := \max\{L_F, L_G\}$. Let $y^h = (y_n)_{0\le n\le N}$, $z^h = (y_n)_{0\le n\le N}$, $u^h = (u_n)_{0\le n\le N}$ and $v^h = (v_n)_{0\le n\le N}$ be grid functions. Then there exist constants κ, λ, μ and η such that for $h < \frac{1}{2L}$*

$$\begin{aligned}
\|y_n - z_n\| \le \exp\left(\kappa L_F t_n + \lambda L_F L_G \sum_{i=1}^{n} h_i^2\right)\Bigg\{\mu\|y_0 - z_0\| + \sum_{i=1}^{n} h_i \|(L_h^{F\circ G} y^h - L_h^{F\circ G} z^h)_i\| \\
+ \eta L_F \sum_{i=1}^{n} h_i^2 \|(L_h^G u^h - L_h^G v^h)_i\|\Bigg\}\,,
\end{aligned} \tag{2.20}$$

where, according to (2.7), *L_h^G is the difference operator of G, and $L_h^{F\circ G}$ is the difference operator of $\Phi^{F\circ G}$ as defined in* (2.16). *Depending on whether F and G are explicit or implicit one step*

methods, the constants κ, λ, μ and η are given in Table 2.1. In case if both F and G are explicit one step methods, the step size restriction can be omitted.

	G explicit	G implicit
F explicit	$\kappa = 1, \lambda = 1, \mu = 1, \eta = 1$	$\kappa = 2, \lambda = 2, \mu = 1, \eta = 2$
F implicit	$\kappa = 4, \lambda = 2, \mu = 2, \eta = 1$	$\kappa = 8, \lambda = 0, \mu = 2, \eta = 2$

Table 2.1: Values of the constants κ, λ and μ in the Theorems 2.13 and 2.15.

Proof. This proof follows [84, p. 52]. Applying the difference operators L_h^G and $L_h^{F\circ G}$ to the grid functions yields

$$(L_h^G u^h)_n = \frac{1}{h_n}(u_n - y_{n-1}) - G(h_n, t_{n-1}, u_n, y_{n-1})\,, \tag{2.21}$$

$$(L_h^G v^h)_n = \frac{1}{h_n}(v_n - z_{n-1}) - G(h_n, t_{n-1}, v_n, z_{n-1})\,, \tag{2.22}$$

$$(L_h^{F\circ G} y^h)_n = \frac{1}{h_n}(y_n - y_{n-1}) - F(h_n, t_{n-1}, y_n, u_n)\,, \tag{2.23}$$

$$(L_h^{F\circ G} z^h)_n = \frac{1}{h_n}(z_n - z_{n-1}) - F(h_n, t_{n-1}, z_n, v_n)\,. \tag{2.24}$$

Case 1: F is an explicit one step method. Subtraction of (2.23) and (2.24) yields

$$\begin{aligned}\|e_n\| &\leq \|e_{n-1}\| + h_n\|F(h_n, t_{n-1}, y_n, u_n) - F(h_n, t_{n-1}, z_n, v_n)\| + h_n\|\epsilon_n^{F\circ G}\| \\ &\leq \|e_{n-1}\| + h_n L_F\|u_n - v_n\| + h_n\|\epsilon_n^{F\circ G}\|\,,\end{aligned} \tag{2.25}$$

where $e_n := y_n - z_n$ and $\epsilon_n^{F\circ G} := (L_h^{F\circ G} y^h - L_h^{F\circ G} z^h)_n$.

Case 1a: G is also an explicit one step method. Then subtraction of (2.21) and (2.22) yields

$$\begin{aligned}\|u_n - v_n\| &\leq \|y_{n-1} - z_{n-1}\| + h_n\|G(h_n, t_{n-1}, u_n, y_{n-1}) - G(h_n, t_{n-1}, v_n, z_{n-1})\| + h_n\|\epsilon_n^G\| \\ &\leq \|e_{n-1}\| + h_n L_G\|e_{n-1}\| + h_n\|\epsilon_n^G\|\,,\end{aligned} \tag{2.26}$$

where $\epsilon_n^G := (L_h^G u^h - L_h^G v^h)_n$. Using (2.26) in (2.25) yields

$$\|e_n\| \leq \|e_{n-1}\| + h_n L_F(1 + h_n L_G)\|e_{n-1}\| + h_n\|\epsilon_n^{F\circ G}\| + h_n^2 L_F\|\epsilon_n^G\|\,.$$

Applying this formalism recursively we get

$$\|e_n\| \leq \|e_0\| + \sum_{i=0}^{n-1} h_{i+1} L_F(1 + h_{i+1} L_G)\|e_i\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F\circ G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\,.$$

The Discrete Gronwall Lemma yields

$$\|e_n\| \leq \exp\left(L_F t_n + L_F L_G \sum_{i=1}^{n} h_i^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F\circ G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\},$$

which holds without restriction on the time step size h.

Case 1b: G is an implicit one step method. Then subtraction of (2.21) and (2.22) yields

$$\begin{aligned}\|u_n - v_n\| &\leq \|y_{n-1} - z_{n-1}\| + h_n\|G(h_n, t_{n-1}, u_n, y_{n-1}) - G(h_n, t_{n-1}, v_n, z_{n-1})\| + h_n\|\epsilon_n^G\| \\ &\leq \|e_{n-1}\| + h_n L_G\Big(\|u_n - v_n\| + \|e_{n-1}\|\Big) + h_n\|\epsilon_n^G\|\,,\end{aligned}$$

and for $h < \frac{1}{2L}$

$$\|u_n - v_n\| \leq \frac{1 + h_n L_G}{1 - h_n L_G}\|e_{n-1}\| + \frac{h_n}{1 - h_n L_G}\|\epsilon_n^G\|\,. \tag{2.27}$$

Using (2.27) in (2.25) yields

$$\|e_n\| \leq \|e_{n-1}\| + h_n L_F \frac{1 + h_n L_G}{1 - h_n L_G}\|e_{n-1}\| + h_n\|\epsilon_n^{F\circ G}\| + \frac{h_n^2 L_F}{1 - h_n L_G}\|\epsilon_n^G\|\,.$$

Applying this formalism recursively we get

$$\|e_n\| \leq \|e_0\| + \sum_{i=0}^{n-1} h_{i+1} L_F \frac{1 + h_{i+1} L_G}{1 - h_{i+1} L_G}\|e_i\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F\circ G}\| + \sum_{i=1}^{n} \frac{h_i^2 L_F}{1 - h_i L_G}\|\epsilon_i^G\|\,.$$

The Discrete Gronwall Lemma yields with $\frac{1}{1-hL_G} < 2$

$$\|e_n\| \leq \exp\left(2L_F t_n + 2L_F L_G \sum_{i=1}^{n} h_i^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F\circ G}\| + 2L_F \sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\}.$$

Case 2: F is an implicit one step method. Subtraction of (2.23) and (2.24) yields

$$\begin{aligned}\|e_n\| &\leq \|e_{n-1}\| + h_n\|F(h_n, t_{n-1}, y_n, u_n) - F(h_n, t_{n-1}, z_n, v_n)\| + h_n\|\epsilon_n^{F\circ G}\| \\ &\leq \|e_{n-1}\| + h_n L_F\Big(\|e_n\| + \|u_n - v_n\|\Big) + h_n\|\epsilon_n^{F\circ G}\|\,.\end{aligned} \tag{2.28}$$

Case 2a: G is an explicit method. Using (2.26) in (2.28) yields

$$\|e_n\| \leq \|e_{n-1}\| + h_n L_F\Big(\|e_n\| + \|e_{n-1}\| + h_n L_G\|e_{n-1}\| + h_n\|\epsilon_n^G\|\Big) + h_n\|\epsilon_n^{F\circ G}\|\,.$$

With $h < \frac{1}{2L}$ and $h_0 := 0$ we get

$$\begin{aligned}(1 - h_n L_F)\|e_n\| &\leq (1 + h_n L_F + h_n^2 L_F L_G)\|e_{n-1}\| + h_n\|\epsilon_n^{F\circ G}\| + h_n^2 L_F\|\epsilon_n^G\| \\ &= (1 - h_{n-1} L_F)\|e_{n-1}\| + \frac{(h_n + h_{n-1})L_F + h_n^2 L_F L_G}{1 - h_{n-1} L_F}(1 - h_{n-1} L_F)\|e_{n-1}\| \\ &\quad + h_n\|\epsilon_n^{F\circ G}\| + h_n^2 L_F\|\epsilon_n^G\|\,,\end{aligned}$$

and setting $w := (1 - h_n L_F)e_n$ results in

$$\|w_n\| \leq \|w_{n-1}\| + \frac{(h_n + h_{n-1})L_F + h_n^2 L_F L_G}{1 - h_{n-1} L_F}\|w_{n-1}\| + h_n\|\epsilon_n^{F\circ G}\| + h_n^2 L_F\|\epsilon_n^G\|\,.$$

Applying this formalism recursively we get

$$\|w_n\| \leq \|w_0\| + \sum_{i=0}^{n-1} \frac{(h_{i+1}+h_i)L_F + h_{i+1}^2 L_F L_G}{1-h_i L_F}\|w_i\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\,.$$

The Discrete Gronwall Lemma yields

$$\|e_n\| \leq \frac{1}{1-h_n L_F} \exp\left(\sum_{i=0}^{n-1} \frac{(h_{i+1}+h_i)L_F + h_{i+1}^2 L_F L_G}{1-h_i L_F}\right)\left\{\|w_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\right\}.$$

For $0 < h < \frac{1}{2L}$ we have $\frac{1}{1-hL_F} \leq e^{\frac{hL_F}{1-hL_F}}$ and $\frac{1}{1-hL_F} < 2$, therefore

$$\begin{aligned}
\|e_n\| &\leq \exp\left(\frac{h_n L_F}{1-h_n L_F}\right) \exp\left(\sum_{i=0}^{n-1} \frac{(h_{i+1}+h_i)L_F + h_{i+1}^2 L_F L_G}{1-h_i L_F}\right) \\
&\quad \times \left\{2\|e_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\right\} \\
&\leq \exp\left(4L_F t_n + 2L_F L_G \sum_{i=1}^{n} h_i^2\right)\left\{2\|e_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\right\}.
\end{aligned}$$

Case 2b: G is also an implicit method. Using (2.27) in (2.28) yields

$$\|e_n\| \leq \|e_{n-1}\| + h_n L_F \left(\|e_n\| + \frac{1+h_n L_G}{1-h_n L_G}\|e_{n-1}\| + \frac{h_n}{1-h_n L_G}\|\epsilon_n^G\|\right) + h_n \|\epsilon_n^{F\circ G}\|\,.$$

With $h < \frac{1}{2L}$ and $h_0 := 0$ we get

$$\begin{aligned}
(1-h_n L_F)\|e_n\| &\leq \left(1 + h_n L_F \frac{1+h_n L_G}{1-h_n L_G}\right)\|e_{n-1}\| + h_n \|\epsilon_n^{F\circ G}\| + \frac{h_n^2 L_F}{1-h_n L_G}\|\epsilon_n^G\| \\
&= (1-h_{n-1}L_F)\|e_{n-1}\| + \frac{h_{n-1}L_F + h_n L_F \frac{1+h_n L_G}{1-h_n L_G}}{1-h_{n-1}L_F}(1-h_{n-1}L_F)\|e_{n-1}\| \\
&\quad + h_n \|\epsilon_n^{F\circ G}\| + \frac{h_n^2 L_F}{1-h_n L_G}\|\epsilon_n^G\|\,,
\end{aligned}$$

and setting $w := (1-h_n L_F)e_n$ results in

$$\|w_n\| \leq \|w_{n-1}\| + \frac{h_{n-1}L_F + h_n L_F \frac{1+h_n L_G}{1-h_n L_G}}{1-h_{n-1}L_F}\|w_{n-1}\| + h_n \|\epsilon_n^{F\circ G}\| + \frac{h_n^2 L_F}{1-h_n L_G}\|\epsilon_n^G\|\,.$$

Applying this formalism recursively we get

$$\|w_n\| \leq \|w_0\| + \sum_{i=0}^{n-1} \frac{h_i L_F + h_{i+1} L_F \frac{1+h_{i+1}L_G}{1-h_{i+1}L_G}}{1-h_i L_F}\|w_i\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + \sum_{i=1}^{n} \frac{h_i^2 L_F}{1-h_i L_G}\|\epsilon_i^G\|\,.$$

The Discrete Gronwall Lemma yields

$$\|e_n\| \leq \frac{1}{1-h_n L_F} \exp\left(\sum_{i=0}^{n-1} \frac{h_i L_F + h_{i+1} L_F \frac{1+h_{i+1}L_G}{1-h_{i+1}L_G}}{1-h_i L_F}\right)\left\{\|w_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + \sum_{i=1}^{n} \frac{h_i^2 L_F}{1-h_i L_G}\|\epsilon_i^G\|\right\}.$$

For $0 < h < \frac{1}{2L}$ we have $\frac{1}{1-hL_F} \leq e^{\frac{hL_F}{1-hL_F}}$, $\frac{1}{1-hL_F} < 2$, $\frac{1}{1-hL_G} < 2$ and $\frac{1+hL_G}{1-hL_G} < 3$, therefore

$$\begin{aligned}\|e_n\| &\leq \exp\left(\frac{h_n L_F}{1-h_n L_F}\right) \exp\left(\sum_{i=0}^{n-1} \frac{h_i L_F + h_{i+1} L_F \frac{1+h_{i+1}L_G}{1-h_{i+1}L_G}}{1-h_i L_F}\right) \\ &\quad \times \left\{2\|e_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + 2L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\right\} \\ &\leq \exp\left(8L_F t_n\right)\left\{2\|e_0\| + \sum_{i=1}^{n} h_i \|\epsilon_i^{F\circ G}\| + 2L_F \sum_{i=1}^{n} h_i^2 \|\epsilon_i^G\|\right\}.\end{aligned}$$

□

Theorem 2.14 (Discrete Stability Theorem for the Concurrent Operator Splitting Scheme) *Let F and G be Lipschitz continuous one step methods according to Definition 2.4 with Lipschitz constants L_F and L_G, respectively, and $L := \max\{L_F, L_G\}$. Let $y^h = (y_n)_{0\leq n\leq N}$, $z^h = (y_n)_{0\leq n\leq N}$, $u_y^h = (u_{y,n})_{0\leq n\leq N}$, $u_z^h = (u_{z,n})_{0\leq n\leq N}$, $v_y^h = (v_{y,n})_{0\leq n\leq N}$ and $v_z^h = (v_{z,n})_{0\leq n\leq N}$ be grid functions. Then there exist constants κ, λ, μ and η such that for $h < \frac{1}{2L}$*

$$\begin{aligned}\|y_n - z_n\| \leq \exp\left(\kappa L t_n + \lambda L^2 \sum_{i=1}^{n} h_i^2\right)\Bigg\{&\|y_0 - z_0\| + \sum_{i=1}^{n} h_i \|(L_h^{F+G} y^h - L_h^{F+G} z^h)_i\| \\ &+ \mu L_F \sum_{i=1}^{n} h_i^2 \|L_h^F u_y^h - L_h^F u_z^h)_j\| \\ &+ \eta L_G \sum_{i=1}^{n} h_i^2 \|L_h^G v_y^h - L_h^G v_z^h)_j\|\Bigg\},\end{aligned} \tag{2.29}$$

where, according to (2.7)*, L_h^F and L_h^G are the difference operators of F and G, respectively, and L_h^{F+G} is the difference operator of Φ^{F+G} as defined in* (2.17)*. Depending on whether F and G are explicit or implicit one step methods, the constants κ, λ, μ and η are given in Table 2.2. In case if both F and G are explicit one step methods, the step size restriction can be omitted.*

	G explicit	G implicit
F explicit	$\kappa = 4, \lambda = 2, \mu = 1, \eta = 1$	$\kappa = 5, \lambda = 3, \mu = 1, \eta = 2$
F implicit	$\kappa = 5, \lambda = 3, \mu = 2, \eta = 1$	$\kappa = 6, \lambda = 4, \mu = 2, \eta = 2$

Table 2.2: Values of the constants κ, λ, μ and η in the Theorems 2.14 and 2.16.

Proof. This proof follows [84, p. 52]. Applying the difference operatorsL_h^F, L_h^G and L_h^{F+G} to the

grid functions we have

$$(L_h^F u_y^h)_n = \frac{1}{h_n}(u_{y,n} - y_{n-1}) - F(h_n, t_{n-1}, u_{y,n}, y_{n-1})\,, \tag{2.30}$$

$$(L_h^F u_z^h)_n = \frac{1}{h_n}(u_{z,n} - z_{n-1}) - F(h_n, t_{n-1}, u_{z,n}, z_{n-1})\,, \tag{2.31}$$

$$(L_h^G v_y^h)_n = \frac{1}{h_n}(v_{y,n} - y_{n-1}) - F(h_n, t_{n-1}, v_{y,n}, y_{n-1})\,, \tag{2.32}$$

$$(L_h^G v_z^h)_n = \frac{1}{h_n}(v_{z,n} - z_{n-1}) - F(h_n, t_{n-1}, v_{z,n}, z_{n-1})\,, \tag{2.33}$$

$$(L_h^{F+G} y^h)_n = \frac{1}{h_n}(y_n - y_{n-1}) - F(h_n, t_{n-1}, u_{y,n}, y_{n-1}) - G(h_n, t_{n-1}, v_{y,n}, y_{n-1})\,, \tag{2.34}$$

$$(L_h^{F+G} z^h)_n = \frac{1}{h_n}(z_n - z_{n-1}) - F(h_n, t_{n-1}, u_{z,n}, z_{n-1}) - G(h_n, t_{n-1}, v_{z,n}, z_{n-1})\,. \tag{2.35}$$

Both F and G are assumed to be Lipschitz continuous, therefore

$$\begin{aligned}
&\Big\| F(h_n, t_{n-1}, u_{y,n}, y_{n-1}) - F(h_n, t_{n-1}, u_{z,n}, z_{n-1}) \\
&+ G(h_n, t_{n-1}, v_{y,n}, y_{n-1}) - G(h_n, t_{n-1}, v_{z,n}, z_{n-1}) \Big\| \\
&\le (L_F + L_G)\|y_{n-1} - z_{n-1}\| + L_F\|u_{y,n} - u_{z,n}\| + L_G\|v_{y,n} - v_{z,n}\| \\
&\le 2L\|e_{n-1}\| + L_F\|e_n^u\| + L_G\|e_n^v\|\,,
\end{aligned}$$

where $e_n := y_n - z_n$, $e_n^u := u_{y,n} - u_{z,n}$ and $e_n^v := v_{y,n} - v_{z,n}$. Subtraction of (2.34) and (2.35) yields

$$\|e_n\| \le \|e_{n-1}\| + 2h_nL\|e_{n-1}\| + h_nL_F\|e_n^u\| + h_nL_G\|e_n^v\| + h_n\|\epsilon_n^{F+G}\|\,, \tag{2.36}$$

where $\epsilon_n^{F+G} := (L_h^{F+G} y^h - L_h^{F+G} z^h)_n$.

Case 1: Both F and G are explicit one step methods. Subtraction of (2.30) and (2.31), and subtraction of (2.32) and (2.33) yield

$$\|e_n^u\| \le (1 + h_nL)\|e_{n-1}\| + h_n\|\epsilon_n^F\|\,, \tag{2.37}$$

$$\|e_n^v\| \le (1 + h_nL)\|e_{n-1}\| + h_n\|\epsilon_n^G\|\,, \tag{2.38}$$

where $\epsilon_n^F := (L_h^F u_y^h - L_h^F u_z^h)_n$ and $\epsilon_n^G := (L_h^F v_y^h - L_h^F v_z^h)_n$. Using (2.37) and (2.38) in (2.36) we get

$$\|e_n\| \le \|e_{n-1}\| + (4h_nL + 2h_n^2L^2)\|e_{n-1}\| + h_n\|\epsilon_n^{F+G}\| + h_n^2L_F\|\epsilon_n^F\| + h_n^2L_G\|\epsilon_n^G\|\,.$$

Applying this formalism recursively we get

$$\|e_n\| \le \|e_0\| + \sum_{i=0}^{n-1}(4h_{i+1}L + 2h_{i+1}^2L^2)\|e_i\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\,.$$

Using the Discrete Gronwall Lemma we infer

$$\begin{aligned}
\|e_n\| &\le \exp\left(\sum_{i=0}^{n-1} 4h_{i+1}L + 2h_{i+1}^2L^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\} \\
&= \exp\left(4Lt_n + 2L^2\sum_{i=1}^{n} h_i^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\},
\end{aligned}$$

which holds without restriction on the time step size.

Case 2: Exactly one of the methods F and G is implicit, and the other method is explicit. Since the concurrent operator splitting scheme is symmetric with respect to F and G, it is sufficient to prove only the case if is F explicit and G is implicit. Then subtraction of (2.32) and (2.33) yields

$$\|e_n^v\| \leq \|e_{n-1}\| + h_n L(\|e_n^v\| + \|e_{n-1}\|) + h_n\|\epsilon_n^G\|\,.$$

With $h < \frac{1}{2L}$ and $h_0 := 0$ we get

$$\|e_n^v\| \leq \frac{1+h_nL}{1-h_nL}\|e_{n-1}\| + \frac{h_n}{1-h_nL}\|\epsilon_n^G\|\,. \tag{2.39}$$

Using (2.37) and (2.39) in (2.36) we get

$$\|e_n\| \leq \|e_{n-1}\| + \Big(3h_nL + h_nL\frac{1+h_nL}{1-h_nL} + h_n^2L^2\Big)\|e_{n-1}\| + h_n\|\epsilon_n^{F+G}\| + h_n^2L_F\|\epsilon_n^F\| + \frac{h_n^2L_G}{1-h_nL}\|\epsilon_n^G\|\,.$$

Applying this formalism recursively and using $\frac{1}{1-hL} < 2$ we get

$$\|e_n\| \leq \|e_0\| + \sum_{i=0}^{n-1}(5h_{i+1}L + 3h_{i+1}^2L^2)\|e_i\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\,.$$

Using the Discrete Gronwall Lemma we infer

$$\begin{aligned}\|e_n\| &\leq \exp\left(\sum_{i=0}^{n-1} 5h_{i+1}L + 3h_{i+1}^2L^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\}\\ &= \exp\left(5Lt_n + 3L^2\sum_{i=1}^{n} h_i^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\}.\end{aligned}$$

Case 3: Both F and G are implicit one step methods. Then subtraction of (2.30) and (2.31) yields

$$\|e_n^u\| \leq \|e_{n-1}\| + h_n L(\|e_n^u\| + \|e_{n-1}\|) + h_n\|\epsilon_n^F\|\,.$$

With $h < \frac{1}{2L}$ and $h_0 := 0$ we get

$$\|e_n^u\| \leq \frac{1+h_nL}{1-h_nL}\|e_{n-1}\| + \frac{h_n}{1-h_nL}\|\epsilon_n^F\|\,. \tag{2.40}$$

Using (2.40) and (2.39) in (2.36) we get

$$\|e_n\| \leq \|e_{n-1}\| + \Big(2h_nL + 2h_nL\frac{1+h_nL}{1-h_nL}\Big)\|e_{n-1}\| + h_n\|\epsilon_n^{F+G}\| + \frac{h_n^2L_F}{1-h_nL}\|\epsilon_n^F\| + \frac{h_n^2L_G}{1-h_nL}\|\epsilon_n^G\|\,.$$

Applying this formalism recursively and using $\frac{1}{1-hL} < 2$ we get

$$\|e_n\| \leq \|e_0\| + \sum_{i=0}^{n-1}(6h_{i+1}L + 4h_{i+1}^2L^2)\|e_i\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + 2L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\,.$$

Using the Discrete Gronwall Lemma we infer

$$\begin{aligned}\|e_n\| &\leq \exp\left(\sum_{i=0}^{n-1} 6h_{i+1}L + 4h_{i+1}^2L^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + 2L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\} \\ &= \exp\left(6Lt_n + 4L^2\sum_{i=1}^{n} h_i^2\right)\left\{\|e_0\| + \sum_{i=1}^{n} h_i\|\epsilon_i^{F+G}\| + 2L_F\sum_{i=1}^{n} h_i^2\|\epsilon_i^F\| + 2L_G\sum_{i=1}^{n} h_i^2\|\epsilon_i^G\|\right\}.\end{aligned}$$

□

Theorem 2.15 (Convergence Theorem for the Consecutive Operator Splitting Scheme)
In the initial value problem (2.1)*, let f and g be continuous and satisfy a Lipschitz condition, and let $M_g = \sup_{\epsilon\in I, x\in\Omega}\|g(\epsilon, x)\| < \infty$. Let F and G be consistent and Lipschitz continuous one step methods according to Definition 2.4 with Lipschitz constants L_F and L_G, respectively. Let y be the solution of* (2.1)*, and $y^h = (y_n)_{0\leq n\leq N}$ be the discrete solution computed by means of the consecutive operator splitting scheme $\Phi^{F\circ G}$ as defined in* (2.16)*. Moreover, let v be the solution of* (2.19) *with $\nu = 0$ and $v^h = (v_n)_{0\leq n\leq N}$ the approximation computed by means of G. If $\|y_0 - y(0)\| \to 0$ and $\|v_0 - v(0)\| \to 0$, then the consecutive operator splitting scheme is convergent, and there exist constants κ, λ and μ such that for sufficiently small h*

$$\begin{aligned}\|y_n - y(t_n)\| \leq \exp\left(\kappa L_F t_n + \lambda L_F L_G\sum_{i=1}^{n} h_i^2\right)\Bigg\{&\mu\|y_0 - y(0)\| + \sum_{i=1}^{n} h_i\|\tau_i^{F\circ G}\| \\ &+ \eta L_F\sum_{i=1}^{n} h_i^2\|\tau_i^G\|\Bigg\},\end{aligned} \tag{2.41}$$

where $\tau^{F+G} = L_h^{F\circ G}y$ and $\tau^G = L_h^G v$. Depending on whether F and G are explicit or implicit one step methods, the constants κ, λ, μ and η are given in Table 2.1.

Proof. We have $L_h^{F\circ G}y^h = 0$ and $L_h^G v^h = 0$. Theorem 2.13 holds by the assumptions, therefore using $z^h = y$ and $u^h = v$ directly yields (2.41). Also Thm. 2.11 holds by the assumptions, therefore $\Phi^{F\circ G}$ is consistent and thus $\|\tau_n^{F\circ G}\| \to 0\,(h \to 0)$ for $n = 1, ..., N$. Furthermore, since G is itself assumed to be consistent, we have $\|\tau_n^G\| \to 0\,(h \to 0)$ for $n = 1, ..., N$. This completes the proof of the convergence of $\Phi^{F\circ G}$. □

Theorem 2.16 (Convergence Theorem for the Concurrent Operator Splitting Scheme)
In the initial value problem (2.1)*, let f and g be continuous and satisfy a global Lipschitz condition with Lipschitz constants L_f and L_g, respectively. Moreover, let $M_f = \sup_{\epsilon\in I, x\in\Omega}\|f(\epsilon, x)\| < \infty$ and $M_g = \sup_{\epsilon\in I, x\in\Omega}\|g(\epsilon, x)\| < \infty$. Let F and G be consistent and Lipschitz continuous one step methods according to Definition 2.4 with Lipschitz constants L_F and L_G, respectively, and $L = \max\{L_F, L_G\}$. Let y be the solution of* (2.1)*, and $y^h = (y_n)_{0\leq n\leq N}$ be the discrete solution computed by means of the concurrent operator splitting scheme Φ^{F+G} as defined in* (2.17)*. Moreover, let u be the solution of* (2.18) *with $\nu = 0$ and $u^h = (u_n)_{0\leq n\leq N}$ the approximation computed by means of F, and v be the solution of* (2.18) *with $\nu = 0$ and $v^h = (v_n)_{0\leq n\leq N}$ the approximation computed by means of G. If $\|y_0 - y(0)\| \to 0$, $\|u_0 - u(0)\| \to 0$ and $\|v_0 - v(0)\| \to 0$, then the concurrent operator splitting scheme is convergent, and there exist constants κ, λ, μ and η such*

that for sufficiently small h

$$\|y_n - y(t_n)\| \leq \exp\left(\kappa L t_n + \lambda L^2 \sum_{i=1}^{n} h_i^2\right)\Bigg\{\|y_0 - y(0)\| + \sum_{i=1}^{n} h_i \|\tau_i^{F+G}\| + \mu L_F \sum_{i=1}^{n} h_i^2 \|\tau_i^F\| + \eta L_G \sum_{i=1}^{n} h_i^2 \|\tau_i^G\|\Bigg\}, \tag{2.42}$$

where $\tau^{F+G} = L_h^{F+G} y$, $\tau^F = L_h^F u$ *and* $\tau^G = L_h^G v$. *Depending on whether* F *and* G *are explicit or implicit one step methods, the constants* κ, λ, μ *and* η *are given in Table 2.2.*

Proof. We have $L_h^{F+G} y^h = 0$, $L_h^F u^h = 0$ and $L_h^G v^h = 0$. Theorem 2.14 holds by the assumptions, therefore using $z^h = y$, $u_y^h = u^h$, $u_z^h = u$, $v_y^h = v^h$ and $v_z^h = v$ directly yields (2.42). Also Theorem 2.12 holds by the assumptions, therefore Φ^{F+G} is consistent and thus $\|\tau_n^{F+G}\| \to 0\,(h \to 0)$ for $n = 1, ..., N$. Furthermore, since F and G are themselves assumed to be consistent, we have $\|\tau_n^F\| \to 0\,(h \to 0)$ and $\|\tau_n^G\| \to 0\,(h \to 0)$ for $n = 1, ..., N$. This completes the proof of the convergence of Φ^{F+G}. □

Remarks:

- The definition of the composed one step schemes can easily be generalized to treat initial value problems

$$\dot{y} = \sum_{i=1}^{k} f_i(t, y) \quad , \quad y(0) = y_0$$

with any number $k \geq 1$ of constituents. Given one step methods $F_1, ..., F_k$, the consecutive and the concurrent operator splitting schemes read

$$\Phi^{F_1 \circ ... \circ F_k}(f_1, ..., f_k, t_n, t_{n-1}, y_{n-1}) = \Phi^{F_1}(f_1, t_n, t_{n-1}, ...\Phi^{F_k}(f_k, t_n, t_{n-1}, y_{n-1})...)\,,$$

$$\Phi^{F_1 + ... + F_k}(f_1, ..., f_k, t_n, t_{n-1}, y_{n-1}) = \sum_{i=1}^{k} \Phi^{F_i}(f_i, t_n, t_{n-1}, y_{n-1}) - (k-1) y_{n-1}\,.$$

Also our theorems on consistency, discrete stability and convergence of the composed one step methods generalize accordingly. Additional summands appear in the proofs of Theorems 2.11 and 2.12, and in the estimates (2.20), (2.29), (2.41) and (2.42). For the consecutive operator splitting scheme, higher order terms up to h^k appear, while for the concurrent operator splitting all additional terms are of order 2. Nevertheless, the statements of the theorems hold unchanged.

- As already mentioned in the beginning of this chapter, also systems of the form

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} f(t, x, y) \\ g(t, x, y) \end{pmatrix} \tag{2.43}$$

can be treated with our composed one step schemes. Let one step methods F for f, G for g and corresponding numerical flows Φ^F, Φ^G be given. We define

$$\hat{f}(t, x, y) = \begin{pmatrix} f(t, x, y) \\ 0 \end{pmatrix} \quad , \quad \hat{g}(t, x, y) = \begin{pmatrix} 0 \\ g(t, x, y) \end{pmatrix}$$

and numerical flows

$$\hat{\Phi}^F(\hat{f}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) = \begin{pmatrix} \Phi^F(f, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \\ y_{n-1} \end{pmatrix},$$
$$\hat{\Phi}^G(\hat{g}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) = \begin{pmatrix} x_{n-1} \\ \Phi^G(g, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \end{pmatrix}.$$

The consecutive operator splitting scheme then reads

$$\begin{aligned} \begin{pmatrix} x_n \\ y_n \end{pmatrix} &= \hat{\Phi}^{F\circ G}(\hat{f}, \hat{g}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \\ &= \hat{\Phi}^F(\hat{f}, t_n, t_{n-1}, x_{n-1}, \hat{\Phi}^G(\hat{g}, t_n, t_{n-1}, x_{n-1}, y_{n-1})) \\ &= \begin{pmatrix} \Phi^F(f, t_n, t_{n-1}, x_{n-1}, \Phi^G(g, t_n, t_{n-1}, x_{n-1}, y_{n-1})) \\ \Phi^G(g, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \end{pmatrix}, \end{aligned} \tag{2.44}$$

and the concurrent operator splitting scheme reads

$$\begin{aligned} \begin{pmatrix} x_n \\ y_n \end{pmatrix} &= \hat{\Phi}^{F+G}(\hat{f}, \hat{g}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \\ &= \hat{\Phi}^F(\hat{f}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) + \hat{\Phi}^G(\hat{g}, t_n, t_{n-1}, x_{n-1}, y_{n-1}) - \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} \Phi^F(f, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \\ \Phi^G(g, t_n, t_{n-1}, x_{n-1}, y_{n-1}) \end{pmatrix}. \end{aligned} \tag{2.45}$$

Theorems 2.12-2.16 hold unchanged for (2.43). Only in Theorem 2.11 an assumption needs to be added that, in addition to the method F, also G must be consistent and Lipschitz continuous.

- We did not investigate if higher order composed one step methods are possible since we are using a first order scheme for the biogeochemical model in the nutrient cycle simulations. Although we did not investigate if our proof technique can be generalized, we nevertheless assume that higher order consecutive schemes may exist, since one can resemble the classical exponential operator splitting methods in our abstract setup. In contrast, our proof of first order consistency for the concurrent variant uses the Local Stability Theorem in a way which does not allow to derive higher order schemes. Therefore, even if higher order concurrent variants exist, one would need to investigate other proof techniques.

Appendix

Theorem 2.17 (Discrete Gronwall Lemma [84, p. 46f.])
Let $(w_n)_{n\geq 0}$, $(a_n)_{n\geq 0}$ and $(b_n)_{n\geq 0}$ be sequences of non-negative real numbers with $w_0 \leq b_0$ and

$$w_n \leq \sum_{i=0}^{n-1} a_i w_i + b_n \quad (n \geq 1).$$

If the sequence $(b_n)_{n\geq 0}$ is monotonically increasing, then

$$w_n \leq \exp\Big(\sum_{i=0}^{n-1} a_i\Big) b_n \quad (n \geq 1)$$

holds.

Theorem 2.18 (Implicit Function Theorem [41, p. 292],[77, p. 128])
Let $G \subset \mathbb{R}^p$ and $H \subset \mathbb{R}^q$ be non-empty open sets, and the function $F : G \times H \longrightarrow \mathbb{R}^q$ continuously differentiable. Furthermore, let $\xi \in G$ and $\eta \in H$ be points with

$$F(\xi, \eta) = 0 \quad \text{and} \quad \frac{\partial F}{\partial y}(\xi, \eta) \text{ invertible.}$$

Then there exists a neighborhood $U \subset G$ of ξ and a neighborhood $V \subset H$ of η and exactly one continuous function $f : U \longrightarrow V$ with

$$f(\xi) = \eta \quad \text{and} \quad F(x, f(x)) = 0 \quad \forall\, x \in U\,.$$

Moreover, for any fixed $x \in U$ is $f(x)$ the only solution of $F(x, y) = 0$ which lies in V.

Theorem 2.19 (Convergence Theorem for the Newton iteration, following theorems 10.2.1 and 10.2.2 in [77, p. 310ff.])
Let $f : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$ be differentiable in an open neighborhood $D_0 \subset D$ of some point $x^* \in int(D)$ with $f(x^*) = 0$. Let ∇f be continuous at x^* and the Jacobian $\nabla f(x^*)$ be non-singular. Then there exists an open neighborhood $S \subset D_0$ of x^* such that the Newton iteration

$$x^{k+1} = x^k - \nabla f(x^k)^{-1} f(x^k)\,, \quad k = 0, 1, ... \tag{2.46}$$

is applicable with any initial vector $x^0 \in S$ and the iterate x^k converges super-linearly towards x^*. Moreover, if f is continuously differentiable on S and the second derivative of f exists at x^* and satisfies

$$\nabla^2 f(x^*)(x, x) \neq 0 \quad \forall\, x \in S\,,\, x \neq 0\,,$$

then the convergence is even quadratic.

In practice, the Newton iteration (2.46) is endowed with some stopping criterion. Widely used stopping criteria are based on prescribed tolerances for the residual norm or a maximum number of iterations. The following algorithm states the Newton iteration as we used it in this work.

Algorithm 1 Newton iteration

1: Set initial vector x^0, tolerance $\epsilon > 0$, $k_{\max} > 0$, and $k = 0$.
2: **while** $\|f(x^k)\| > \epsilon$ and $k < k_{\max}$ **do**
3: Solve $\nabla f(x^k) c^k = -f(x^k)$ for $c^k \in \mathbb{R}^n$.
4: Set $x^{k+1} = x^k + c^k$, and $k \leftarrow k + 1$.
5: **end while**

3 Numerical experiments on the convergence of the composed one step schemes

In this chapter we present numerical experiments on the convergence of the composed one step schemes which we presented in the previous Chapter 2. Our experiments are based on a natural convection fluid flow scenario. This scenario comprises an incompressible fluid flow model using the Boussinesq approximation to account for buoyancy, and a temperature evolution model. We use a second order monolithic solver to compute a reference solution which we compare with results obtained from the first order composed one step schemes. In the first section of this chapter, we outline the natural convection scenario in terms of the underlying modeling and discretization. The second section is dedicated to the numerical experiments on the convergence of the consecutive and the concurrent operator splitting schemes using the monolithic scheme as a reference.

3.1 Natural convection scenario

Our natural convection scenario is situated in some bounded domain $\Omega \subset \mathbb{R}^d$, where $\mathbb{R}^d$ represents the physical space of dimension $d = 2$ or $d = 3$. We consider a time interval $(0, T)$ with initial time $t = 0$ and final time $t = T > 0$ for evolution of the scenario. For the derivation of the model, we first recall the definition of a material volume and the Reynolds Transport Theorem.

Definition 3.1 (Material Volume)
The material volume $V(t) \subset \Omega$ of a set of fluid particles is the volume which is occupied by the particles at time $t \in (0, T)$.

Theorem 3.1 (Reynolds Transport Theorem [7])
Let $V(t) \subset \Omega$, $t \in (0, T)$ be a material volume. Then for any scalar differentiable function $\phi : \Omega \times (0, T) \to \mathbb{R}$ holds

$$\frac{d}{dt} \int_{V(t)} \phi \, dx = \int_{V(t)} \partial_t \phi + \nabla \cdot (\phi \boldsymbol{u}) \, dx \quad in \quad (0, T) \,, \tag{3.1}$$

where $\boldsymbol{u}$ is the velocity field of the fluid.

3.1.1 Continuity equation

The total mass m of the fluid particles building a material volume $V(t)$ is

$$m = \int_{V(t)} \rho \, dx$$

where ρ denotes the density of the fluid. Since the mass of this material volume does not change over time we have according to Thm. 3.1

$$0 = \frac{d}{dt} m = \frac{d}{dt} \int_{V(t)} \rho \, dx = \int_{V(t)} \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) \, dx \, .$$

Since the material volume was chosen arbitrary, this leads to the continuity equation

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0 \quad \text{in} \quad \Omega \times (0, T) \, , \tag{3.2}$$

which expresses the physical principle of conservation of mass.

3.1.2 Cauchy equation of motion

According to the second Newton axiom, the temporal variation of the momentum I is equal to the resultant force F, i.e. $dI/dt = F$. The momentum of a material volume is

$$I = \int_{V(t)} \rho \mathbf{u} \, dx$$

and the resultant force $F = F^v + F^s$ is the sum of a volumetric and a surface force. The volumetric force

$$F^v = \int_{V(t)} \rho f^v \, dx$$

acts on the fluid particles in $V(t)$ and is proportional to the mass, where f^v denotes the acceleration acting on the particles. The surface force

$$F^s = \int_{\partial V(t)} f^s \, ds$$

acts on the boundary $\partial V(t)$ and is proportional to the surface area, where f^s denotes the force per unit area acting on the surface. Using Thm. 3.1 we can write component wise

$$\int_{V(t)} \partial_t (\rho u_i) + \nabla \cdot (\rho u_i \mathbf{u}) = \int_{V(t)} \rho f_i^v \, dx + \int_{\partial V(t)} f_i^s \, ds \quad (i = 1, ..., d) \, ,$$

where $d = 2$ or $d = 3$ is the dimension of the physical space. It has been proved that there exists a second order tensor T, the stress tensor, such that

$$f^s = \mathbf{n} \cdot T \, ,$$

where $\mathbf{n}$ denotes the outer unit normal field on $\partial V(t)$ [7]. By means of the Gauss Theorem [41] it follows

$$\int_{\partial V(t)} f^s \, ds = \int_{V(t)} \sum_{j=1}^{d} \frac{\partial T_{ji}}{\partial x_j} \, dx \quad (i = 1, ..., d) \, .$$

Since the material volume was chosen arbitrary, this leads to the Cauchy equation of motion

$$\partial_t (\rho u_i) + \nabla \cdot (\rho u_i \mathbf{u}) = \rho f_i^v + \sum_{j=1}^{d} \frac{\partial T_{ji}}{\partial x_j} \quad \text{in} \quad \Omega \times (0, T) \quad (i = 1, ..., d) \, , \tag{3.3}$$

which expresses the momentum balance of a continuum under the influence of volumetric and surface forces.

3.1.3 Constitutive equations

The constitutive equations define the relation of the stresses in the continuum and the deformation or the rate of deformation. A fluid is characterized by the property that it can be arbitrarily deformed by shearing forces. The shearing force necessary to cause a deformation tends to zero whenever the rate of deformation tends to zero. This property born from the viscosity defines a fluid [99]. Due to the conservation of angular momentum in the fluid the stress tensor S can be assumed to be symmetric [7]. Polar fluids, where the angular momentum is not conserved in general, are not considered in this work.
A stress is called hydrostatic if on any surface element it acts in normal direction and is independent of the orientation. A hydrostatic stress has the form

$$S_{ij} = -p\delta_{ij} \quad (i,j = 1,...,d)\,,$$

where δ_{ij} denotes the Kronecker symbol. In general the stress can be written

$$S_{ij} = -p\delta_{ij} + P_{ij} \quad (i,j = 1,...,d) \tag{3.4}$$

with a second order tensor P, the viscous stress tensor. To derive constitutive equations for a non-elastic fluid, assumptions with respect to the stress tensor are made. A Stokesian fluid is characterized by the following assumptions:

1. The stress tensor S is a continuous function of the deformation tensor $e = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^\top)$ and of the thermodynamic state, but independent of other kinematic quantities.
2. The fluid is homogeneous, i.e. S does not explicitly depend on x.
3. The fluid is isotropic.
4. The stress is hydrostatic if there is no deformation, i.e. $S_{ij} = -p\delta_{ij}$ for $e = 0$.

The stress tensor of a Stokesian fluid has the form

$$S_{ij} = -p\delta_{ij} + \alpha\delta_{ij} + \beta e_{ij} + \gamma \sum_{k=1}^{d} e_{ik}e_{kj} \quad (i,j = 1,...,d)\,,$$

where α, β and γ are functions of the invariants of the deformation tensor. A Newtonian fluid is defined as a linear Stokesian fluid, i.e. the viscous stress tensor is a linear function of e. Therefore β is a constant, and $\gamma = 0$. The only linear invariant of the deformation tensor e is its trace $\nabla \cdot \mathbf{u}$. Due to the fourth assumption of a Stokesian fluid, α must be a linear function of $\nabla \cdot \mathbf{u}$. Thus P has the form

$$P_{ij} = \lambda(\nabla \cdot \mathbf{u})\delta_{ij} + 2\mu e_{ij} \quad (i,j = 1,...,d) \tag{3.5}$$

with constants $\lambda, \mu \in \mathbb{R}$, where μ is called the dynamic viscosity.

3.1.4 Boussinesq approximation

The Boussinesq approximation assumes the fluid to be incompressible, yet incorporating buoyancy effects. Due to the incompressibility assumption, the continuity equation (3.2) reduces to

$$\nabla \cdot \mathbf{u} = 0$$

and the Cauchy equation of motion (3.3) becomes

$$\rho\Big[\partial_t\mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u}\Big] = \rho f^v + \nabla p + \mu\Delta\mathbf{u}\,. \tag{3.6}$$

As volumetric force we have $\rho f^v = -\rho g \mathbf{e}_d$ with the gravitational acceleration g and upward vertical unit vector $\mathbf{e}_d$. Introducing a hydrostatic ground state with density ρ_0, pressure p_0, and temperature θ_0 which only depend on the vertical coordinate and which fulfill

$$\frac{\partial p_0}{\partial x_d} = -g\rho_0$$

and the ideal gas law

$$p_0 = R\rho_0\theta_0\,,$$

we write the actual quantities as $\rho = \rho_0 + \hat{\rho}$, $p = p_0 + \hat{p}$ and $\theta = \theta_0 + \hat{\theta}$, where the hat symbol denotes the deviation from the ground state. Therefore, (3.6) reads

$$\rho_0\Big(1 + \frac{\hat{\rho}}{\rho_0}\Big)\Big[\partial_t\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\Big] - \mu\Delta\mathbf{u} = -\rho_0 g\mathbf{e}_d - \hat{\rho}g\mathbf{e}_d - \nabla p_0 - \nabla\hat{p} = -\hat{\rho}g\mathbf{e}_d - \nabla\hat{p}\,.$$

Assuming furthermore that the variations of density, pressure and temperature are small compared to the ground states, the momentum equation simplifies to

$$\partial_t\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} + \frac{1}{\rho_0}\nabla\hat{p} = -\frac{\hat{\rho}}{\rho_0}g\mathbf{e}_d\,,$$

where $\nu = \mu/\rho_0$ is the kinematic viscosity. This equation is known as the Boussinesq approximation of the equation of motion with the buoyancy term $\hat{\rho}/\rho_0 g\mathbf{e}_d$ [28, p. 184]. From the ideal gas law the simplification $-\hat{\rho}/\rho_0 = \hat{\theta}/\theta_0$ can be derived [28, p. 184 ff.]. This leads to the form

$$\partial_t\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} + \frac{1}{\rho_0}\nabla\hat{p} = \frac{\hat{\theta}}{\theta_0}g\mathbf{e}_d\,. \tag{3.7}$$

3.1.5 Heat equation

We use the heat equation to model the evolution of the temperature θ of the fluid. Following [18], the conservation of heat energy derived from the first law of thermodynamics leads to

$$\rho c\frac{d\theta}{dt} - \nabla\cdot(k\nabla\theta) = 0\,,$$

where c denotes the specific heat capacity and k the thermal conductivity of the medium. Using Thm. 3.1 the heat equation reads

$$\rho c\Big[\partial_t\theta + \nabla\cdot(\theta\mathbf{u})\Big] - \nabla\cdot(k\nabla\theta) = 0 \quad \text{in} \quad \Omega\times(0,T)\,. \tag{3.8}$$

3.1.6 Natural convection model

We use a two-dimensional natural convection model for our numerical experiments. It is composed of the continuity equation (3.2), the Boussinesq approximation of the momentum equation (3.7), and the heat equation (3.8). We defined the ground states as $\rho_0 \equiv 1\,\mathrm{kg/m^3}$, $p_0 \equiv 0\,\mathrm{kg/ms^2}$, and $\theta_0 \equiv 273.15\,\mathrm{K}$. Assuming the fluid being incompressible, the continuity equation (3.2) reduces to $\nabla\cdot\mathbf{u} = 0$. Using this, and an isotropic thermal conductivity $k \equiv const$, the heat equation (3.8) simplifies to

$$\partial_t\theta + (\mathbf{u}\cdot\nabla)\theta - \alpha\Delta\theta = 0\,,$$

where $\alpha = k/\rho c$ denotes the thermal diffusivity.

Summing up, the natural convection model reads

$$\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} - \frac{1}{\rho}\nabla\hat{p} - \frac{\hat{\theta}}{\theta_0} g\mathbf{e}_d = 0 \quad \text{in} \quad \Omega\times(0,T)\,, \tag{3.9a}$$

$$\nabla\cdot\mathbf{u} = 0 \quad \text{in} \quad \Omega\times(0,T)\,, \tag{3.9b}$$

$$\partial_t\theta + (\mathbf{u}\cdot\nabla)\theta - \alpha\Delta\theta = 0 \quad \text{in} \quad \Omega\times(0,T)\,, \tag{3.9c}$$

$$\mathbf{u}(0) = 0 \quad \text{in} \quad \Omega\,, \tag{3.9d}$$

$$-\alpha\Delta\theta(0) = 0 \quad \text{in} \quad \Omega\,. \tag{3.9e}$$

$$\mathbf{u} = 0 \quad \text{on} \quad \partial\Omega\,, \tag{3.9f}$$

$$\theta = \theta_{\text{hot}} \quad \text{on} \quad \Gamma_{\text{hot}}\,, \tag{3.9g}$$

$$\theta = \theta_{\text{cold}} \quad \text{on} \quad \Gamma_{\text{cold}}\,, \tag{3.9h}$$

$$\nabla\theta\cdot\mathbf{n} = 0 \quad \text{on} \quad \Gamma_{\text{N}}\,, \tag{3.9i}$$

where $\Omega = (0,1)\times(0,1)$ is the domain with boundary $\partial\Omega = \Gamma_{\text{hot}}\cup\Gamma_{\text{cold}}\cup\Gamma_{\text{N}}$, and $(0,T)$ with $T = 30$ is the time interval. We use the Dirichlet-type boundary conditions (3.9g) and (3.9h) to model a heated wall $\Gamma_{\text{hot}} = \{0\}\times[0,1]$ and a cooled wall $\Gamma_{\text{cold}} = \{1\}\times[0,1]$. On the remaining boundary part $\Gamma_{\text{N}} = (0,1)\times\{0\}\cup(0,1)\times\{1\}$ we use the homogeneous Neumann-type boundary condition (3.9i) to model thermally insulated walls. For the velocity, we impose the no-slip condition (3.9f) on the whole boundary. In the initial state, we assume the fluid to be at rest by means of the initial condition (3.9d), and an equilibrium temperature state (3.9e). As parameters, we used the kinematic viscosity $\nu = 1.57\times10^{-5}\,\text{m}^2/\text{s}$, the thermal diffusivity $\alpha = 1.9\times10^{-5}\,\text{m}^2/\text{s}$, the gravitational acceleration $g = 9.81\,\text{m}/\text{s}^2$, the heated wall temperature $\theta_{\text{hot}} = 283.15\,\text{K}$, and the cooled wall temperature $\theta_{\text{cold}} = \theta_0 = 273.15\,\text{K}$. These parameters are realistic for an air flow scenario. The Rayleigh number is

$$\text{Ra} = \frac{g(\theta_{\text{hot}} - \theta_0)}{\nu\alpha\theta_0} \approx 10^9\,.$$

Theory on existence and uniqueness of strong and weak solutions of the Navier-Stokes equations can be found e.g. in [102] and [98].

3.1.7 Spatial discretization

We use a conforming Lagrange finite element Galerkin method for the spatial discretization of the natural convection model (3.9). The domain Ω is covered by a grid Ω_h of quadrilateral cells. For the velocity and pressure variable we use inf-sup-stable Q_2/Q_1 Taylor-Hood elements, and for the temperature we use Q_2 elements [27]. Denoting the trial and test functions for the velocity variable by $\boldsymbol{\varphi}$, for the pressure variable by ψ, and for the temperature variable by ϕ, the finite element ansatzes read

$$\mathbf{u}_h(x,t) = \sum_{i=1}^{n_{\mathbf{u}}} u_i(t)\boldsymbol{\varphi}_i(x)\,,$$

for the discrete velocity field $\mathbf{u}_h$ where the $\boldsymbol{\varphi}_i$ form a nodal basis of the Q_2 finite element space $\mathbf{U}_h = \{\boldsymbol{\varphi}\in Q_2(\Omega_h)\,,\ \boldsymbol{\varphi} = 0 \text{ on } \partial\Omega\}\subset H_0^1(\Omega)$, and

$$p_h(x,t) = \sum_{i=1}^{n_p} p_i(t)\psi_i(x)$$

for the discrete pressure field p_h where the ψ_i form a nodal basis of the Q_1 finite element space $P_h = \{\psi \in Q_1(\Omega_h)\} \subset L^2(\Omega)$, and

$$\theta_h(x,t) = \sum_{i=1}^{n_\theta} \theta_i(t)\phi_i(x)$$

for the discrete temperature field θ_h where the ϕ_i form a nodal basis of the Q_2 finite element space $T_h = \{\phi \in Q_2(\Omega_h)\,,\ \phi = 0 \text{ on } \Gamma_{\text{hot}} \cup \Gamma_{\text{cold}}\} \subset H^1(\Omega)$. For $t \in (0,T)$, the time-dependent coefficients can be written in vector form as

$$\begin{aligned} u(t) &= \Big(u_1(t), ..., u_{n_{\mathbf{u}}}(t)\Big)^\top \in \mathbb{R}^{n_{\mathbf{u}}}\,, \\ p(t) &= \Big(p_1(t), ..., p_{n_p}(t)\Big)^\top \in \mathbb{R}^{n_p}\,, \\ \theta(t) &= \Big(\theta_1(t), ..., \theta_{n_\theta}(t)\Big)^\top \in \mathbb{R}^{n_\theta}\,. \end{aligned}$$

Using above ansatz, the following variational form results from the natural convection model:

$$\begin{aligned} &\sum_{i=1}^{n_{\mathbf{u}}} \dot{u}_i(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_k) + \sum_{i,j=1}^{n_{\mathbf{u}}} u_i u_j((\boldsymbol{\varphi}_i \cdot \nabla)\boldsymbol{\varphi}_j, \boldsymbol{\varphi}_k) \\ &+\nu \sum_{i=1}^{n_{\mathbf{u}}} u_i(\nabla\boldsymbol{\varphi}_i, \nabla\boldsymbol{\varphi}_k) - \frac{1}{\rho}\sum_{i=1}^{n_p} p_i(\nabla\psi_i, \boldsymbol{\varphi}_k) \qquad (3.10a) \\ &\qquad -\frac{g}{\theta_0}\sum_{i=1}^{n_\theta} \theta_i(\phi_i \mathbf{e}_d, \boldsymbol{\varphi}_k) = 0 \quad \text{in} \quad (0,T) \quad (k = 1, ..., n_{\boldsymbol{u}})\,, \\ &\qquad \sum_{i=1}^{n_{\mathbf{u}}} u_i(\nabla \cdot \boldsymbol{\varphi}_i, \psi_k) = 0 \quad \text{in} \quad (0,T) \quad (k = 1, ..., n_p)\,, \qquad (3.10b) \\ &\sum_{i=1}^{n_\theta} \dot{\theta}_i(\phi_i, \phi_k) + \sum_{i=1}^{n_\theta} \theta_i((\mathbf{u}_h \cdot \nabla)\phi_i, \phi_k) \\ &\qquad +\alpha \sum_{i=1}^{n_\theta} \theta_i(\nabla\phi_i, \nabla\phi_k) = 0 \quad \text{in} \quad (0,T) \quad (k = 1, ..., n_\theta)\,, \qquad (3.10c) \\ &\qquad u_k(0) = 0 \quad (k = 1, ..., n_{\boldsymbol{u}})\,, \qquad (3.10d) \\ &\qquad p_k(0) = 0 \quad (k = 1, ..., n_p)\,, \qquad (3.10e) \\ &\qquad \theta_k(0) = \theta_{\text{hot}} \quad \text{if node } k \text{ lies on } \Gamma_{\text{hot}}\,, \qquad (3.10f) \\ &\qquad \theta_k(0) = \theta_{\text{cold}} \quad \text{if node } k \text{ lies on } \Gamma_{\text{cold}}\,, \qquad (3.10g) \\ &\alpha \sum_{i=1}^{n_\theta} \theta_i(0)(\nabla\phi_i, \nabla\phi_k) = 0 \quad \text{if node } k \text{ lies in } \Omega \cup \Gamma_{\text{N}}\,. \qquad (3.10h) \end{aligned}$$

Here, the boundary conditions (3.9f)-(3.9i), which are incorporated in the finite element spaces, have been treated in the usual way as indicated e.g. in [27] or [9] and are thus implicitly included in the variational form. Equation (3.10) represents an initial value problem for the time-dependent coefficient vectors u, p and θ. Although not present in the continuous model (3.9), where the pressure has the role of a Lagrange multiplier for the velocity, we artificially introduced the initial condition (3.10e) on the pressure to avoid ambiguity. The methodology

of discretizing a partial differential equation with respect to space and obtaining an ordinary differential equation in time, which is subsequently treated by means of an integrator, is called the method of lines [94]. We introduce the following short notation to represent Eq. (3.10):

$$M[\boldsymbol{\varphi}]\dot{u} + N[\boldsymbol{\varphi},\boldsymbol{\varphi},\boldsymbol{\varphi}](u,u) + \nu A[\boldsymbol{\varphi}]u - \frac{1}{\rho}B[\psi,\boldsymbol{\varphi}]p - \frac{g}{\theta_0}C[\phi,\boldsymbol{\varphi}]\theta = 0 \quad \text{in} \quad (0,T)\,, \tag{3.11a}$$

$$D[\boldsymbol{\varphi},\psi]u = 0 \quad \text{in} \quad (0,T)\,, \tag{3.11b}$$

$$M[\phi]\dot{\theta} + N[\boldsymbol{\varphi},\phi,\phi](u,\theta) + \alpha A[\phi]\theta = 0 \quad \text{in} \quad (0,T)\,, \tag{3.11c}$$

$$u(0) = 0\,, \tag{3.11d}$$

$$p(0) = 0\,, \tag{3.11e}$$

$$\alpha\tilde{A}[\phi]\theta(0) = \tilde{\theta}\,, \tag{3.11f}$$

where

$$\Big(M[\boldsymbol{\varphi}]\Big)_{ki} = (\boldsymbol{\varphi}_i,\boldsymbol{\varphi}_k) \quad , \quad \Big(N[\boldsymbol{\varphi},\boldsymbol{\varphi},\boldsymbol{\varphi}](x,y)\Big)_k = \sum_{i,j=1}^{n_{\mathbf{u}}} x_i y_j((\boldsymbol{\varphi}_i\cdot\nabla)\boldsymbol{\varphi}_j,\boldsymbol{\varphi}_k) \quad \text{for} \quad x,y\in\mathbb{R}^{n_{\mathbf{u}}}\,,$$

$$\Big(M[\phi]\Big)_{ki} = (\phi_i,\phi_k) \quad , \quad \Big(N[\boldsymbol{\varphi},\phi,\phi](x,y)\Big)_k = \sum_{i=1}^{n_{\mathbf{u}}}\sum_{j=1}^{n_\theta} x_i y_j((\boldsymbol{\varphi}_i\cdot\nabla)\phi_j,\phi_k) \quad \text{for} \quad x\in\mathbb{R}^{n_{\mathbf{u}}}, y\in\mathbb{R}^{n_\theta}\,,$$

$$\Big(A[\boldsymbol{\varphi}]\Big)_{ki} = (\nabla\boldsymbol{\varphi}_k,\nabla\boldsymbol{\varphi}_i) \quad , \quad \Big(A[\phi]\Big)_{ki} = (\nabla\phi_k,\nabla\phi_i)\,,$$

$$\Big(B[\psi,\boldsymbol{\varphi}]\Big)_{ki} = (\nabla\psi_i,\boldsymbol{\varphi}_k) \quad , \quad \Big(C[\phi,\boldsymbol{\varphi}]\Big)_{ki} = (\phi_i\mathbf{e}_d,\boldsymbol{\varphi}_k) \quad , \quad \Big(D[\boldsymbol{\varphi},\psi]\Big)_{ki} = (\nabla\cdot\boldsymbol{\varphi}_i,\psi_k)\,,$$

$$\Big(\tilde{A}[\phi]\Big)_{ki} = \begin{cases}\delta_{ki} & \text{if node } k \text{ lies on } \Gamma_{\text{hot}}\cup\Gamma_{\text{cold}}\\ (\nabla\phi_k,\nabla\phi_i) & \text{else}\end{cases} \quad , \quad \tilde{\theta}_k = \begin{cases}\theta_{\text{hot}} & \text{if node } k \text{ lies on } \Gamma_{\text{hot}}\\ \theta_{\text{cold}} & \text{if node } k \text{ lies on } \Gamma_{\text{cold}}\\ 0 & \text{else}\end{cases}\,.$$

In order to apply the operator splitting schemes proposed in Section 2, we use the form

$$(\dot{u},\dot{\theta})^\top = \mathcal{F}(u,p,\theta) + \mathcal{G}(u,\theta) \quad \text{in} \quad (0,T)\,, \tag{3.12a}$$

$$(u,\theta)^\top(0) = (0,\theta_0)^\top\,, \tag{3.12b}$$

where

$$\mathcal{F}(u,p,\theta) = \begin{pmatrix} -M[\boldsymbol{\varphi}]^{-1}\Big(N[\boldsymbol{\varphi},\boldsymbol{\varphi},\boldsymbol{\varphi}](u,u) + \nu A[\boldsymbol{\varphi}]u - \frac{1}{\rho}B[\psi,\boldsymbol{\varphi}]p - \frac{g}{\theta_0}C[\phi,\boldsymbol{\varphi}]\theta + D[\boldsymbol{\varphi},\psi]u\Big) \\ 0\end{pmatrix}\,,$$

$$\mathcal{G}(u,\theta) = \begin{pmatrix} 0 \\ -M[\phi]^{-1}\Big(N[\boldsymbol{\varphi},\phi,\phi](u,\theta) + \alpha A[\phi]\theta\Big)\end{pmatrix}\,,$$

$$\theta_0 = \tilde{A}[\phi]^{-1}\tilde{\theta}\,.$$

Clearly, $\mathcal{F}$ represents the Boussinesq fluid model, whereas $\mathcal{G}$ represents the temperature evolution model.

3.2 Numerical experiments

Our goal is to study the impact of the operator splitting approaches from Section 2.4.1 on the quality of the solution of the natural convection scenario. To this end, we conduct numerical

experiments where we use the operator splitting schemes with different time step sizes, and compare to a reference solution. We define three test series which differ in the time stepping scheme as follows:

1. **Monolithic test series:** (3.12) is integrated as a fully coupled system using the Crank-Nicolson time stepping scheme as stated in Alg. 2.

Algorithm 2 Monolithic solution scheme for (3.12) using the Crank-Nicolson integrator.

1: Set initial solution $(u^0, \theta^0)^\top = (0, \theta_0)^\top$.
2: **for** $n = 1, 2, ...$ **do**
3: Solve $(u^n, \theta^n)^\top = (u^{n-1}, \theta^{n-1})^\top + \frac{\Delta t}{2}\Big[\mathcal{F}(u^n, p, \theta^n) + \mathcal{G}(u^n, \theta^n) + \mathcal{F}(u^{n-1}, p, \theta^{n-1}) + \mathcal{G}(u^{n-1}, \theta^{n-1})\Big]$.
4: **end for**

2. **Consecutive operator splitting test series:** (3.12) is integrated using the consecutive operator splitting scheme as stated in Alg. 3, where the Crank-Nicolson method is used separately for the fluid and the temperature model.

Algorithm 3 Consecutive operator splitting scheme for (3.12) using the Crank-Nicolson integrator.

1: Set initial solution $(u^0, \theta^0)^\top = (0, \theta_0)^\top$.
2: **for** $n = 1, 2, ...$ **do**
3: Solve $(\hat{u}^n, \hat{\theta}^n)^\top = (u^{n-1}, \theta^{n-1})^\top + \frac{\Delta t}{2}\Big[\mathcal{F}(\hat{u}^n, p, \hat{\theta}^n) + \mathcal{F}(u^{n-1}, p, \theta^{n-1})\Big]$.
4: Solve $(u^n, \theta^n)^\top = (\hat{u}^n, \hat{\theta}^n)^\top + \frac{\Delta t}{2}\Big[\mathcal{G}(u^n, \theta^n) + \mathcal{G}(\hat{u}^n, \hat{\theta}^n)\Big]$.
5: **end for**

3. **Concurrent operator splitting test series:** (3.12) is integrated using the concurrent operator splitting scheme as stated in Alg. 4, where the Crank-Nicolson method is used separately for the fluid and the temperature model.

Algorithm 4 Concurrent operator splitting scheme for (3.12) using the Crank-Nicolson integrator.

1: Set initial solution $(u^0, \theta^0)^\top = (0, \theta_0)^\top$.
2: **for** $n = 1, 2, ...$ **do**
3: Solve $(\hat{u}^n, \hat{\theta}^n)^\top = (u^{n-1}, \theta^{n-1})^\top + \frac{\Delta t}{2}\Big[\mathcal{F}(\hat{u}^n, p, \hat{\theta}^n) + \mathcal{F}(u^{n-1}, p, \theta^{n-1})\Big]$.
4: Solve $(\tilde{u}^n, \tilde{\theta}^n)^\top = (u^{n-1}, \theta^{n-1})^\top + \frac{\Delta t}{2}\Big[\mathcal{G}(\tilde{u}^n, \tilde{\theta}^n) + \mathcal{G}(u^{n-1}, \theta^{n-1})\Big]$.
5: Set $(u^n, \theta^n)^\top = (\hat{u}^n, \hat{\theta}^n)^\top + (\tilde{u}^n, \tilde{\theta}^n)^\top - (u^{n-1}, \theta^{n-1})^\top$.
6: **end for**

We use the same spatial finite element discretization according to Sec. 3.1.7 in all test configurations. The rectangular computational grid Ω_h comprises 256×256 cells. We use inf-sup-stable Q_2/Q_1 Taylor-Hood elements for the velocity and pressure, and Q_2 elements for the temperature,

which results in 855,556 spatial degrees of freedom. In every test series, we compute solutions with varying time step sizes $\Delta t = \frac{1}{2}, \frac{1}{4}, ..., \frac{1}{64}$. We take the solution computed with the monolithic scheme with the time step size $\Delta t = \frac{1}{128}$ as the reference solution for comparison. For each test case out of the three test series, we compute the Euclidean error norm with respect to the reference solution at each time step. Our expectation is to observe the second order convergence of the Crank-Nicolson scheme for the monolithic test series, and to observe the first order convergence of the consecutive and concurrent operator splitting schemes as we proved in Section 2.4.1 for the two other test series.

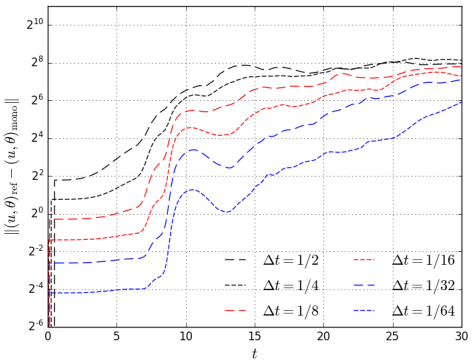


Figure 3.1: Error $\|(u,\theta)_{\text{ref}} - (u,\theta)_{\text{mono}}\|$ between the reference and the solution computed by means of the monolithic scheme using Alg. 2.

The results from the first test series using the monolithic integration scheme are plotted in Fig. 3.1. They show the convergence of the solution $(u,\theta)_{\text{mono}}$ with successively smaller time step sizes $\Delta t = \frac{1}{2}, \frac{1}{4}, ..., \frac{1}{64}$ towards the reference solution $(u,\theta)_{\text{ref}}$. Note that the reference solution was also computed using this monolithic scheme with the time step size $\Delta t = \frac{1}{128}$. As stated above, we expect second order convergence for the monolithic test series. Though, for $\Delta t \geq \frac{1}{16}$ our results do not meet this expectation. Although the time step size is successively decreased by a factor 2 using $\Delta t = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$, the error decreases less than the expected factor $2^2 = 4$. While the deviation from the expectation is only small for short integration times up to $t \approx 7$, it is certainly significant for larger integration times when using time step sizes $\Delta t \geq \frac{1}{16}$. We also observe a considerable increase of the error at $t \approx 7$ for all time step sizes. Nevertheless, when

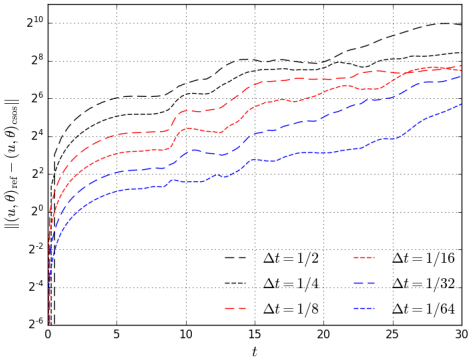


Figure 3.2: Error $\|(u,\theta)_{\text{ref}} - (u,\theta)_{\text{csos}}\|$ between the reference and the solution computed by means of the consecutive operator splitting scheme using Alg. 3.

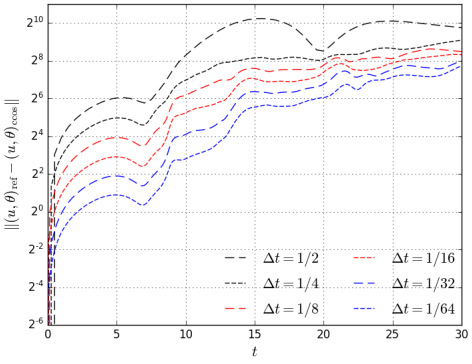
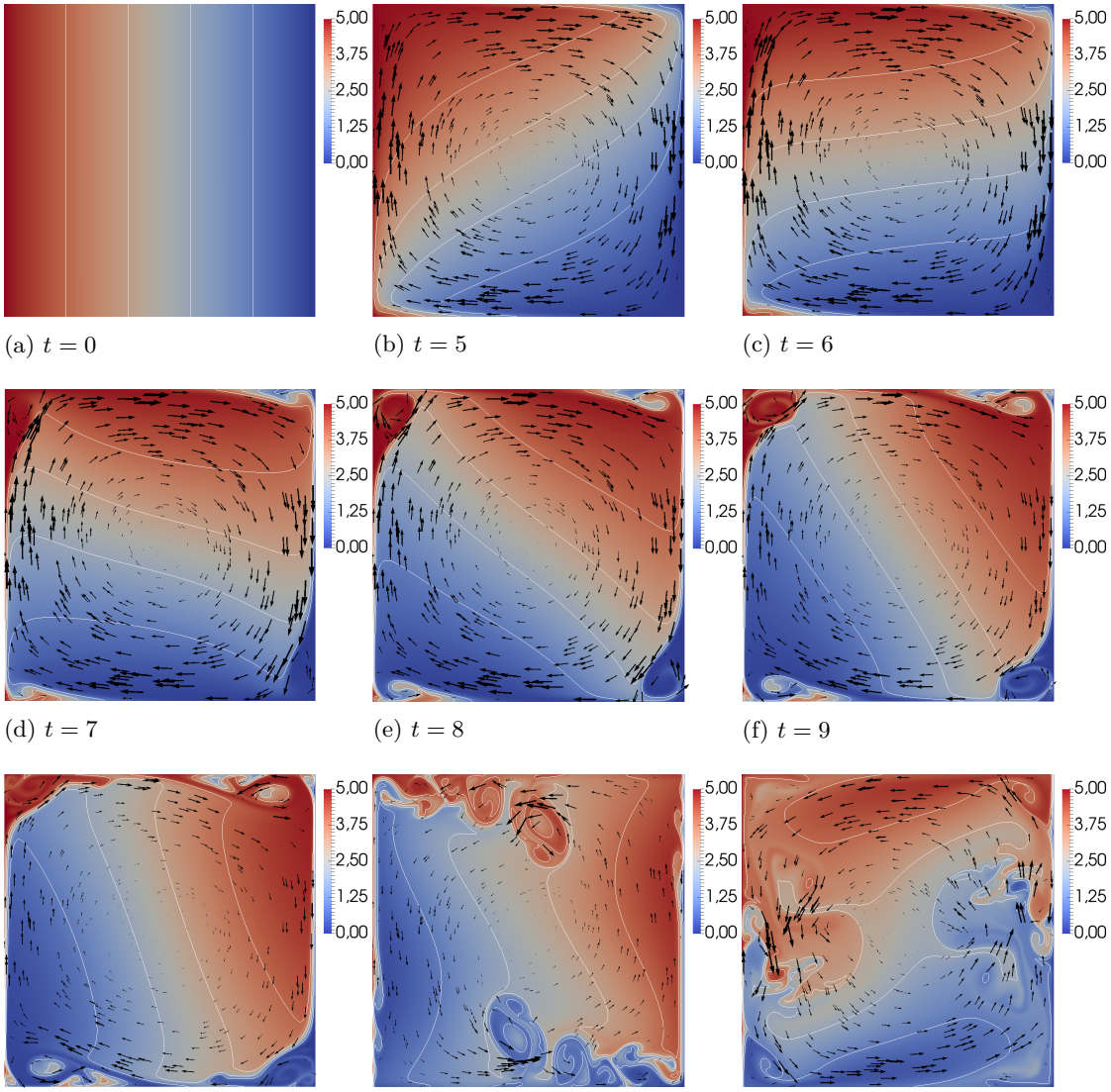


Figure 3.3: Error $\|(u,\theta)_{\text{ref}} - (u,\theta)_{\text{ccos}}\|$ between the reference and the solution computed by means of the concurrent operator splitting scheme using Alg. 4.



(a) $t = 0$ (b) $t = 5$ (c) $t = 6$

(d) $t = 7$ (e) $t = 8$ (f) $t = 9$

(g) $t = 10$ (h) $t = 15$ (i) $t = 20$

Figure 3.4: Reference temperature distribution (color) and flow velocity (arrows) at selected time steps. Note that the first vortices form up in the corners of the domain at $t \approx 7$. The vortices subsequently travel counterclockwise, and eventually cover large parts of the domain.

further decreasing the time step size to $\Delta t = \frac{1}{32}, \frac{1}{64}$ the expected second order convergence is actually achieved.

Figures 3.2 and 3.3 show the results from the consecutive and concurrent operator splitting test series, respectively. They both show very accurately the expected first order convergence for small integration times up to $t \approx 7$. For larger integration times, the first order convergence is only maintained when using the smallest time step sizes $\Delta t = \frac{1}{32}, \frac{1}{64}$. However, the concurrent

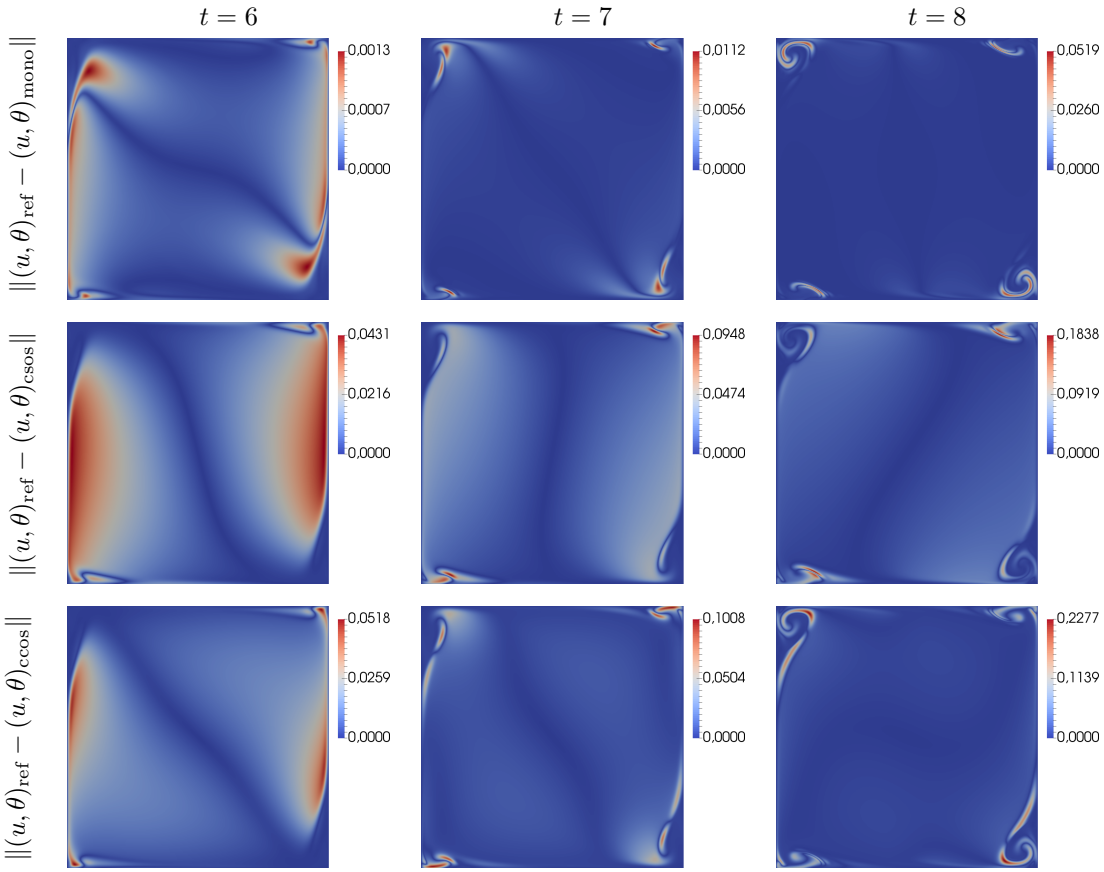


Figure 3.5: Error norm at times $t = 6$ (left column), $t = 7$ (middle column) and $t = 8$ (right column) between the reference u_{ref} and the monolithic solution (top row), the consecutive OS solution (middle row) and the concurrent OS solution (bottom row) using $\Delta t = 1/16$. Note that the color bars reflect the observed error increase around $t \approx 7$.

operator splitting scheme does not maintain the first order convergence for integration times larger than $t \approx 20$. As in the first test series, we also observe a noticeable error increase at $t \approx 7$ in these two test series.

In all three test series we observe that the expected order of convergence of the time stepping schemes is achieved accurately, or with only small deviations, for integration times up to $t \approx 7$. Furthermore, all three test series show a remarkable error increase at $t \approx 7$. The reason for that lies in the behavior of the natural convection scenario. As can be seen from the visualization of the reference solution in Fig. 3.4, a global clockwise flow evolves from the initial equilibrium temperature distribution. At $t \approx 7$ the first vortices form up in the corners of the domain. The vortices start to travel counterclockwise along the boundary, and eventually cover large parts of the domain. The visualizations of the error between the reference solution and the solutions from the three test series in Fig. 3.5 show the error growth around $t \approx 7$, and they show that the main error contributions appear in the corners. It is this change of the flow regime which is reflected in the error curves in Figures 3.1-3.3. Obviously, the initial flow regime allows for

larger time step sizes $\Delta t \geq \frac{1}{16}$ in all three test series, whereas the flow regime dominated by vortices requires smaller time step sizes $\Delta t \leq \frac{1}{32}$. Interestingly, among all three test series, the flow regime change causes the most drastic error deterioration for the second order monolithic integrator. As stated above, the first order operator splitting schemes also exhibit a remarkable error increase, but less drastic than the monolithic scheme. In the initial flow regime, the monolithic scheme is significantly more accurate than the operator splitting schemes. However, in the vortex-dominated flow regime the consecutive operator splitting yields an accuracy of the same order of magnitude as the monolithic scheme for $\Delta t \leq \frac{1}{4}$, and the concurrent operator splitting yields only slightly less accurate solutions.

We conclude from the results of our experiments that both the consecutive and the concurrent operator splitting scheme can yield accurate results for the time integration of the natural convection scenario. Since the operator splitting schemes investigated in this work are of first order as proved in Section 2.4.1, one cannot expect to achieve the same accuracy as higher order schemes for non-stiff problems. Indeed, the second order monolithic scheme shows better accuracy than the splitting schemes in the initial flow regime. Nevertheless, we indeed observed the expected first order convergence of the splitting schemes in the numerical experiments. However, the monolithic scheme for the fully coupled natural convection model showed its expected second order convergence only for the smallest time step sizes $\Delta t \leq \frac{1}{32}$. In contrast, the splitting schemes showed their expected first order convergence very accurately already for the larger time step sizes $\Delta t \leq \frac{1}{2}$. In the vortex-dominated flow regime, the operator splitting schemes performed similarly well as the monolithic scheme with respect to the achieved accuracy, although the splitting schemes are of lower order than the monolithic scheme. This is due to the stiffness of the ODE system induced by the flow regime change, which imposes stronger restrictions on the time step size. It turns out that the second order monolithic scheme suffers drastically from the flow regime change in the sense of a largely increased error. This error increase is much smaller, yet still noticeable, for the first order splitting schemes. All three schemes end up with a comparable accuracy for all but the smallest time step sizes. Remarkably, only for the smallest time step size $\Delta t = \frac{1}{64}$ is the monolithic scheme significantly more accurate than the concurrent operator splitting scheme, and equally accurate as the consecutive scheme.

4 Dynamic parallel communication mechanism in OpenPALM

As indicated in the introduction, we employ the OpenPALM [57, 80, 13] software coupler tool to realize the model coupling. Based on our work [109], in this chapter we recapitulate the development of the new features for advanced dynamic parallel communication routines in OpenPALM, which we use in the nutrient cycle simulations. OpenPALM is a coupler with advanced features like dynamic and concurrent execution models, the ability to couple parallel codes, and a flexible communication scheme. It is a general purpose coupling tool, although its main focus lies on scientific computing and numerical simulation. The fundamental concept of OpenPALM is to consider applications as a composition of models which can be coupled by means of a data transfer mechanism. One may in principal regard anything which can be implemented and executed as a computer program as a model in terms of OpenPALM. Models may represent a vast variety of computational tasks. Typical examples include reading or writing files, performing algebraic operations, solving systems of equations, up to large applications such as complete climate codes or ocean models. In our case, we regard biogeochemical and hydrological models as the building blocks to be coupled for nutrient cycle simulations. OpenPALM's goal is to easily enable the coupling of new and of existing models and codes, even if they were not meant to be coupled in the first place. Each model may be implemented individually. This offers the possibility to develop specialized solvers for the coupled models or to reuse existing codes with only minimal modifications. However, as we outlined in Sec. 1.1.3, OpenPALM's communication features are restricted to the case where data sizes and data distributions among the models are known a priori. In this chapter, we present the basic terms and concepts of the legacy OpenPALM version 4.1.4, and our developments towards the new dynamic features available in the current version 4.2.3 as open source[1].

OpenPALM consists of three main components: a graphical user interface named *PrePALM*, the *driver* and the *library*. The user can compose a coupled application in a pre-processing step with the help of the PrePALM graphical user interface (GUI). Its main feature is a canvas where the user can describe the coupling algorithm in a graphical form. This is done by defining execution paths, named *branches* in OpenPALM, scheduling the models by arranging them on the branches, and by connecting the models to indicate data transfer. Figure 4.1 shows an example of two independent execution branches, with one model scheduled to each of them.
OpenPALM allows dynamic control flows in the coupling algorithm. This includes the conditional execution of models where it is not known a priori if and when conditions are fulfilled, repeated execution of models in loops where it is not known a priori if and how often the loop will be executed, or execution switches with multiple alternative paths. Complex control flows can be defined using an arbitrary number of branches.
OpenPALM features two levels of parallelism. On the one hand, models can run concurrently on separate sets of processors when they are scheduled to separate execution branches. On the other hand, OpenPALM is able to couple models which are internally parallelized supporting both shared and distributed memory parallel models, which may internally use message passing,

[1] http://www.cerfacs.fr/globc/PALM_WEB/user.html retrieved on June 13, 2017

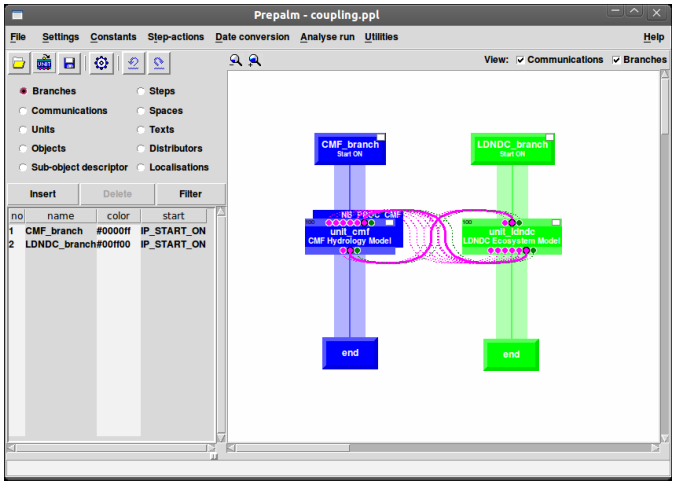


Figure 4.1: The canvas of the PrePALM graphical user interface. The coupling algorithm in this example has two independent execution branches, depicted as the blue and green vertical bars. Each of the branches has one model, represented by the two boxes in the middle, scheduled to it. Data transfer is defined through the connections between the models.

multi-threading and/or accelerators.
A key feature is the communication mechanism. Owing to OpenPALM's philosophy of coupling individual models, it is necessary to facilitate data transfer between models and at the same time keeping them general and independent from any particular application. Models are viewed as entities which produce and/or consume data and perform certain computational tasks, so that generality and reusability for any purpose is maintained. Therefore, models cannot know about communication partners. Instead, they need a way to request for input data or to announce the availability of output data without information about source or target of the communication. OpenPALM offers communication routines for sending and receiving data, which fulfill these requirements. These routines, among others, are implemented in the *library*. Developers can use them in the model's source code and link against the library. The communication routines are independent from the specific application at hand by using an abstract description of the data to be exchanged. These library routines used in the models are complemented by the OpenPALM *driver*. The driver is a special entity which is automatically adjoined to any coupled application. It has two main purposes: to orchestrate the execution of the branches and models, and to act as a broker for the data transfer between the models. The driver starts, stops, and monitors the execution of the branches, and controls the models' access to resources such as files, memory, or processors. It also forms the counterpart to the communication routines used in the models. Since models do in general not know their communication partners, they announce data transfer requests to the driver. The driver then deduces the correct matching of source and target, and arranges a connection between the corresponding models.
In the following sections we describe the concepts and implementation aspects which are relevant for the data transfer mechanism in more detail. In particular, we elaborate on the new mechanism for adapting the communication routes dynamically to changes of the data distribution in parallel models during runtime.

4.1 OpenPALM terms and concepts at the application level

4.1.1 Units

In OpenPALM terminology, a component or model that can be scheduled in a coupling algorithm is called a *unit*. Usually, units are defined by the user and may represent any kind of computational task. The granularity of the tasks may range from simple algebraic operations to whole simulation models. In addition, OpenPALM offers a predefined set of linear algebra units representing BLAS-like routines [3] and linear solvers for use with matrix and vector data structures.
Units are defined through an *identity (ID) card*. The ID card declares the properties of a unit which are relevant for OpenPALM like its name, its internal parallelism in terms of OpenMP [76] threads or MPI [5] processes, and its data objects which may be exchanged with other units. Details are described in [72]. The tasks represented by a unit can be implemented by the user in Fortran 77 [1], Fortran 90 [2], C [4] or C++ [6] programming language, making it possible to reuse existing codes and to couple units written in different programming languages.
An important aspect of the unit definition is the declaration of data objects which may be exchanged with other units. For OpenPALM to be able to manage the data transfer, it is necessary to describe the data in an abstract way in the ID card. Such an abstract description is composed of *objects*, *spaces*, *distributors* and *localizations*, which are explained in the following subsections.

4.1.2 Spaces

An OpenPALM *space* is an abstract description of a data type. To manage data transfer between units, OpenPALM needs to know the specification of this data. The space concept is based on the intrinsic data types of the programming languages for logical variables, characters, integers, single precision, double precision and complex floating point numbers. Table 4.1 lists the basic data types which are predefined in OpenPALM. Typically, spaces represent arrays of these basic types; such spaces are called *regular*. The *rank* of the space, i.e. the number of array dimensions, may be chosen between one and seven. The restriction to a maximum rank of seven is imposed by the Fortran 90 standard. The *shape* of the space may have individual extents of arbitrary length in each dimension. OpenPALM also provides the possibility to define *custom* spaces for describing derived types.

Basic data type	**OpenPALM keyword**	**Fortran 77 / 90**	**C / C++**
logic type	`PL_LOGICAL`	`LOGICAL`	`int`
character type	`PL_CHARACTER`	`CHARACTER`	`char`
integer type	`PL_INTEGER`	`INTEGER`	`int`
single precision floating point number	`PL_REAL`	`REAL`	`float`
double precision floating point number	`PL_DOUBLE_PRECISION`	`DOUBLE PRECISION`	`double`
complex floating point number	`PL_COMPLEX`	`COMPLEX`	`float complex`
derived types	`PL_AUTO_SIZE`		

Table 4.1: OpenPALM keywords and corresponding data types of Fortran 77/90 and C/C++ for defining spaces. When using complex numbers in C/C++, it is assumed that `complex.h` is included.

A space must be defined in the ID card of a unit. The required information is a name, a shape, and the size in bytes of the elements which make up this space. The sizes of the basic element types are known in OpenPALM. For custom spaces of derived element type, one must describe the composition of the derived element type out of known types. It is possible to nest the definition of custom spaces provided that the root of the nesting relies on the basic types. The keyword `PL_AUTO_SIZE` must be used for custom spaces so that OpenPALM can infer the derived element type size from its composition.

4.1.3 Distributors and localizations

If a unit is parallelized for distributed memory architectures using MPI, also its data objects may be split into parts and distributed among several processes. Any process of the unit may hold an individual local part of the overall global data. However, data transfer is described in the coupling algorithm on the unit level where internal parallelism and distribution of objects are hidden. Whenever data is transferred, OpenPALM must automatically take into account the local contributions of the processes belonging to the unit. The means for describing the distribution of data are the *distributors* and *localizations*. A distributor defines the decomposition of the global data into parts and the memory layout of the local storage, whereas a localization defines the ordered set of processes which take part in the distribution of an object. OpenPALM automatically assumes the default localization which consists only of the process with the lowest MPI rank. There are three predefined localizations which cover the following prevalent cases:

- `SINGLE_ON_FIRST_PROC`
  The data is not distributed, but held entirely by the first process of the unit. This is the default localization for all data without explicit definition of another localization.
- `DISTRIBUTED_ON_ALL_PROCS`
  The data is distributed over all processes of the unit.
- `REPLICATED_ON_ALL_PROCS`
  The data is not distributed, but replicated on all processes of the unit.

For all other cases, *custom localizations* can be defined in the unit's ID card. A localization definition comprises a name, a permutation of the process ranks which take part in the object distribution, and a keyword to distinguish distributed and replicated data. The user has to explicitly declare the localization in any other case than the three predefined cases. This is typically done when an object is distributed only over a subset of the processes belonging to the unit, or if the order of the local parts in the decomposition is different from the order of the MPI process ranks.

Regular distributors

Regular distributors can be used to describe a block cyclic decomposition of the data. The user defines a process grid with the same rank as the space of the data has, and an elementary block size. The data is split into blocks of the elementary block size. These blocks are cyclically numbered in each dimension with a cycling length equal to the process grid size in this dimension. Then the blocks are assigned to the process with the same coordinates in the process grid. Blocks which are assigned to the same process are stored contiguously in its local memory. Furthermore, it is possible to define arbitrary coordinates in the process grid to specify the process which shall receive the first local block. The local memory layout is defined by the shape of the local memory and an offset for the location of the first local block. Figure 4.2 shows a two-dimensional example of a regular distributor.

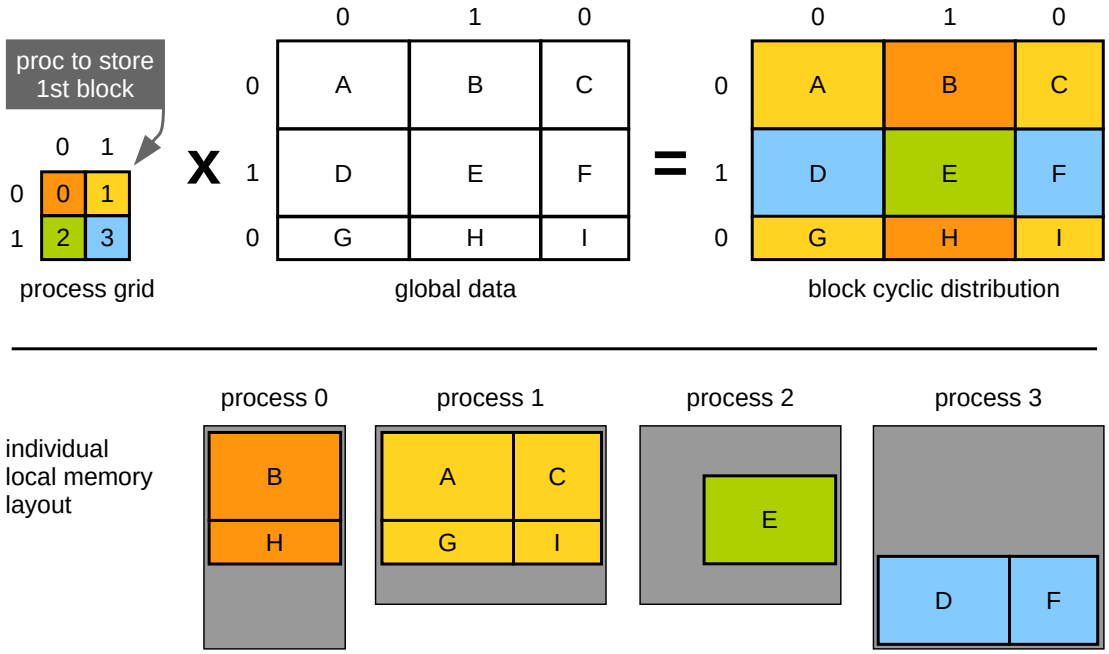


Figure 4.2: Example of a two-dimensional regular distribution. Four processes are arranged in a 2×2 grid. The data is split into blocks of the elementary block size. The last block in each dimension is smaller to fit the shape of the data. The block cyclic distribution begins with process rank 1 which stores block A. The complete distribution is indicated by the colors of the corresponding process. The lower part of the figure depicts a possible local memory layout of the four processes. Each processes may provide local memory with an individual size which is depicted in gray color. The local blocks are stored contiguously with an individual offset.

Custom distributors

Custom distributors do not rely on a regular pattern of the distribution, but one can describe any structured or unstructured decomposition of the data. This is done by explicitly specifying each block through its size, its location in the global data object, the process which stores the block and its location in the local memory. Figure 4.3 shows a two-dimensional example of a custom distributor.

4.1.4 Objects

Objects are the identifiers of the actual pieces of data which a unit may send to or receive from other units. They must be declared in the ID cards of the units. Objects carry a user-given name which must be unique within the same unit. OpenPALM internally suffixes any object name with the corresponding unit name so that the declaration is independent from other units. The data type of an object is defined by a space. Objects can be defined as input, output, or both, depending on the purpose of the data. If multiple instances of the same object shall be distinguished, one can use the *time* and *tag* attributes. For example, in time dependent simulations, units may use the same object to exchange the solution for every time step, but the new solution shall be treated as an additional instance of the object instead of overwriting the

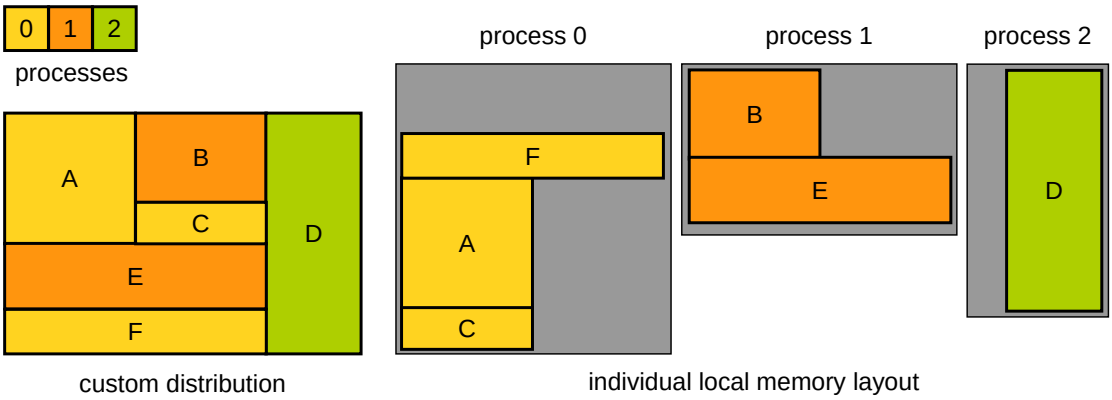


Figure 4.3: Example for the use of a custom distributor to describe an unstructured two-dimensional block decomposition. The three processes use an individual memory layout to store the blocks.

one from the last time step. The tag attribute serves the same purpose to distinguish instances of the same object, and can be used independently from the time attribute. For parallel units, the declaration of an object may further include a distributor and a localization to specify its distribution among the processes of the unit.

4.1.5 Sub-objects

Whenever a situation occurs where not a whole object but only a part of it needs to be exchanged between units, one can use the *sub-objects* mechanism. For example, this might be useful for obtaining the boundary values from a solution which is defined on an entire domain. Sub-objects act as a filter on the communication. Similar to distributors, there are *regular* and *custom sub-objects*. A regular sub-object describes a block cyclic selection from the data, whereas custom sub-objects can describe any regular or irregular subset of the data. It depends on the specific application at hand whether any sub-objects are needed or not. Sub-objects are therefore defined in PrePALM and not in the ID cards to maintain the independence of the units. Sub-objects can be used either for the source object or for the target object or for both objects. Figures 4.4 and 4.5 show examples for the use of a regular and a custom sub-object, respectively. For technical reasons, there is the predefined *identity sub-object* which actually comprises the whole original object. The identity sub-object is always assumed implicitly unless the user defines another sub-object.

4.2 Data exchange between units

OpenPALM provides two basic routines for exchanging data between units, namely `PALM_Put` and `PALM_Get` for sending and receiving, respectively. These routines are implemented in the OpenPALM library. They can be used in the unit source code by including the OpenPALM header file and linking the library.

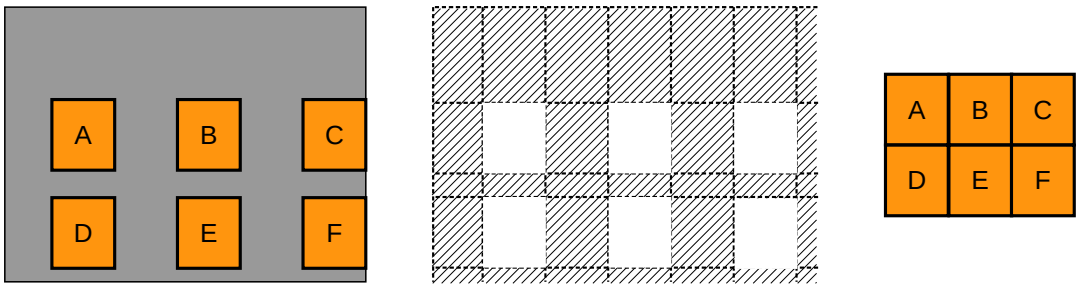


Figure 4.4: Regular sub-object example. The pattern (center) defines the selection from the global object (left), which is assembled into the sub-object (right).

4.2.1 The PALM_Put routine

The `PALM_Put` routine can be used in the units' source code to send out data objects. It takes as input parameters the space and object name associated with the data, the time and tag, and the pointer to the memory location where the data resides; it returns an error code.
Owing to OpenPALM's philosophy that producing and sending out data objects shall not block the execution of the source unit, `PALM_Put` is non-blocking in the sense that it does not wait for communication partners to receive the object. In case that a target unit is not yet ready for reception, the data object is temporarily stored in a dedicated mail buffer storage managed by the driver, so that the source unit can proceed its execution.

4.2.2 The PALM_Get routine

The `PALM_Get` routine can be used in the units' source code to receive data objects. It takes as input parameters the space and object name associated with the data, the time and tag, and the pointer to the memory location where the received data shall be stored; it returns an error code. It is assumed that units call `PALM_Get` only when data is needed to continue the computation. Therefore, the `PALM_Get` routine is blocking in the sense that it returns only when the requested data object has been received, otherwise it waits for the delivery.

4.2.3 Direct communication

A direct communication happens if the target unit has already notified its readiness to receive an object by calling `PALM_Get` before the source unit calls the corresponding `PALM_Put`. In this case, the OpenPALM driver arranges the connection between the units and the object is transferred by means of MPI messages directly from the source to the target. If one or both units are MPI-parallelized and the object is distributed on either side, then the communication is actually done by transferring the distributed object parts between the individual processes of the units according to the intersection of the involved distributors and localizations.

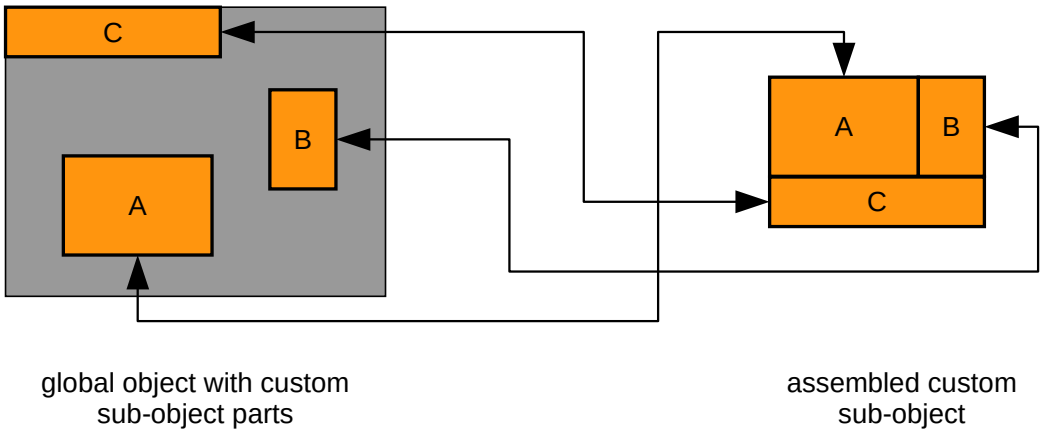


Figure 4.5: Custom sub-object example. Arbitrary selections from the global object (left) can by defined by means of a custom sub-object. It maps the parts to the assembled sub-object (right), which is indicated by the arrows in the schematic.

4.2.4 Indirect communication

An indirect communication happens if the target unit is not yet ready to receive an object at the moment when the source unit calls `PALM_Put` to send it. In this case, the driver adopts the role of an intermediate mail storage. It receives the object from the source unit and stores it temporarily in the so called *mailbuf*. Later, when the actual target unit is ready to receive the object and calls `PALM_Get`, the driver serves this request with the temporarily stored object from the mailbuf. If one or both objects are MPI-parallelized and the object is distributed on either side, then the object is put to the mailbuf, temporarily stored, and forwarded to the individual target processes by parts according to the intersection of the involved distributors and localizations. The actual data transfer is done by means of MPI messages between the source unit and the driver for intermediate storage in the mailbuf, and between the driver and the target unit for later delivery. The mailbuf is volatile in the sense that stored objects are erased once they have been delivered to the target.
If the driver runs out of memory for storing objects in the mailbuf, it may spawn one or more additional processes which serve as memory slaves extending the available memory capacity. If memory slaves are active, the driver may order them to receive and temporarily store the object from the source unit, and later forward it to the target unit. Whenever a memory slave runs empty because it has delivered all stored objects to the target units, the driver may terminate it. Memory slaves can be spawned again if needed.

4.2.5 Buffer communication

Users can not only define communication between two units, but also with the buffer. This is a storage provided by the driver which can be explicitly used as source or target in communications, in contrast to the mailbuf which is never the source or target of a communication but only an intermediate storage. The buffer is non-volatile in the sense that stored objects are erased only upon invocation of corresponding commands by the user, which is another important difference from the mailbuf where objects are automatically erased after delivery. The preservation of objects in the buffer allows to receive them multiple times by an arbitrary number of units. It

is also possible to perform operations on the objects in the buffer, e.g. summation, averaging or interpolation of data which represents a time series. The driver may use the same mechanism of memory slaves as described for the indirect communication if its buffer memory capacity is exhausted.

4.2.6 Optimized communication mode

The purpose of the optimized communication mode is to avoid the interaction with the driver, which is necessary in the default non-optimized communication mode where the driver needs to inform the units about the routing table of the object. Instead, the optimized mode enforces a direct communication using a pre-computed routing table and dedicated MPI communicators between source and target unit. As a tradeoff, the optimized mode imposes constraints on the coupling algorithm, since not only the target side but also the source side uses blocking MPI routines to facilitate the data transfer, thus implying the risk of deadlocks.

4.3 OpenPALM terms and concepts for internal communication management

We introduce further terms and concepts of OpenPALM which are relevant for the description of the internal communication mechanism, and for the presentation of our new developments.

4.3.1 Entities

Entities are those structures which can be scheduled in an OpenPALM coupling algorithm and which may take part in communications. One such type of entity are the units, which are defined above. In addition to the units, there are other types of entities. These are the driver with its permanent buffer, the driver or memory slaves with the mail buffer, blocks of units which are integrated into the same executable, and code which is not part of a unit but directly imported from PrePALM.

4.3.2 Tubes, communication events, comids, and the commstate table

As can be seen from the example in Fig. 4.1, communication between entities is defined by means of connections in the PrePALM canvas. Each such communication channel is called a *tube*, and it is characterized by its source entity and object and its target entity and object. A *communication event* is specified by a tube and a time-tag combination and, in case of replicated objects where the localization is `REPLICATED_ON_ALL_PROCS`, the source and target process association. For each tube, the possible communications through this tube are defined by the time-tag combinations the user has given in PrePALM and by the localization associations. Since the time-tag combinations rely only on PrePALM- and other constants and arithmetic expressions of them, and since the localization associations are given by the distribution of the objects with fixed numbers of processes, there is only a finite number of possible communication events for each tube. Therefore, all of the possible communication events can be determined a priori from the user-given information contained in the coupling algorithm described in PrePALM. However, it is usually not known in advance, if, how often and when any particular communication event will occur. Each communication event can be given a unique number by counting all possible communication events, which is the concept of the *comids*. A comid is a non-negative integer value, by which a certain communication event can be identified uniquely throughout the whole application. The

driver does a bookkeeping of all communication events in its *commstate table*. It keeps track of the calls to `PALM_Put` and `PALM_Get`, and of the contributions of the individual processes in the case of MPI-parallel entities. It also does a version control for multiple occurrences of the same communication event. The driver uses the information contained in the commstate table to serve the entities' communication requests.

4.3.3 EOS and DOR

Whenever a communication event occurs, OpenPALM determines the corresponding comid. The comid then allows to deduce the combination of entity, object and space, short *EOS*, for both the source side and the target side of the associated tube. According to these two EOS, one for each side of the tube, OpenPALM further determines any associated distributors, localizations and sub-objects. For preparing the communications, OpenPALM computes the intersection of all relevant distributors, localizations and sub-objects. This intersection generates a *distributed object representation (DOR)* for the source and target side of each tube. A DOR specifies the splitting of the possibly distributed global object into the parts resulting from the intersection, and the process by process matching of these parts between source and target entity. Thus, the DOR parts are non-overlapping, each DOR part represents a piece of the object which is transferred between a specific source process and a specific target process, and their union resembles the global object. For an actual data transfer, OpenPALM temporarily allocates memory for the DOR parts. On the source side, the input data is copied piecewise to the DOR memory, which is called *disassembling*. The DOR allows to iterate through the parts, which are sent successively from the DOR memory to the corresponding receiver process. On the target side, each process receives its DOR parts in the corresponding DOR memory. Once all parts have arrived at the target, the object is rebuilt by copying the DOR parts to the correct locations in the output data memory, which is called *assembling*. After the transmission has completed, the temporary DOR memory is freed again.

4.4 Internal data transfer mechanism in the legacy OpenPALM version 4.1.4

In the following, we present the legacy data transfer mechanism as it was implemented in OpenPALM version 4.1.4. We state the legacy `PALM_Put` and `PALM_Get` routines in Algs. 5 and 6, respectively, and the legacy implementation of the driver's reactions to these communication routines in Algs. 7 to 10. Some steps of the algorithms are marked with an asterisk indicating that an explanatory comment is given below.

Algorithm 5 Legacy `PALM_Put`

1: Input: space name, object name, time, tag, pointer to local memory where the local object is stored.
2: Determine the comids which are relevant for this call to `PALM_Put`.(*)
3: Set $flag = 0$.
4: **for** each comid **do**
5: **if** this is a non-optimized communication **then**
6: Check local mail buffer flag.(*)
7: **if** $flag == 0$ **then**
8: Notify the `PALM_Put` to the driver.(*)
9: Receive the driver's answer.(*)
10: Set $flag = 1$.(*)
11: **end if**
12: Determine rank and current shape of the space.
13: **if** the space is dynamic **then**
14: Check that the distributor is `SINGLE_ON_FIRST_PROC`, otherwise abort.(*)
15: Compute the DOR according to the distributor intersection from the source side.(*)
16: **end if**
17: Allocate temporary memory for the DOR parts and disassemble the local object.
18: Determine which target processes shall receive a part of the disassembled local object from this source process.
19: **for** each process of the target's distributor **do**
20: **if** that target process shall receive a part **then**
21: **if** the object shall be written to a file **then**
22: Write the object part to the file.
23: **else**
24: Send a connection request to the driver.(*)
25: Receive a connection authorization from the driver.(*)
26: Extract the communication type, the receiver entity type, and the receiver MPI process rank from the drivers answer.
27: **if** the receiver is a unit or a memory slave **then**
28: Create an MPI intercommunicator to the receiver process.
29: **end if**
30: Send the object part to the receiver process.(*)
31: **if** the receiver is a unit or a memory slave **then**
32: Delete the intercommunicator to the receiver process.
33: **end if**
34: **end if**
35: **end if**
36: Notify the completion of the transfer of this object part to the driver.
37: **end for**
38: **if** the space is dynamic **then**
39: Delete the DOR which was just created in step 15.
40: **else**
41: Deallocate the temporary memory for the DOR parts.
42: **end if**

43: **else** this is an optimized communication
44: Get the MPI intracommunicator for this object.
45: Determine rank and current shape of the space.
46: **if** the space is dynamic **then**
47: Check that the source distributor is `SINGLE_ON_FIRST_PROC`, otherwise abort.(*)
48: Compute the DOR according to the distributor intersection from the source side.(*)
49: **end if**
50: Allocate temporary memory for the DOR parts and disassemble the local object.
51: **for** each process of the target distributor **do**
52: **if** that target process shall receive a part **then**
53: Create an MPI intercommunicator to the receiver process.
54: Send the object part to the receiver process.(*)
55: Delete the intercommunicator to the receiver process.
56: Call `MPI_Barrier` on the local process group of this object's MPI intracommunicator.(*)
57: **end if**
58: **end for**
59: **if** the space is dynamic **then**
60: Delete the DOR which was just created in step 48.
61: **else**
62: Deallocate the temporary memory for the DOR parts.
63: **end if**
64: **end if**
65: **end for**

Explanatory comments on Alg. 5:
Step 2: Note that the calling EOS might be the source in several tubes. For each affected tube, the time-tag and localization matching is done, and the corresponding comids are determined. The rank of the calling process with respect to the source distributor is derived from the localization.
Step 6: Units which are integrated into the same block can use local mail buffer memory to exchange data instead of using the driver's mail buffer.
Step 8: The driver's reaction to this notification is stated in Alg. 7.
Step 9: The driver's answer is given in step 15 of Alg. 7.
Step 10: The flag is used to ensure that the driver is notified only once, even if there are multiple comids. The driver can itself determine all relevant comids.
Steps 14 and 47: This is the reason why the legacy `PALM_Put` algorithm can only handle dynamic spaces for non-distributed objects.
Steps 15 and 48: Since dynamic spaces are only possible with non-distributed objects in the legacy `PALM_Put`, the resulting DOR just represents the new space shape.
Step 24: The driver's reaction to this connection request is stated in Alg. 8.
Step 25: The driver's answer is given in steps 3, 8 or 24 of Alg. 8.
Step 30: The object part is received in step 26 of Alg. 6 in case of a direct communication, or in step 18 or 20 of Alg. 8 in case of an indirect communication, or in step 12 or 14 of Alg. 8 in case of a buffer communication.
Step 54: The object part is received in step 49 of Alg. 6.
Step 56: The MPI barrier is called on the local process group, i.e. only on the source side. This is necessary to avoid race conditions where several source processes could send object parts to the same target process at the same time.

Algorithm 6 Legacy `PALM_Get`

```
1: Input: space name, object name, time, tag, pointer to local memory where the local object
   shall be stored after reception.
2: Check if there is a comid for this call to PALM_Get.
3: if there is a comid then
4:     if this is a non-optimized communication then
5:         Notify the PALM_Get to the driver.(*)
6:         Receive the driver's answer.(*)
7:         Check local mail buffer flag.(*)
8:         Determine rank and current shape of the space.
9:         if the space is dynamic then
10:            Check that the target distributor is SINGLE_ON_FIRST_PROC, otherwise abort.(*)
11:            Compute the DOR according to the distributor intersection from the target side.(*)
12:        end if
13:        Get the number of source object parts for this target process.
14:        if the object shall be read from a file then
15:            Read local object parts from file.
16:        end if
17:        Receive information from the driver where the object parts are located.(*)
18:        Allocate temporary memory for the DOR parts.
19:        for each source process do
20:            if that source process contributes an object part then
21:                Send a connection request to the driver.(*)
22:                Check if an MPI intercommunicator is needed.(*)
23:                if an MPI intercommunicator is needed then
24:                    Create an MPI intercommunicator to the sender process.
25:                end if
26:                Receive the object part from the sender process.(*)
27:                if an MPI intercommunicator was need then
28:                    Delete the MPI intercommunicator to the sender process.
29:                end if
30:            end if
31:        end for
32:        Assemble the DOR parts to build the local object.
33:        if the space is dynamic then
34:            Delete the DOR which was just created in step 11.
35:        else
36:            Deallocate the temporary memory for the DOR parts.
37:        end if
38:    else this is an optimized communication
39:        Determine rank and current shape of the space.
40:        if the space is dynamic then
41:            Check that the target distributor is SINGLE_ON_FIRST_PROC, otherwise abort.(*)
42:            Compute the DOR according to the distributor intersection from the target side.(*)
43:        end if
44:        Get the number of source DOR parts for this target process.
45:        Allocate temporary memory for the DOR parts.
46:        for each source process do
47:            if that source process contributes a DOR part then
48:                Create an MPI intercommunicator to the sender process.
```

```
49:                 Receive the object part from the sender process.(*)
50:                 Delete the MPI intercommunicator to the sender process.
51:             end if
52:         end for
53:         if the space is dynamic then
54:             Delete the DOR which was just created in step 42.
55:         else
56:             Deallocate the temporary memory for the DOR parts.
57:         end if
58:     end if
59: end if
```

Explanatory comments on Alg. 6:
Step 5: The driver's reaction to this notification is stated in Alg. 9.
Step 6: The driver's answer is given in steps 13 or 22 of Alg. 9.
Step 7: Units which are integrated into the same block can use local mail buffer memory to exchange data instead of using the driver's mail buffer.
Steps 10 and 41: This is the reason why the legacy `PALM_Get` algorithm can only handle dynamic spaces for non-distributed objects.
Steps 11 and 42: Since dynamic spaces are only possible with non-distributed objects in the legacy `PALM_Get`, the resulting DOR just represents the new space shape.
Step 17: The object part locations are sent by the driver in step 12 of Alg. 7 in case of a direct communication, or in step 23 of Alg. 9 in case of an indirect or buffer communication.
Step 21: The driver's reaction to this connection request is stated in Alg. 10.
Step 22: An MPI intercommunicator needs to be created if the sender is not the driver, to which an intercommunicator exists anyway, and when the local mail buffer is not used.
Step 26: The object part is sent in step 30 of Alg. 5 in case of a direct communication, or in step 10 or 12 of Alg. 10 in case of an indirect communication, or in step 3 or 5 of Alg. 10 in case of a buffer communication.
Steps 49: The object part is sent in step 54 of Alg. 5.

Algorithm 7 Driver reaction on the `PALM_Put` notification in step 8 of Alg. 5

```
1: Determine the comids which are relevant for this call to PALM_Put.
2: for each comid do
3:     Insert this PALM_Put into the commstate table.
4:     Search the commstate table for a waiting PALM_Get matching this comid.
5:     if there is a waiting PALM_Get then
6:         Update the commstate table that the PALM_Get will be served by this PALM_Put.
7:         if the source is the buffer then
8:             Determine if the object is located at the driver or a memory slave.
9:         end if
10:        for each target process do
11:            Send answer to target process.(*)
12:            Send object part locations to the target process.(*)
13:        end for
14:    end if
15:    Send answer to the source process.(*)
16: end for
```

Explanatory comments on Alg. 7:
Step 11: This answer is received in step 6 of Alg. 6.
Step 12: The object part locations are received in step 17 of Alg. 6.
Step 15: This answer is received in step 9 of Alg. 5.

Algorithm 8 Driver reaction on the `PALM_Put` connection request in step 24 of Alg. 5

1: Query the commstate table if this `PALM_Put` matches a waiting `PALM_Get`.
2: **if** a `PALM_Get` is waiting for the connection **then**
3: Send a connection authorization to the source process telling that the communication type is direct, the target is a unit, and the MPI rank of the receiver process.(*)
4: **else** the target is the buffer, or the object must be temporarily stored in the mailbuf
5: Determine if the target is the buffer, or if the mailbuf shall take the object part.
6: Determine if the driver or a memory slave shall receive the object.
7: **if** the driver receives the object part **then**
8: Send a connection authorization to the source process telling whether the communication type is buffer or indirect, the receiver is the driver, and the MPI rank of the driver.(*)
9: **end if**
10: **if** the target is the buffer **then**
11: **if** the driver receives the object part **then**
12: Receive the object part from the source process and put it in the buffer.(*)
13: **else**
14: Order a memory slave to receive the object part and to put it in the buffer.(*)
15: **end if**
16: **else**
17: **if** the driver receives the object part **then**
18: Receive the object part from the source process and put it in the mailbuf.(*)
19: **else**
20: Order a memory slave to receive the object part and to put it in the mailbuf.(*)
21: **end if**
22: **end if**
23: **if** a memory slave receives the object **then**
24: Send a connection authorization to the source process telling whether the communication type is buffer or indirect, the receiver is a memory slave, and the MPI rank of the memory slave.(*)
25: **end if**
26: **end if**

Explanatory comments on Alg. 8:
Steps 3, 8 and 24: This answer is received in step 25 of Alg. 5.
Steps 12, 14, 18 and 20: The object part is sent in step 30 of Alg. 5.

Algorithm 9 Driver reaction on the `PALM_Get` notification in step 5 of Alg. 6

```
1: Determine the comids which are relevant for this call to PALM_Get.
2: Query the commstate table to choose a comid.
3: if this PALM_Get does not read from a file then
4:     if not all processes of the target entity have announced this PALM_Get yet then
5:         Return.(*)
6:     end if
7:     if there is neither a matching PALM_Put currently active nor is the object available in the
       mailbuf or buffer then
8:         Memorize the PALM_Get to be wakened later in step 5 of Alg. 7 when a matching
           PALM_Put occurs.
9:         Return.(*)
10:    end if
11: else the object is read from a file
12:    for each target process do
13:        Send a dummy answer to the target process.(*)
14:    end for
15: end if
16: for each target process do
17:    for each source process do
18:        if the object part is in the mailbuf then
19:            Determine whether the object part is stored on the driver or a memory slave.
20:        end if
21:    end for
22:    Send an answer to the target process telling the chosen comid for this PALM_Get.(*)
23:    Send the object part locations to the target process.(*)
24: end for
```

Explanatory comments on Alg. 9:
Step 5: The driver only continues to serve this `PALM_Get` when all processes of the target entity have announced it. Since the target processes wait for the answer in step 6 of Alg. 6, this implies a synchronization among the target processes.
Step 9: Since the target processes are waiting for the answer in step 6 of Alg. 6, this makes the target wait until a matching `PALM_Put` occurs.
Step 13: This is necessary to let the target entity continue since it is waiting for the answer in step 6 of Alg. 6.
Step 22: This answer is received in step 6 in Alg. 6.
Step 23: The object part locations are received in step 17 in Alg. 6.

Algorithm 10 Driver reaction on the `PALM_Get` connection request in step 21 of Alg. 6

```
1: if the source is the buffer then
2:     if the object is stored in the driver then
3:         Send the object part to the target process.(*)
4:     else
5:         Determine which memory slave has the object and order it to send the object part to
           the target process.(*)
6:     end if
```

```
 7: else the source is not the buffer
 8:     if the object part is served from the mailbuf then
 9:         if the object is stored on the driver then
10:             Send the object part to the target process.(*)
11:         else
12:             Determine which memory slave has the object and order it to send the object part
                to the target process.(*)
13:         end if
14:     end if
15: end if
```

Explanatory comments on Alg. 10:
Steps 3, 5, 10 and 12: The object part is received in step 26 of Alg. 6.

4.5 New features to enable dynamic spaces, distributors and sub-objects in OpenPALM version 4.2.3

As indicated above, the legacy OpenPALM version 4.1.4 exhibited several restrictions in the use of spaces, distributors and sub-objects. The goal of our work on OpenPALM is to overcome these restrictions. We have developed and implemented a mechanism which allows to use dynamic spaces not only for non-distributed objects, but also for distributed or replicated objects, in OpenPALM version 4.2.3. Moreover, we have developed and implemented a means for changing the distributor and sub-object definitions during runtime of OpenPALM applications, while maintaining the consistency between sources and targets of communications without the need for additional synchronizations. Using a similar denomination as for the dynamic spaces, we call them *dynamic distributors* and *dynamic sub-objects*. In the following, we describe the implementation of the new features, and how they are used by means of the new API functions `PALM_Distributor_set` and `PALM_Subobject_set`.
In the legacy OpenPALM version 4.1.4, the dynamic spaces feature already existed, but with the restriction to be used only for non-distributed objects. Moreover, all distributors and sub-objects must have been defined at compile time, and could not anymore be changed at runtime. The new features overcome these restrictions through several modifications of the relevant mechanisms. In particular, it is not anymore necessary to define all distributors and sub-objects at compile time. All spaces, distributors and sub-objects which are already known at compile time of the OpenPALM application can be defined in the ID cards of the units, in PrePALM, or through dedicated functions which are implemented for the purpose of returning a distributor or sub-object definition, as it was already the case in the legacy version. This makes the new developments compatible with existing applications. But with the new version, it is also possible to only declare distributors and sub-objects without defining them, i.e. to only signalize their existence without providing a concrete distribution or a concrete sub-object definition. The definition can be given later at runtime through the API functions `PALM_Distributor_set` and `PALM_Subobject_set`, respectively. This new feature is particularly useful in the case when a distribution or sub-object cannot be known at compile time, e.g. because a parallel unit uses a domain decomposition which is determined only during runtime. In the legacy OpenPALM version, one needed to start the unit, memorize its data distribution and sub-objects, stop the application, provide the distribution and sub-object definitions, recompile the OpenPALM application, and finally run it. Even worse, as soon as any distribution or sub-object changed, it was necessary to repeat these steps. This could happen frequently, e.g. when using a different

number of processes or a different computational grid. With the new version, one can save this effort and provide the distributions and sub-objects only when they are known during runtime. Of course, all affected distributors and sub-objects must be defined before a communication can be done. But this is no restriction in general, because it would be meaningless for a unit to communicate without knowing its data distribution and sub-objects. The necessary information would naturally be available in the units before they send or receive data.
Another modification of the mechanism is the introduction of a *dynamicity flag*. Each distributor and sub-object has an individual flag which can either hold the value `static` or `dynamic`. The value `static` means that the corresponding distributor or sub-object cannot be changed anymore after it has been defined for the first time, which might be at compile time or later at runtime. The value `dynamic` indicates that the corresponding distributor or sub-object is allowed to be changed an arbitrary number of times. We introduced this flag for a performance reason. It allows to skip the request for possible updates on distributors or sub-objects if they are `static`. All dynamicity flags must be set at compile time in the ID cards or in PrePALM, and they cannot be altered during runtime. This is no restriction, because the user would usually know whether a unit needs to change a distributor or sub-object several times, or if it will stay fixed once it has been defined. Even if this is not known at compile time, one can set the flag to `dynamic` to not restrict the unit.
The driver as well as all processes of all entities hold lists of the spaces, of the distributors and of the sub-objects. These lists are used to keep track of all space, distributor and sub-object definitions. Whenever a space, a distributor or a sub-object changes, the new definition is appended as a new version in the corresponding list. Entities can only set new versions for their own spaces, distributors or sub-objects, and all changes are announced to the driver. This ensures that entities cannot disorder each others' lists, and that the driver is up to date with the newest versions of all lists at all times. Another consequence is that unnecessary synchronization between entities is avoided. Only when a communication event involves a dynamic space, distributor or sub-object on the remote side, the entity requests the driver for possible updates before computing the DOR.
Complementing the space, distributor and sub-object lists, we have enhanced the driver's commstate table. In addition to the functionalities described in Sec. 4.3.2, the commstate table also keeps track of the space, distributor and sub-object versions which were used in a communication event. This is necessary to check compliance between source and target sides of the affected DORs. It is also necessary to detect a mismatch situation where the source and the target side of a tube use different versions of a space, of a distributor or of a sub-object. This mismatch situation can occur in an indirect communication. When the source entity issues the `PALM_Put`, it uses the newest known versions at that time to compute the DOR. If the target entity did not yet issue the `PALM_Get`, the object is received and temporarily stored in the mailbuf. If the target entity later on changes the space, the distributor or the sub-object before it calls the `PALM_Get`, then the DOR stored in the mailbuf and the DOR in the target entity do not match. To cope with that, we have implemented a redistribution functionality which is used in target entities. In a mismatch situation, when the temporarily stored object in the mailbuf has a DOR with an outdated version of a space, a distributor or a sub-object, the target entity switches back to that outdated version to receive the object from the mailbuf. It aborts the transfer if the spaces have changed, since there is no way to transform an object from one space to another. However, if the distributor or the sub-object has changed, the object itself is still intact. Only its distribution or the sub-object selection may have changed. The target entity computes an auxiliary DOR as the intersection of the outdated distributor and sub-object version and the new distributor and sub-object version. This auxiliary DOR contains all necessary information so that the target entity can internally redistribute the object received from the mailbuf to comply with the new

distributor and sub-object.

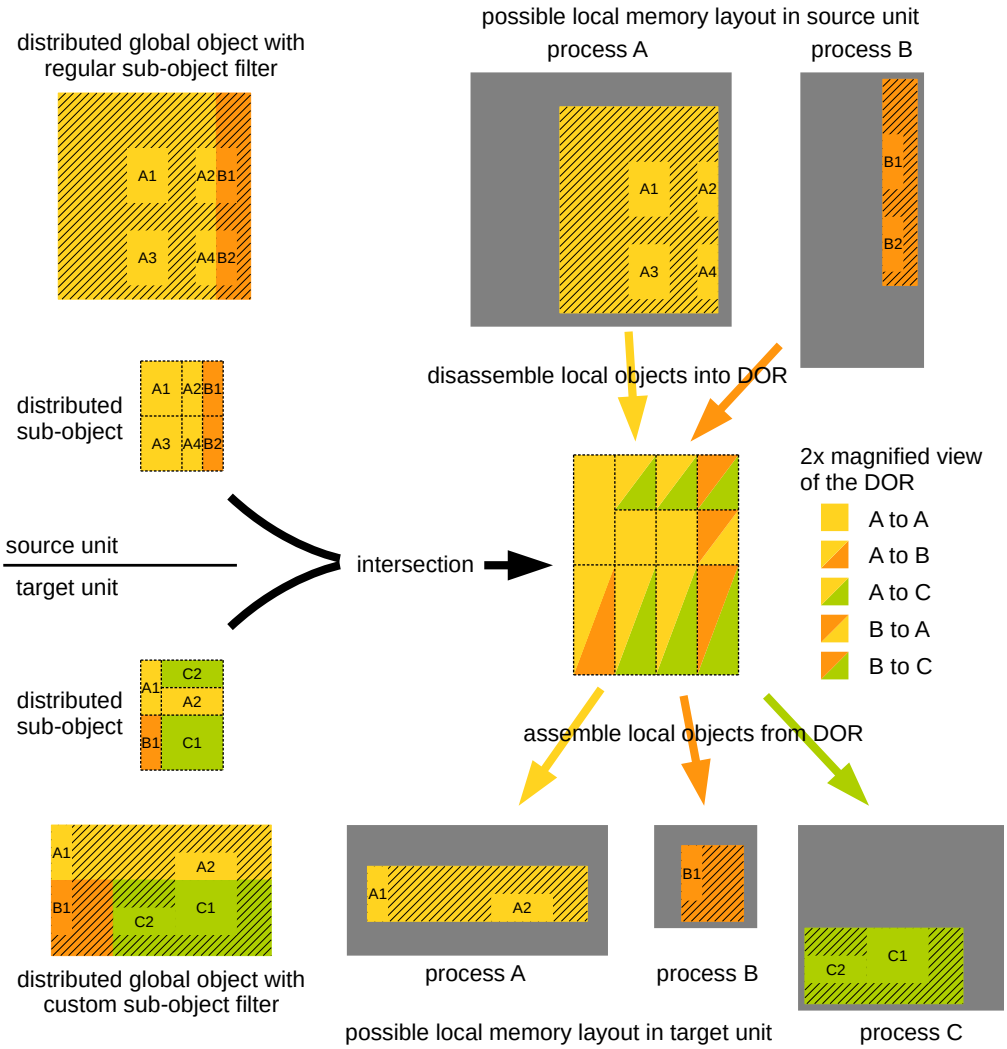


Figure 4.6: Example of distributed sub-objects. Source (top) and target (bottom) unit hold different global objects, but expose matching sub-objects. The DOR is the result of the intersection of the two distributed sub-objects. The routing table defines for each pair of source and target process the transfer of the relevant sub-object parts.

4.5.1 New API routine PALM_Distributor_set

We have developed a new API routine which allows to define distributors in an OpenPALM application during runtime. Like the `PALM_Put` and `PALM_Get` routines, this new API routine can be used in the source code of units. It takes as input the distributor name and an encoded definition of the object distribution. It installs the new distributor version in the unit where the routine is called, and it announces it to the driver. Units can only set new versions for their own distributors which are marked with the `dynamic` flag, and for `static` distributors which have

only been declared but not yet defined. The routine is described in Alg. 11.

Algorithm 11 `PALM_Distributor_set`

1: Input: distributor name, encoded definition of the new distribution.
2: Check if the calling entity uses the distributor for at least one of its objects, otherwise return.
3: Check if the calling process belongs to the localization of the distributor, otherwise return.
4: Check if the distributor is `dynamic`, or if it is `static` but still undefined, otherwise abort.
5: **if** this process has rank 0 in the associated localization **then**$^{(*)}$
6: Send a notification to the driver telling the size of the encoded definition of the new distribution.$^{(*)}$
7: Send the encoded definition to the driver.$^{(*)}$
8: **end if**
9: Append the new version in the local distributor list.
10: Call `MPI_Barrier` on the localization.$^{(*)}$

Explanatory comments on Alg. 11:
Step 5: Only the first process in the localization communicates with the driver.
Step 6: The driver's reaction to this notification is stated in Alg. 12.
Step 7: The driver receives the encoded distributor definition in step 1 in Alg. 12.
Step 10: Without this barrier, it could happen that a process other than the first process finishes setting the new distributor version and issues a communication before the first process has announced the new distributor version to the driver. This would result in a wrong mapping of the object parts between source and target. The barrier is necessary to prevent from that.

Algorithm 12 Driver reaction on the `PALM_Distributor_set` notification in step 6 of Alg. 11

1: Receive the encoded definition from the entity.$^{(*)}$
2: Append the new version in the driver's distributor list.

Explanatory comments on Alg. 12:
Step 1: The encoded distributor definition is sent in step 7 of Alg. 11.

4.5.2 New API routine PALM_Subobject_set

Similar to the `PALM_Distributor_set` routine, we have developed another new API routine which allows to define sub-objects in an OpenPALM application during runtime. Like the `PALM_Put` and `PALM_Get` routines, this new API routine can be used in the source code of units. It takes as input the sub-object name and an encoded definition of the sub-object filter. It installs the new version in the unit where the routine is called, and it announces it to the driver. Units can only set new versions for their own sub-objects which are marked with the `dynamic` flag, or for `static` sub-objects which have only been declared but not yet defined. The routine is described in Alg. 13.

Algorithm 13 `PALM_Subobject_set`

1: Input: sub-object name, encoded definition of the new sub-object filter.
2: Check if the calling entity owns the sub-object, otherwise return.
3: Check if the calling process belongs to the localization of the object using this sub-object, otherwise return.
4: Check if the sub-object is `dynamic`, or if it is `static` but still undefined, otherwise abort.
5: **if** this process has rank 0 in the associated localization **then**$^{(*)}$
6: Send a notification to the driver telling the size of the encoded definition of the new sub-object.$^{(*)}$
7: Send the encoded definition to the driver.$^{(*)}$
8: **end if**
9: Append the new version in the local sub-object list.
10: Call `MPI_Barrier` on the localization.$^{(*)}$

Explanatory comments on Alg. 13:
Step 5: Only the first process in the localization communicates with the driver.
Step 6: The driver's reaction to this notification is stated in Alg. 14.
Step 7: The driver receives the encoded sub-object definition in step 1 in Alg. 14.
Step 10: Without this barrier, it could happen that a process finishes setting the new sub-object version and issues a communication before the first process has announced the new sub-object version to the driver. This would result in a wrong mapping of the object parts between source and target. The barrier is necessary to prevent from that.

Algorithm 14 Driver reaction on the `PALM_Subobject_set` notification in step 6 of Alg. 13

1: Receive the encoded definition from the entity.$^{(*)}$
2: Append the new version in the driver's sub-object list.

Explanatory comments on Alg. 14:
Step 1: The encoded sub-object definition is sent in step 7 of Alg. 13.

4.5.3 Enhanced API routines `PALM_Put` and `PALM_Get`

In this subsection, we present the enhanced data transfer mechanism which is able to use dynamic spaces, dynamic distributors and dynamic sub-objects. We state the enhanced `PALM_Put` and `PALM_Get` routines in Algs. 15 and 16, respectively, and the enhanced implementations of the driver's reactions to these communication routines in Algs. 18 to 21. Some steps of the algorithms are marked with an asterisk indicating that an explanatory comment is given below.

Algorithm 15 Enhanced `PALM_Put`

```
1: Input: space name, object name, time, tag, pointer to local memory where the local object is stored.
2: Determine the comids which are relevant for this call to PALM_Put.(*)
3: if at least one of the affected spaces, distributors or sub-objects is dynamic, or static but still undefined then
4:     Send an update notification to the driver.(*)
5:     Send the last known versions of the spaces, distributors and sub-objects to the driver.(*)
6:     Receive the list of available updates from the driver.(*)
7:     for each new space, distributor or sub-object version do
8:         Receive the definition of the new space, distributor or sub-object version from the driver.(*)
9:         Append the new version in the corresponding local space, distributor or sub-object list.
10:    end for
11: end if
12: Set flag = 0.
13: for each comid do
14:    if this is a non-optimized communication then
15:        if flag == 0 then
16:            Notify the PALM_Put to the driver.(*)
17:            for each relevant comid do
18:                Receive information from the driver whether this is a new object version.(*)
19:                if this is the first process which contributes a part to the object then
20:                    Determine the newest available distributor and sub-object versions for the target entity.
21:                    if source and newest target sub-object do not comply then
22:                        Determine the sub-object which this source process has last used for the target.
23:                        if source and last used target sub-object do not comply then
24:                            Use the IDENTITY sub-object for the target.
25:                        end if
26:                    end if
27:                    if source and newest target distributor do not comply then
28:                        Determine the distributor which this process has last used for the target.
29:                        if source and last used target distributor do not comply then
30:                            Use the SINGLE_PROC distributor for the target.
31:                        end if
32:                    end if
33:                    Send the chosen distributor and sub-object versions to the driver.(*)
34:                else
35:                    Receive the chosen distributor and sub-object versions from the driver.(*)
36:                end if
37:            end for
38:            Receive the driver's answer.(*)
39:            Set flag = 1.(*)
40:        end if
41:        Determine rank and current shape of the space.
42:        if the source distributor or sub-object have changed since the last communication event then
43:            Compute the intersection of the source distributor and sub-object.
44:        end if
```

```
45:         if the target distributor or sub-object have changed since the last communication
            event then
46:             Compute the intersection of the target distributor and sub-object.
47:         end if
48:         if there is a new source or target distributor or sub-object version, or the source space
            has changed since the last communication event then
49:             if there is an old DOR then
50:                 Delete the old DOR.
51:             end if
52:             Compute the new DOR.
53:         end if
54:         Allocate temporary memory for the DOR parts and disassemble the local object.
55:         Determine which target processes shall receive a part of the disassembled local object
            from this source process.
56:         for each process of the target's distributor do
57:             if that target process shall receive a part then
58:                 if the object shall be written to a file then
59:                     Write the object part to the file.
60:                 else
61:                     Send a connection request to the driver.(*)
62:                     Receive a connection authorization from the driver.(*)
63:                     Extract the communication type, the receiver entity type, and the receiver
                        MPI process rank from the drivers answer.
64:                     if the receiver is a unit or a memory slave then
65:                         Create an MPI intercommunicator to the receiver process.
66:                     end if
67:                     Send the object part to the receiver process.(*)
68:                     if the receiver is a unit or a memory slave then
69:                         Delete the intercommunicator to the receiver process.
70:                     end if
71:                 end if
72:             end if
73:             Notify the completion of the transfer of this object part to the driver.
74:         end for
75:         Deallocate the temporary memory for the DOR parts.
76:     else this is an optimized communication
77:         Get the MPI intracommunicator for this object.
78:         Determine rank and current shape of the space.
79:         if the source distributor or sub-object have changed since the last communication
            event then
80:             Compute the intersection of the source distributor and sub-object.
81:         end if
82:         if the target distributor or sub-object have changed since the last communication
            event then
83:             Compute the intersection of the target distributor and sub-object.
84:         end if
```

85: **if** there is a new source or target distributor or sub-object version, or the source space has changed since the last communication event **then**
86: **if** there is an old DOR **then**
87: Delete the old DOR.
88: **end if**
89: Compute the new DOR.
90: **end if**
91: Allocate temporary memory for the DOR parts and disassemble the local object.
92: **for** each process of the target distributor **do**
93: **if** that target process shall receive a part **then**
94: Create an MPI intercommunicator to the receiver process.
95: Send the object part to the receiver process.(*)
96: Delete the intercommunicator to the receiver process.
97: Call `MPI_Barrier` on the local process group of this object's MPI intra-communicator.(*)
98: **end if**
99: **end for**
100: Deallocate the temporary memory for the DOR parts.
101: **end if**
102: **end for**

Explanatory comments on Alg. 15:
Step 2: Note that the calling EOS might be the source in several tubes. For each affected tube, the time-tag and localization matching is done, and the corresponding comids are determined. The rank of the calling process with respect to the source distributor is derived from the localization.
Step 4: The driver's reaction to this notification is stated in Alg. 17.
Step 5: The driver receives the last known versions in step 1 of Alg. 17.
Step 6: The driver sends the list of available updates in step 3 of Alg. 17.
Step 8: The driver sends the new versions in step 5 of Alg. 17.
Step 16: The driver's reaction to this notification is stated in Alg. 18.
Step 18: This information is sent by the driver in step 4 of Alg. 18.
Step 33: The chosen versions are received by the driver in step 6 of Alg. 18.
Step 35: The chosen versions are sent by the driver in step 8 of Alg. 18.
Step 38: The driver's answer is given in step 27 of Alg. 18.
Step 39: The flag is used to ensure that the driver is notified only once, even if there are multiple comids. The driver can itself determine all relevant comids.
Step 61: The driver's reaction to this connection request is stated in Alg. 19.
Step 62: The driver's answer is given in steps 3, 8 or 24 of Alg. 19.
Step 67: The object part is received in step 48 of Alg. 16 in case of a direct communication, or in step 18 or 20 of Alg. 19 in case of an indirect communication, or in step 12 or 14 of Alg. 19 in case of a buffer communication.
Step 95: The object part is received in step 96 of Alg. 16.
Step 97: The MPI barrier is called on the local process group, i.e. only on the source side. This is necessary to avoid race conditions where several source processes could send object parts to the same target process for the same communication event.

Algorithm 16 Enhanced `PALM_Get`

```
1:  Input: space name, object name, time, tag, pointer to local memory where the local object
    shall be stored after reception.
2:  Check if there is a comid for this call to PALM_Get.
3:  if there is a comid then
4:      if this is a non-optimized communication then
5:          Notify the PALM_Get to the driver.(*)
6:          Receive the driver's answer.(*)
7:          Send the last known versions of the spaces, distributors and sub-objects to the
            driver.(*)
8:          Receive the list of available updates from the driver.(*)
9:          for each new space, distributor or sub-object version do
10:             Receive the definition of the new space, distributor or sub-object version from the
                driver.(*)
11:             Append the new version in the corresponding local space, distributor or sub-object
                list.
12:         end for
13:         Receive the versions of the source and target distributors and sub-objects which the
            sender has used for this object.(*)
14:         Check local mail buffer flag (*)
15:         Determine rank and current shape of the space.
16:         if the source distributor or sub-object have changed since the last communication
            event then
17:             Compute the intersection of the source distributor and sub-object.
18:         end if
19:         if there is a new target distributor or sub-object version since the last communication
            event, or the source entity did not use the current target distributor and
            sub-object then
20:             Compute the intersection of the target distributor and sub-object.
21:         end if
22:         if there is a new source or target distributor or sub-object version, or the source space
            has changed since the last communication event then
23:             if there is an old DOR then
24:                 Delete the old DOR.
25:             end if
26:             Compute a new DOR as the intersection of the source distributor and sub-object
                and of target distributor and sub-object which the source entity has used.
27:         end if
28:         Get the number of source object parts for this target process.
29:         if the object shall be read from a file then
30:             Read local object parts from file.
31:         end if
32:         Receive information from the driver where the object parts are located.(*)
33:         Determine if the object must be redistributed within the target entity after reception.
34:         if the object must be redistributed after reception then
35:             Allocate temporary memory for the local object redistribution.
36:         end if
37:         Allocate memory for the DOR parts.
```

38: **for** each source process **do**
39: **if** that source process contributes an object part **then**
40: Send a connection request to the driver.(*)
41: Check if an MPI intercommunicator is needed.(*)
42: **if** an MPI intercommunicator is needed **then**
43: Create an MPI intercommunicator to the sender process.
44: **end if**
45: **if** the object must be redistributed after reception **then**
46: Receive the object part from the sender process in the temporary memory.(*)
47: **else**
48: Receive the object part from the sender process in the local object memory.(*)
49: **end if**
50: **if** an MPI intercommunicator was needed **then**
51: Delete the MPI intercommunicator to the sender process.
52: **end if**
53: **end if**
54: **end for**
55: **if** the object must be redistributed **then**
56: Compute a temporary DOR as the intersection between the outdated target distributor and sub-object versions which the source entity has used, and the newest target distributor and sub-object versions.
57: Assemble the local object from the DOR which the source entity has used into the temporary memory.
58: Disassemble the local object from the temporary memory into the source side of the temporary DOR.
59: **for** each process of this entity which shall receive an object part from this process in the internal redistribution **do**
60: Call the non-blocking `MPI_Isend` to send the object part to that process.
61: **end for**
62: **for** each process of this entity which shall send an object part to this process in the internal redistribution **do**
63: Call the non-blocking `MPI_Irecv` to receive the object part from that process.
64: **end for**
65: Call `MPI_Waitall` to wait for the completion of all send and receive operations.
66: Assemble the local object from the target side of the temporary DOR into the local object memory.
67: Delete the temporary DOR and deallocate the temporary memory.
68: **else**
69: Assemble the local object from the target side of the DOR into the local object memory.
70: **end if**
71: Send a notification to the driver telling that this process has finished the `PALM_Get`.

```
72:     else this is an optimized communication
73:         Determine whether any affected space, distributor or sub-object might require an
            update.
74:         if an update is required then
75:             Send an update notification to the driver.(*)
76:             Send the last known space, distributor and sub-object versions to the driver.(*)
77:             Receive the list of available updates from the driver.(*)
78:             for each available update do
79:                 Receive the new version from the driver and append it to the corresponding
                    space, distributor or sub-object list.(*)
80:             end for
81:             Unpack all updates into the corresponding version lists of the affected spaces,
                distributors and sub-objects.
82:         end if
83:         Determine rank and current shape of the space.
84:         Determine whether a new DOR must be computed.
85:         if a new DOR must be computed then
86:             if there exists already an old DOR then
87:                 Delete the old DOR.
88:             end if
89:             Compute the new DOR.
90:         end if
91:         Get the number of source DOR parts for this target process.
92:         Allocate temporary memory for the DOR parts if necessary.
93:         for each source process do
94:             if that source process contributes a DOR part then
95:                 Create an MPI intercommunicator to the sender process.
96:                 Receive the object part from the sender process.(*)
97:                 Delete the MPI intercommunicator to the sender process.
98:             end if
99:         end for
100:         Assemble the local object from the DOR.
101:     end if
102: end if
```

Explanatory comments on Alg. 16:
Step 5: The driver's reaction to this notification is stated in Alg. 20.
Step 6: The driver's answer is given in step 17 of Alg. 18 or in step 13 or 22 of Alg. 20.
Step 7: The driver receives the last known versions in step 18 of Alg. 18 or in step 23 of Alg. 20.
Step 8: The driver sends the list of available updates in step 19 of Alg. 18 or in step 24 of Alg. 20.
Step 10: The driver sends the new version in step 21 of Alg. 18 or in step 26 of Alg. 20.
Step 13: The driver sends the chosen versions in step 23 of Alg. 18 or in step 28 of Alg. 20.
Step 14: Units which are integrated into the same block can use local mail buffer memory to exchange data instead of using the driver's mail buffer.
Step 32: The object part locations are sent by the driver in step 24 of Alg. 18 in case of a direct communication, or in step 29 of Alg. 20 in case of an indirect or buffer communication.

Step 40: The driver's reaction to this connection request is stated in Alg. 21.
Step 41: An MPI intercommunicator needs to be created if the sender is not the driver, to which an intercommunicator exists anyway, and when the local mail buffer is not used.
Step 46: The object part is sent in step 10 or 12 of Alg. 21.
Step 48: The object part is sent in step 67 of Alg. 15 in case of a direct communication, or in step 10 or 12 of Alg. 21 in case of an indirect communication, or in step 3 or 5 of Alg. 21 in case of a buffer communication.
Step 75: The driver's reaction to this notification is stated in Alg. 17.
Step 76: The driver receives the last known versions in step 1 of Alg. 17.
Step 77: The driver sends the list of available updates in step 3 of Alg. 17.
Step 79: The driver sends the new versions in step 5 of Alg. 17.
Steps 96: The object part is sent in step 95 of Alg. 15.

Algorithm 17 Driver reaction on the space, distributor or sub-object update notification in step 4 of Alg. 15 or in step 75 of Alg. 16

```
1: Receive the last known versions of the spaces, distributors and sub-objects from the entity.(*)
2: Compare the last known versions of the entity with the newest available versions to determine
   the necessary updates.
3: Send the list of available updates to the entity.(*)
4: for each available space, distributor or sub-object update do
5:     Send the definition of the new version to the entity.(*)
6: end for
```

Explanatory comments on Alg. 17:
Step 1: The last known versions are sent in step 5 of Alg. 15 or in step 76 of Alg. 16.
Step 3: The list of available updates is received in step 6 of Alg. 15 or in step 77 of Alg. 16.
Step 5: The definition of the new version is received in step 8 of Alg. 15 or in step 79 of Alg. 16.

Algorithm 18 Enhanced driver reaction on the `PALM_Put` notification in step 16 of Alg. 15

```
1: Determine the comids which are relevant for this call to PALM_Put.
2: for each comid do
3:     Insert this PALM_Put into the commstate table.
4:     Send information to the source process whether this is a new object version.(*)
5:     if if this source process is the first process for this PALM_Put then
6:         Receive the space, distributor and sub-object versions which the source process uses
           for this PALM_Put, and store them in the commstate table.(*)
7:     else
8:         Read the space, distributor and sub-object versions which the first process used for
           this PALM_Put from the commstate table, and send them to the source process.(*)
9:     end if
10:    Search the commstate table for a waiting PALM_Get matching this comid.
11:    if there is a waiting PALM_Get then
12:        Update the commstate table that the PALM_Get will be served by this PALM_Put.
13:        if the source is the buffer then
14:            Determine if the object is located at the driver or a memory slave.
15:        end if
```

16: **for** each target process **do**
17: Send answer to target process.(*)
18: Receive the last known space, distributor and sub-object versions from this target process.(*)
19: Determine if updates are available for this target process, and send the list of available updates to the target process.(*)
20: **for** each available update **do**
21: Send the new version to the target process.(*)
22: **end for**
23: Read the space, distributor and sub-object versions which the source entity uses for this `PALM_Put` from the commstate table, and send them to this target process.(*)
24: Send object part locations to this target process.(*)
25: **end for**
26: **end if**
27: Send answer to the source process.(*)
28: **end for**

Explanatory comments on Alg. 18:
Step 4: This information is received by the source process in step 18 of Alg. 15.
Step 6: The chosen versions for the spaces, distributors and sub-objects are send in step 33 of Alg. 15.
Step 8: The chosen versions for the spaces, distributors and sub-objects are received in step 35 of Alg. 15.
Step 17: This answer is received in step 6 of Alg. 16.
Step 18: The last known versions are sent in step 7 of Alg. 16.
Step 19: The update list is received in step 8 of Alg. 16.
Step 21: The new version is received in step 10 of Alg. 16.
Step 23: The chosen versions for the space, distributors and sub-objects are received in step 13 of Alg. 16.
Step 24: The object part locations are received in step 32 of Alg. 16.
Step 27: This answer is received in step 38 of Alg. 15.

Algorithm 19 Enhanced driver reaction on the `PALM_Put` connection request in step 61 of Alg. 15

1: Query the commstate table if this `PALM_Put` matches a waiting `PALM_Get`.
2: **if** a `PALM_Get` is waiting for the connection **then**
3: Send a connection authorization to the source process telling that the communication type is direct, the target is a unit, and the MPI rank of the receiver process.(*)
4: **else** the target is the buffer, or the object must be temporarily stored in the mailbuf
5: Determine if the target is the buffer, or if the mailbuf shall take the object part.
6: Determine if the driver or a memory slave shall receive the object.
7: **if** the driver receives the object part **then**
8: Send a connection authorization to the source process telling whether the communication type is buffer or indirect, the receiver is the driver, and the MPI rank of the driver.(*)
9: **end if**

10: **if** the target is the buffer **then**
11: **if** the driver receives the object part **then**
12: Receive the object part from the source process and put it in the buffer.(*)
13: **else**
14: Order a memory slave to receive the object part and to put it in the buffer.(*)
15: **end if**
16: **else**
17: **if** the driver receives the object part **then**
18: Receive the object part from the source process and put it in the mailbuf.(*)
19: **else**
20: Order a memory slave to receive the object part and to put it in the mailbuf.(*)
21: **end if**
22: **end if**
23: **if** a memory slave receives the object **then**
24: Send a connection authorization to the source process telling whether the communication type is buffer or indirect, the receiver is a memory slave, and the MPI rank of the memory slave.(*)
25: **end if**
26: **end if**

Explanatory comments on Alg. 19:
Steps 3, 8 and 24: This answer is received in step 62 of Alg. 15.
Steps 12, 14, 18 and 20: The object part is sent in step 67 of Alg. 15.

Algorithm 20 Enhanced driver reaction on the `PALM_Get` notification in step 5 of Alg. 16

1: Determine the comids which are relevant for this call to `PALM_Get`.
2: Query the commstate table to choose a comid.
3: **if** this `PALM_Get` does not read from a file **then**
4: **if** not all processes of the target entity have announced this `PALM_Get` yet **then**
5: Return.(*)
6: **end if**
7: **if** there is neither a matching `PALM_Put` currently active nor is the object available in the mailbuf or buffer **then**
8: Memorize the `PALM_Get` to be wakened later in step 11 of Alg. 18 when a matching `PALM_Put` occurs.
9: Return.(*)
10: **end if**
11: **else** the object is read from a file
12: **for** each target process **do**
13: Send a dummy answer to the target process.(*)
14: **end for**
15: **end if**
16: **for** each target process **do**
17: **for** each source process **do**
18: **if** the object part is in the mailbuf **then**
19: Determine whether the object part is stored on the driver or a memory slave.
20: **end if**
21: **end for**

22: Send an answer to the target process telling the chosen comid for this `PALM_Get`.(*)
23: Receive the last known space, distributor and sub-object versions from this target process.(*)
24: Determine if updates are available and send the list of available updates to the target process.(*)
25: **for** each available update **do**
26: Send the new version to the target process.(*)
27: **end for**
28: Read the space, distributor and sub-object versions which the source entity uses for this `PALM_Put` from the commstate table, and send them to this target process.(*)
29: Send the object part locations to the target process.(*)
30: **end for**

Explanatory comments on Alg. 20:
Step 5: The driver only continues to serve this `PALM_Get` when all processes of the target entity have announced it. Since the target processes wait for the answer in step 6 of Alg. 16, this implies a synchronization among the target processes.
Step 9: Since the target processes are waiting for the answer in step 6 of Alg. 16, this makes the target wait until a matching `PALM_Put` occurs.
Step 13: This is necessary to let the target entity continue since it is waiting for the answer in step 6 of Alg. 16.
Step 22: This answer is received in step 6 in Alg. 16.
Step 23: The last known versions are sent in step 7 of Alg. 16.
Step 24: The update list is received in step 8 of Alg. 16.
Step 26: The new version is received in step 10 of Alg. 16.
Step 28: The chosen versions are received in step 13 of Alg. 16.
Step 29: The object part locations are received in step 32 in Alg. 16.

Algorithm 21 Enhanced driver reaction on the `PALM_Get` connection request in step 40 of Alg. 16

1: **if** the source is the buffer **then**
2: **if** the object is stored in the driver **then**
3: Send the object part to the target process.(*)
4: **else**
5: Determine which memory slave has the object and order it to send the object part to the target process.(*)
6: **end if**
7: **else** the source is not the buffer
8: **if** the object part is served from the mailbuf **then**
9: **if** the object is stored on the driver **then**
10: Send the object part to the target process.(*)
11: **else**
12: Determine which memory slave has the object and order it to send the object part to the target process.(*)
13: **end if**
14: **end if**
15: **end if**

Explanatory comments on Alg. 21:
Steps 3, 5, 10 and 12: The object part is received in step 46 or 48 of Alg. 16.

4.6 Realization of the concurrent operator splitting scheme using OpenPALM

In this section, we review our work [107] on the implementation of the concurrent operator splitting scheme for the natural convection scenario presented in Sec. 3. Our goal is to illustrate the use of OpenPALM to realize the concurrent operator splitting. This allows to actually propagate the fluid model and the temperature model concurrently while synchronizing only at global time steps. Both models are internally parallelized by means of individual domain decompositions on separate sets of processes. This allows to adapt the allocation of computing resources for the models to meet their individual computational demands. We recapitulate the performance tests conducted in our work [107] showing that the concurrent operator splitting can yield superior parallel efficiency compared to a monolithic solution scheme.
Figure 4.7 shows a schematic of the realization of the concurrent operator splitting using OpenPALM. The coupling results in a multiple program, multiple data (MPMD) setup where the two

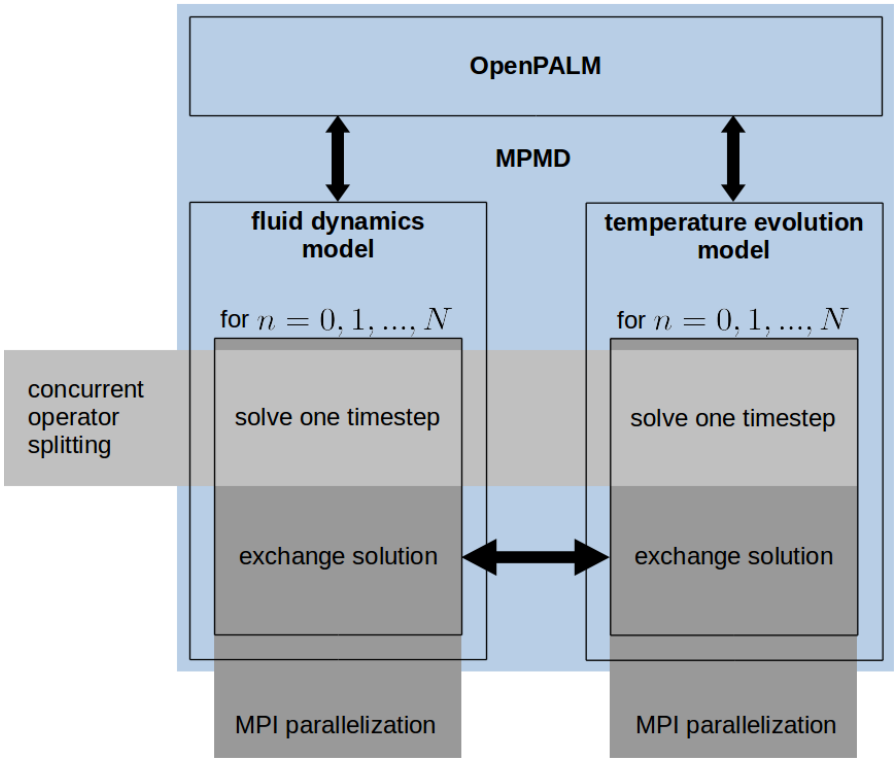
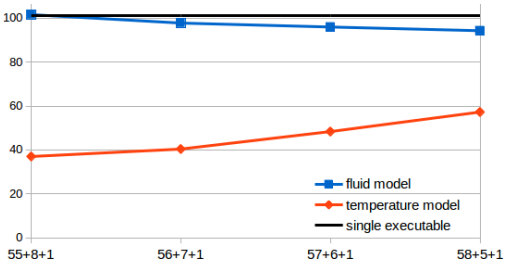
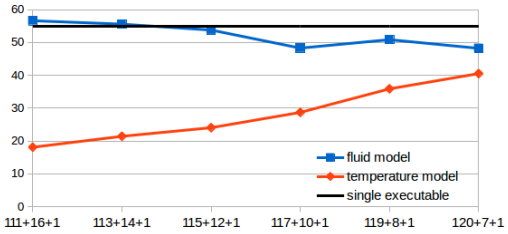


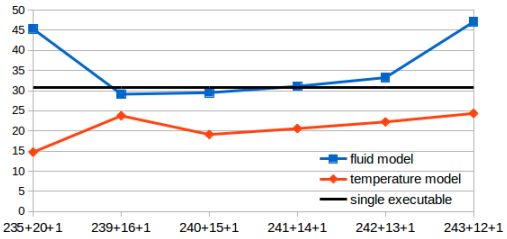
Figure 4.7: Schematic of the realization of the concurrent operator splitting using OpenPALM. The coupling results in a multiple program, multiple data (MPMD) setup where the two models and the OpenPALM driver are each compiled into their own executable and run on separate sets of processes. Both models are internally parallelized using individual domain decompositions. Figure taken from [107].



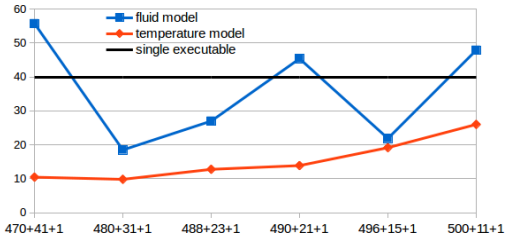
(a) 64 processes in total.



(b) 128 processes in total.



(c) 256 processes in total.



(d) 512 processes in total.

Figure 4.8: Runtime per computed time step [sec] for the fluid model (blue) and the temperature model (red) in the concurrent operator splitting scheme with varying parallel configurations, and of the monolithic solution scheme (black) for comparison. The parallel configurations are indicated as $a+b+1$ where a and b are the number of processes of the fluid and of the temperature model, respectively, plus one OpenPALM driver process. The total number of processes is kept constant within each test series. Figures taken from [107].

models and the OpenPALM driver are each compiled into their own executable.

We performed runtime performance tests with varying parallel configurations. Fig. 4.8 shows the runtime results for computing one time step using the concurrent operator splitting scheme with varying parallel configurations. The parallel configuration is indicated as $a+b+1$ where a and b are the number of processes used for the fluid model and for the temperature model, respectively, plus one OpenPALM driver process. The total number of processes is kept constant within each of the test series using $a+b+1 = 64, 128, 256, 512$. The fluid model runtime is indicated in blue, and the temperature model runtime in red. For comparison, we indicate the monolithic scheme's runtime using the total number of processes in black. In all cases, the time needed for data transfer between the models in the concurrent operator splitting tests was negligible. Therefore, the global runtime of concurrent operator splitting computations was equal to the runtime of the slower model. The computational demand of the fluid model is much larger than that of the temperature model. This is due to the nonlinearity of the fluid model, and due to the larger individual model problem size. Therefore, we used considerably more processes for the fluid model than for the temperature model in the concurrent scheme. In all four test series we achieved smaller runtimes for the concurrent operator splitting scheme than for the monolithic scheme by appropriately allocating processes to the models. For the parallel configurations using 64 processes in total (Fig. 4.8a) and 256 processes in total (Fig. 4.8c), the runtime benefit was only moderate. The 128 processes in total configuration (Fig. 4.8b) yielded considerable speedup of the concurrent over the monolithic scheme. Remarkably, the 512 processes in total concurrent

configuration (Fig. 4.8d) yielded tremendous speedup over the monolithic scheme. However, the concurrent tests showed non-monotone behavior of the model runtimes, and in particular for the fluid model, when shifting processes between the models. The reason is that the efficiency of the block-Jacobi incomplete LU decomposition preconditioner used for the GMRES linear solver [93] depends in a non-monotone way on the domain decomposition of the models. This evoked the variation of the runtime results. Nevertheless, we proved that the inherent parallelism of the concurrent operator splitting scheme on the level of the coupling algorithm can be exploited to improve the overall parallel performance compared to monolithic schemes. Furthermore, we demonstrated that OpenPALM is a suitable tool to implement the concurrent scheme using individually parallelized models.

5 Biogeochemistry-hydrology coupling for nutrient cycle simulations

This chapter is dedicated to the modeling and implementation of the biogeochemistry-hydrology coupling which we use for nutrient cycle simulations. First, we outline the role of nutrients and related biogeochemical processes in ecosystems. As indicated in Chap. 1, we regard those chemical elements as nutrients which are essential for the growth of living organisms. Carbon (C), oxygen (O), hydrogen (H) and nitrogen (N) are needed in the largest portion among all nutrients [32, 43]. The predominant appearance of these nutrients is in gaseous form like molecular nitrogen (N_2) or carbon dioxide (CO_2), in mineral form, in inorganic ion form like ammonium (NH_4^+) or nitrate (NO_3^-), and in organic form as part of diverse N and C compounds. Nutrient cycling denotes the movement of nutrients between the compartments of ecosystems and their transformation by geochemical and biological processes [78]. C and N cycles in agricultural soils supply nutrients for vegetation and crop growth, and affect groundwater and surface water eutrophication, and greenhouse gas (GHG) emissions such as nitrous oxide (N_2O), CO_2, and methane (CH_4). Among these GHG emitted from soils, N_2O has by far the largest 100-year global warming potential (GWP-100) of 265, while CH_4 has a GWP-100 of 28 [33]. The global warming potential is an indicator for the capacity of a chemical compound to retain heat in the atmosphere. It is defined relative to CO_2, whose GWP is set to 1. Nitrous oxide is an intermediate product of the denitrification pathway of the nitrogen cycle [69, 23]. Denitrification denotes the reduction of nitrate and nitrite (NO_2^-) to molecular nitrogen by microbes, with the intermediate products of nitric oxide (NO) and N_2O. Denitrification is closely connected to the nitrification pathway, which denotes the oxidation of ammonia (NH_3) and NH_4^+ to NO_3^- by microbes, with the intermediate products of hydroxylamine (NH_2OH) and NO_2^-. The main processes contributing to the nitrification pathway

$$NH_3/NH_4^+ \quad \rightarrow \quad NH_2OH \quad \rightarrow \quad NO_2^- \quad \rightarrow \quad NO_3^- \,,$$

are heterotrophic nitrification, which includes the conversion of organic N compounds to NH_3/NH_4^+, autotrophic ammonia oxidation and autotrophic nitrite oxidation. The main processes contributing to the denitrification pathway

$$NO_3^- \quad \rightarrow \quad NO_2^- \quad \rightarrow \quad NO \quad \rightarrow \quad N_2O \quad \rightarrow \quad N_2 \,,$$

are coupled and distinct nitrification-denitrification, anaerobic or micro-aerobic denitrification, co-denitrification with NO/N_2O, and nitrate ammonification [14]. Nitrification requires aerobic conditions, i.e. the presence of O_2, since it uses oxygen as electron acceptor for oxidizing the nitrogen. In contrast, the denitrification processes stated above occur under anaerobic conditions, i.e. in absence of O_2. Some microbes use denitrification for respiration under anaerobic conditions, where nitrogen is reduced and thus takes the role of the electron acceptor instead of oxygen. Another important process that can occur under anaerobic conditions is methanogenesis, i.e. the microbial formation of CH_4 by decomposition of organic matter with the intermediate product CO_2 [23], e.g. the decomposition of acetic acid on the pathway

$$CH_3COOH \quad \rightarrow \quad CO_2 \quad \rightarrow \quad CH_4 \quad .$$

This brief outline of some pathways contributing to the carbon and nitrogen cycle in terrestrial ecosystems indicates the complexity of the biological, geochemical and hydrological processes involved in production, consumption and transport of nutrients in soils, and the fluxes of nutrients between compartments. Greenhouse gases such as N_2O and CH_4 are produced in soils mainly as intermediate or final result of microbial activity in de-/nitrification and de-/composition processes. Furthermore, inorganic nutrient forms such as NO_3^- play an important role since plant uptake of nutrients is mostly only possible in ion form [78]. However, excess nitrogen availability in soils can result in ground water and surface water eutrophication due to nitrate leaching [83], and in increased emissions of nitric oxide and nitrous oxide. In fact, agricultural soils have been identified as major anthropogenic sources of surplus nitrate input to water bodies, of N_2O greenhouse gas emissions, and also as a considerable source of CH_4 emissions. Yet, the quantification of these effects is difficult. Measurement techniques used to assess trace gas exchange between soil and atmosphere are the traditional closed chamber method, an advanced chamber technique called fast box, and eddy covariance (EC) techniques in conjunction with tunable diode laser or quantum cascade laser spectrometers [14]. Closed chambers are easy to use and inexpensive, but only cover a small site. Spatial and temporal variability is hard to assess with traditional closed chambers due to their long measurement cycles of 30-60 minutes. Fast box measurements can be taken in much shorter cycles of seconds allowing for good temporal resolution. Spatial coverage can be improved with the fast box method by relocating it between measurements, which is however manpower intensive. Eddy covariance techniques are able to measure N_2O fluxes for areas larger than 1 hectare (ha) [105]. A combination of chamber and EC measurements can be used to obtain both the landscape-scale trace gas fluxes for estimating emissions and local measurement data to study site-specific processes [14]. However, indirect effects due to the transport of nutrients are hardly assessable with measurements. Such indirect effects may appear with large spatial and temporal extent, but they may nevertheless contribute significantly to GHG and nutrient budgets. Furthermore, the strong influence on the nutrient cycles exerted by environmental factors such as temperature and moisture, by soil and vegetation properties, and by land management can result in a high variability of microbial activity. This gives rise to *"hot spots"* and *"hot moments"*, i.e. events of greenhouse gas emissions or nutrient leaching which are highly localized with respect to space and time.

Understanding the complex interactions of biogeochemical processes, assessing nutrient fluxes on various spatial and temporal scales, estimating impacts of current and projected future land use, land management and climate conditions, and developing mitigation strategies are primary goals in ecosystem research. As indicated above, field measurement techniques are in many situations useful for investigations. However, they can be cost and manpower intensive, and limited in spatial and temporal coverage and resolution. For these reasons, it is often difficult to capture indirect effects, hot spots and hot moments with measurements. In particular, tracing these effects back to determine their origin and evolution is hardly possible. Furthermore, prediction of ecosystem feedback on changes in land use, land management and climate, and the evaluation of mitigation strategies to reduce greenhouse gas emissions and water eutrophication are often out of the scope of measurements.

Modeling and numerical simulation have been proved to be appropriate approaches to address these issues. Considerable research has been dedicated to develop biogeochemical process models with varying scope and complexity [40, 17, 91, 87, 69, 59, 50, 34, 36]. Traditional modeling approaches focus on the nitrogen and carbon cycle and related microbial activity. However, biogeochemical models are typically one-dimensional, representing a single site soil column. Lateral fluxes are ignored in such models, although they contribute significantly to nutrient cycles. In

particular, topographical effects on hydrology and lateral transport of nutrients are neglected. Efforts to overcome these shortcomings include the extension of existing hydrological models with simple biogeochemical process descriptions as e.g. in [56, 29], or one-way coupling of hydrology models to biogeochemical models, e.g. [47]. Although aforementioned works present progress in simulating nutrient cycles and the related biogeochemical and hydrological processes in terrestrial ecosystems, it still remains a challenging task. All four issues related to multiphysics simulations which we identified in the introduction are involved. First, the models need to resolve short-term processes like surface water runoff, infiltration or greenhouse gas emission hot spots on the scale of days, hours or even minutes, as well as long-term processes like emission and nutrient inventories on the scale of years or even decades. Second, hydrological models are often given in terms of partial differential equations, e.g. by means of the Richards equation for subsurface flow and kinematic wave equations for overland flow, and biogeochemical models are typically given in terms of functional relations of nutrient availability, microbial activity and other factors. Therefore, the models demand for different numerical treatment. In particular, the definition of biogeochemical models has usually not continuous but discrete form in the first place, both with respect to space in terms of soil layers and with respect to time to adhere to the mostly fixed time intervals of the given input data series. Third, the models often demand for different computational resources due to their different mathematical nature. Fourth, existing simulation codes need to be reused to maintain their well-tested capabilities and due to the prohibitive effort for re-implementing them for the purpose of model coupling. Together, these four issues form a major obstacle impeding comprehensive nutrient cycle simulations. Common approaches are hence often limited in spatial or temporal extent or resolution, or model complexity.
To address this complex of problems, we recently proposed an advanced approach which uses a bidirectional coupling of the process-oriented biogeochemical model LandscapeDNDC [36, 55, 24] and the hydrology model Catchment Modelling Framework (CMF) [54]. This coupling of dedicated models addressing biogeochemistry and vegetation (LandscapeDNDC), and hydrology (CMF) has been demonstrated to allow for previously unfeasible simulations of landscape-scale scenarios addressing greenhouse gas emissions and nitrate leaching from agricultural soils [108, 64, 92]. We use the OpenPALM coupler tool and our new dynamic distributor developments described in Sec. 4.5 to realize the model coupling. This allows us to reuse the existing, well-established simulation codes of LandscapeDNDC and CMF in our multiphysics simulations. In fact, their different mathematical nature yields largely different demands for numerical treatment and computational resources. Our composed one step schemes described in Sec. 2.4.1 and their implementation using OpenPALM to couple the models allow us to use high performance computing appropriately tuned to the individual model demands.
In Sec. 5.1 and 5.2, we outline the biogeochemistry and the hydrology model, respectively. For the multiphysics simulations we use a hybrid operator splitting scheme derived from the consecutive and concurrent scheme proposed in Sec. 2.4.1. We present this hybrid scheme and its implementation using OpenPALM in Sec. 5.3. Section 5.4 is dedicated to the revision of our previously published contributions and the new advances in this work.

5.1 Biogeochemical modeling

Biogeochemical modeling is based on extensive efforts to capture the complexity of ecosystem dynamics. Stating all involved processes and their modeling would clearly exceed the scope of this work. Instead, we give the exact references demonstrating important developments of ecosystem models, and where equations and parameterizations of the basic processes related to

the terrestrial C and N cycle and trace gas emissions can be found.
A model definition for microbial respiration and denitrification in homogeneous soil is given in [60, p. 338-340]. The model includes microbial growth rates, glucose carbon change rates, CO_2 production, electron acceptor change rates of O_2, NO_3^-, NO_2^-, N_2O and N_2, nitrate assimilation, and C and N mineralization and immobilization. Model parameters are derived from experiments. In [59], an extension of that model is presented which takes anaerobiosis, soil water dynamics and transport of dissolved C and N compounds into account. The model is tailored to represent the simulation counterpart to soil aggregate experiments.
An abstraction of NO and N_2O formation from the underlying biogeochemistry is developed in [69, p. 8, 18]. The authors state that the regulation of N-trace gas production and consumption in soils is well-understood on the cellular level in terms of substrate availability. However, the authors stress that the impact of ecosystem properties and factors on the relative proportion of end products of biogeochemical processes is less well-understood. They introduce a process-oriented approach where the ecosystem regulation of nitrification and denitrification is modeled by means of process pipe abstractions with *"leaking"* N-trace gas emissions.
[17, p. 9762, 9764, 9766f.] presents the basis of the widely-adopted *"DeNitrification / DeComposition"* (DNDC) model system. It comprises sub-models for thermal-hydraulic flows, for (de-)nitrification, and for (de-)composition. Using the process-oriented approach mentioned above, the model system predicts temperature and moisture profiles from climate input data which are driving the C and N biogeochemistry in agricultural soils. An extension of the DNDC model system with a plant growth sub-model and cropping practice routines is presented in [16, p. 239f.].
A refinement of microbial growth modeling is developed in [8, p. 1744-1747, 1750], where the authors introduce the index of physiological state as a new state variable. This advanced model is able to capture microbial transition between active and dormant state, priming action on the decomposition of insoluble soil organic matter, and reduced efficiency of microbial biosynthesis under N deficit.
Further advancement and specialization of the DNDC model system is presented in [15, p. 4373, 4376, 4378] for forest ecosystems, and in [61, p. 274-276] for agricultural ecosystems. The former work describes the construction of the PnET-N-DNDC model system from the Photosynthesis-Evapotranspiration (PnET) model [49], the DNDC model system [17], and a new nitrification model. The main model innovation is the consideration of anaerobic micro-sites by means of the anaerobic volume fraction of the soil, which is termed an *"anaerobic balloon* in [15]. The second work mentioned above ([61]) presents an advanced DNDC model system for agricultural soils consisting of two main components. The first component comprises soil climate, crop growth and (de-)composition sub-models, and the second component comprises (de-)nitrification and fermentation sub-models. This model system predicts soil environmental factors such as temperature, moisture, acidity, oxidation/reduction potential and substrate concentrations. Based on these factors, biogeochemical processes related to the N and C cycles, and in particular NO, N_2O and CH_4 fluxes, are calculated.

5.1.1 LandscapeDNDC

LandscapeDNDC is a process-based ecosystem simulation framework [36]. It represents a generalization of the agricultural and the forest DNDC model system. LandscapeDNDC combines modules for plant growth, micro-meteorology, water cycling and carbon and nitrogen biogeochemistry for forest, arable, and grassland ecosystems. It was successfully used in various studies, e.g. plant growth [106], water balance [44], soil respiration and carbon exchange [70], trace gas emission [24, 53] and NO_3 leaching [83, 53]. The key model objects of LandscapeDNDC are

one-dimensional *"kernels"* representing a single soil column. Multiple kernels can be arranged in a lateral grid to extent the model application domain from site to regional scales. Vertical discretization is done by means of subsurface soil layers and above ground vegetation and canopy layers. Kernels can be configured to represent specific soil properties, vegetation structure and microbiology and the related biogeochemical processes by attaching the corresponding process modules. The water cycle module of LandscapeDNDC uses a tipping bucket approach which models the vertical downward seepage of soil water through the layers. LandscapeDNDC does not account for any lateral exchange.

Seeking to capture the high complexity of natural ecosystems, LandscapeDNDC comprises a multitude of state equations to model the various involved processes. Here we explicitly state only the most important model equations related to microbial nitrification and denitrification, and to the N_2O greenhouse gas emissions. The presentation given here follows our previous work [108]. The derivation of these equations, and of the other model components, can be taken from the references given at the beginning of the section.

To address the high sensitivity of nitrification and denitrification on the presence or absence of oxygen, LandscapeDNDC uses the anaerobic volume fraction f_{av} which is modeled as

$$f_{\text{av}} = \exp(-\sqrt{\alpha c(\text{O}_2)}) \,,$$

where $\alpha > 0$ is a scaling parameter and $c(\text{O}_2)$ is the concentration of oxygen. Generally, we denote the concentration of some chemical species as $c(\cdot)$. The main steps of nitrification are the oxidation of ammonium (NH_4^+) and ammonia (NH_3) to nitrite (NO_2^-), and the oxidation of nitrite to nitrate (NO_3^-). The nitrification rate is

$$\partial_t c(\text{NO}_2^-) = k[\text{NO}_2^-] \frac{2 f_{\text{act},\text{NO}_2^-}(T,\theta)}{\frac{1}{f_{\text{NO}_2^-}(\text{pH})} + \frac{1}{f_{\text{NO}_2^-}(c(\text{NH}_4^+))}} (1 - f_{\text{av}}) \,,$$

where $k[\text{NO}_2^-]$ is a model parameter, $f_{\text{act},\text{NO}_2^-}(T,\theta)$ is an activity factor of NO_2^--producing microbes which depends on temperature T and soil water content θ as introduced in [8], and $f_{\text{NO}_2^-}(\text{pH})$ and $f_{\text{NO}_2^-}(c(\text{NH}_4^+))$ are factors modeling the impact of acidity and ammonium concentration on the nitrification rate. The formation of N_2O as a byproduct of the nitrification pathway is modeled as

$$\partial_t c(\text{N}_2\text{O}) = k[\text{N}_2\text{O}] f_{\text{act},\text{N}_2\text{O}}(T,\theta) f_{\text{N}_2\text{O}}(pH) \partial_t c(\text{NO}_2) \,.$$

The modeling of denitrification, i.e. the reduction of NO_3^- via NO_2^-, NO and N_2O to N_2, follows [60]. To unify the notation for the denitrification pathway, we denote the set of reducible nitrogen compounds as $\mathcal{N}_{\text{red}} = \{\text{NO}_3^-, \text{NO}_2^-, \text{NO}, \text{N}_2\text{O}\}$. The denitrification rates are then modeled as

$$\partial_t c(\mathcal{N}) = -c f_{\text{act},\mathcal{N}}(T,\theta) \left(\frac{\mu(\mathcal{N})}{Y(\mathcal{N})} + m(\mathcal{N}) \frac{c(\mathcal{N})}{\bar{c}(\bar{\mathcal{N}})} \right) \frac{\bar{c}(\bar{\mathcal{N}})}{k[\mathcal{N}] + \bar{c}(\bar{\mathcal{N}})} f_{\mathcal{N},\text{C}} f_{\text{av}} f_{\mathcal{N}}(\text{pH}) + \partial_t c(\mathcal{N}^-)$$

for $\mathcal{N} \in \mathcal{N}_{\text{red}}$, where again $k[\cdot]$ are model parameters, $f_{\mathcal{N},\text{C}}$ are factors modeling the impact of carbon availability, and

$$\bar{c}(\bar{\mathcal{N}}) = \sum_{\mathcal{N} \in \mathcal{N}_{\text{red}}} c(\mathcal{N}) \,.$$

The term $\partial_t c(\mathcal{N}^-)$ denotes the turnover rate of the next higher reduced N compound, e.g. NO_2^- is the next higher reduced N compound from NO_3^-. The vertical diffusion of volatile species $\mathcal{V}$ through the soil matrix depending on water content and temperature is modeled as

$$\partial_t c(\mathcal{V}) = \partial_z \left[D_{\mathcal{V}} (1-\theta)^{\nu_1} n^{\nu_2} \left(\frac{T}{T_0} \right)^{\nu_3} \partial_z c(\mathcal{V}) \right] ,$$

where n is the soil porosity, T_0 is the reference temperature, $D_{\mathcal{V}}$ is the species' diffusion coefficient for $\theta = 0$, $n = 1$, $T = T_0$, and ν_1, ν_2, ν_3 are model parameters. The vertical partial derivatives are approximated using first order finite differences between adjacent soil layers.
This brief outline of some of the equations making up the LandscapeDNDC model system demonstrates its mathematical nature with time dependent state variables including species concentrations, microbial biomass and their active and dormant fractions, and water content. The full model definition including the components which are not presented here, e.g. the vegetation, water and micro-climate models, or the (de-)composition and other processes, can be taken from the references given above. Overall, the underlying mathematical form is a system of ordinary differential equations. The model states are discrete in space since they are associated to soil layers, and dependent on time.
As we discussed above, we are coupling LandscapeDNDC with the dedicated hydrology model Catchment Modelling Framework (CMF) to account for lateral exchange of nutrients. Therefore, we disable the water cycle module of LandscapeDNDC and leave all calculations related to the movement of water and transport of dissolved substances to the hydrology model. The shared state variables of CMF and LandscapeDNDC are the soil water content and the concentrations of solutes. We use w to denote the shared water states, and s to denote the shared solute states. Conceptually, the action of the biogeochemical model on the shared states can be written as

$$\partial_t(w, s) = f_{\text{bgc}}(w, s, t), \tag{5.1}$$

where f_{bgc} denotes the functional representation of the model action on the shared state variables. Note that, although we disable the water cycle module of LandscapeDNDC, it still acts on the water states through the calculation of ice in the micro-climate module. However, LandscapeDNDC does neither expose its shared model states (w, s) in the sense of vectors to the user, nor the functional representation f_{bgc} of its action on them. Instead, LandscapeDNDC exposes a function for propagating the model system by one time step. The time step size must be chosen as an integer fraction of 24 hours due to internal restrictions on the preprocessing of input data. The propagation is computed by means of a quasi steady state approximation (QSSA). Fast processes are assumed to attain equilibrium instantaneously, and corresponding states are thus approximated from algebraic relations. Slower processes are integrated by means of the explicit Euler method. Using the notation introduced in Sec. 2.4.1 for composed one step methods, we denote this biogeochemistry integrator by F_{bgc} and the corresponding procedure by $\Phi^{F_{\text{bgc}}}$. The computation of one time step for the biogeochemistry model then reads

$$(\tilde{w}_n, \tilde{s}_n) = \Phi^{F_{\text{bgc}}}(f_{\text{bgc}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})\,. \tag{5.2}$$

We denote the solution for the new time instance with the tilde symbol to emphasize its role as preliminary result used in a concurrent operator splitting scheme to form the global result.

5.2 Hydrological modeling

We consider porous media flow in the subsurface soil and free surface flow above the ground, including advection of dissolved substances. Our models are based on the equation of continuity for the water and for the substances, respectively:

$$\partial_t\theta + \nabla \cdot \mathbf{q} = \gamma \tag{5.3a}$$

$$\partial_t(c\theta) + \nabla \cdot (c\mathbf{q}) = \eta \tag{5.3b}$$

where c is the concentration of any dissolved substance in the water, and γ and η represent any sources or sinks of water and substances. θ and $\mathbf{q}$ have different meaning for subsurface and

overland flow, which we outline in the following paragraphs.

Overland flow

For overland flow, we label all terms with an "o" superscript. $\theta = \theta^{\mathrm{o}}$ in (5.3a) and (5.3b) means the volume of water per unit area, i.e. water height above ground, and $\mathbf{q} = \mathbf{q}^{\mathrm{o}}$ means volume of water crossing horizontal unit width in unit time. We treat overland flow as a special case of an open channel flow, which is often described with a kinematic wave approach. The mean flow velocity is given by $\mathbf{v}^{\mathrm{o}} = \mathbf{q}^{\mathrm{o}}/\theta^{\mathrm{o}}$. Equating gravitational force $\rho g S A(\theta^{\mathrm{o}}) L$ and frictional force $f \rho v^2 P L$ yields

$$v = \Big(\frac{gSA(\theta^{\mathrm{o}})}{fP}\Big)^{\frac{1}{2}} = C\sqrt{RS}$$

with cross-sectional flow area $A(\theta^{\mathrm{o}})$, channel length L, whetted perimeter P, friction coefficient f, slope of the energy line S, downstream flow speed v, and $C := \sqrt{g/f}$, $R := A(\theta^{\mathrm{o}})/P$ [62]. R is also known as hydraulic radius, and the slope of the energy line S is obtained as the derivative of the overland water hydraulic head $z + \theta^{\mathrm{o}}$ in direction of the flow, where z is the elevation of the ground with respect to a given reference. Based on the assumption that friction varies with R as $f \propto R^{-1/3}$, Manning derived the formula

$$v = \frac{1}{n_{\mathrm{M}}}\sqrt{S}R^{\frac{2}{3}} \tag{5.4}$$

where n_{M} is a coefficient describing the roughness of the flow bed, also called "Manning's n" [20]. The kinematic wave approach thus yields the flow in downstream direction as

$$q^{\mathrm{o}} = \frac{1}{n_{\mathrm{M}}}\sqrt{S}{\theta^{\mathrm{o}}}^{\frac{5}{3}}\,.$$

Subsurface flow

For the subsurface porous media flow, we label the terms with an "s" superscript. According to [12], porous media is characterized through the following properties:

- The space is occupied by a number of phases. A phase is defined as that portion of the space which is occupied by a material with uniform properties, and which is separated from other materials by a well-defined interface.
- At least one of the phases is solid.
- The phases are distributed throughout the whole space.

Clearly, the solid phase in our case is the soil. The voids or open spaces between the solid soil particles are called pores. The pore space is occupied by a wetting phase and a gas phase, in our case water and air. A common assumption in hydrology is that the gas phase pressure is uniform in space. Therefore, the two-phase model decouples, and often only the water flow is of interest [45]. An important porous media characteristic is its porosity

$$n = \lim_{\mathrm{vol}(V)\to 0} \frac{\text{pore space in } V}{\mathrm{vol}(V)}\,,$$

which is to be understood in terms of a density as a representative pore ensemble average in the soil. The volumetric water content of the soil θ^{s} is defined similarly as

$$\theta^{\mathrm{s}} = \lim_{\mathrm{vol}(V)\to 0} \frac{\text{volume of water in } V}{\mathrm{vol}(V)} .$$

Clearly, $\theta^{\mathrm{s}} \leq n$ since the maximum soil water content is achieved when the pores are entirely filled. This state is called full saturation, where $\theta^{\mathrm{s}} = n$ by definition.
With respect to the equations of continuity 5.3, we write $\theta = \theta^{\mathrm{s}}$ and $\mathbf{q} = \mathbf{q}^{\mathrm{s}}$ meaning volumetric water content of the soil and volumetric flux, i.e. volume of water crossing unit cross-sectional area in unit time, respectively. According to Darcy's law (1856) for saturated flow in porous media, the volumetric flux is related to the total soil water potential ψ_w^{s} by

$$\mathbf{q}^{\mathrm{s}} = -K\nabla\psi_w^{\mathrm{s}} ,$$

where K denotes a proportionality factor known as hydraulic conductivity [12, 97]. Buckingham postulated in 1907 that Darcy's law is also valid in the case of unsaturated soils [75], where the hydraulic conductivity becomes a function of the water content $K = K(\theta^{\mathrm{s}})$ yielding [12]

$$\mathbf{q}^{\mathrm{s}} = -K(\theta^{\mathrm{s}})\nabla\psi_w^{\mathrm{s}} .$$

Using this flux law, Richards derived in 1931 from (5.3a) the equation

$$\partial_t\theta^{\mathrm{s}} - \nabla\cdot\Big[K(\theta^{\mathrm{s}})(\nabla\psi_m + \rho g\mathbf{e}_z)\Big] = \gamma , \tag{5.5}$$

which is known as the Richards equation [90]. ψ_m denotes the matrix potential of the soil, and the total soil water potential is $\psi_w^{\mathrm{s}} = \psi_m + \rho g z$. We use the van-Genuchten-Mualem model [73, 104] to approximate the soil water retention curve $\theta^{\mathrm{s}}(\psi_m)$ and the hydraulic conductivity $K(\theta^{\mathrm{s}})$.

Interface condition between overland and subsurface flow

At the interface Γ_{S} between subsurface and overland flow, water may infiltrate or escape the soil. Since the free surface overland water flow is explicitly included in our hydrology model, we can treat infiltration by means of Richards approach, thus avoiding limited flux boundaries and switching conditions. The infiltration/exfiltration flow depends on the difference of the hydraulic head across the soil surface Γ_{S}. The overland water hydraulic head is $h^{\mathrm{o}} = \theta^{\mathrm{o}} + z$. According to Richards' approach we model the flow q_{S} through the soil surface in upward vertical direction as

$$q_{\mathrm{S}} = -K(\theta^{\mathrm{s}}|_{\Gamma_{\mathrm{S}}})(h^{\mathrm{o}} - h^{\mathrm{s}}|_{\Gamma_{\mathrm{S}}}) = -K(\theta^{\mathrm{s}}|_{\Gamma_{\mathrm{S}}})(\theta^{\mathrm{o}} - \psi_m|_{\Gamma_{\mathrm{S}}}) , \tag{5.6}$$

where $\theta^{\mathrm{s}}|_{\Gamma_{\mathrm{S}}}$, $h^{\mathrm{s}}|_{\Gamma_{\mathrm{S}}}$ and $\psi_m|_{\Gamma_{\mathrm{S}}}$ are the traces of the soil water content, of the soil water hydraulic head, and of the matrix potential on the soil surface Γ_{S}, respectively. The soil surface Γ_{S} is on the one hand the overland flow domain, and on the other hand the upward boundary of the subsurface flow domain. Therefore, the infiltration/exfiltration flow q_{S} through Γ_{S} represents source or sink with respect to the overland flow, and a Neumann boundary condition with respect to the subsurface flow.
The advection of solutes must take the direction of the flow into account. For infiltration, the solute flow is determined by the concentration in the overland water, while for exfiltration it is

determined by the concentration in the soil water. We denote by $c(q_S)$ the concentration at the origin of the flow as

$$c(q_S) = \begin{cases} c^s & \text{if } q_S > 0 \,, \\ c^o & \text{if } q_S < 0 \,, \\ 0 & \text{else} \,. \end{cases}$$

Therefore, the solute flow through the soil surface Γ_S in upward vertical direction is $c(q_S)q_S$.

Boundary conditions

In the neighborhood Γ_{out} of the lowest point on the boundary, we impose boundary conditions which ensure that water may flow out of the area of study, but not into it. For the overland flow, we prescribe a slope S_{out} pointing downwards from $\Gamma^o_{out} := \Gamma_{out} \cap \Gamma_S$. According to Manning's law (5.4), this causes an overland outflow $\mathbf{q}^o_{out}$ whenever $\theta^o > 0$ holds at Γ^o_{out}, or no flow otherwise, i.e. $\mathbf{q}^o_{out} = 0$. For the subsurface flow, we prescribe a Dirichlet condition on the water potential ψ^s_{out}. On all other parts of the boundary, denoted Γ_0 and $\Gamma^o_0 := \Gamma_0 \cap \Gamma_S$, we prescribe the no-flow homogeneous Neumann conditions $\mathbf{q}^{o/s} = 0$. No boundary conditions for the concentrations $c^{o/s}$ need to be prescribed on Γ_{out} and on Γ_0, since no inflow can happen through these boundaries.

Meteorological and environmental factors

We model the precipitation as a source term γ_{prec} for the overland flow. In contrast, evaporation acts as a sink of water. Depending on the state of the system, it may happen either with ponding water on the soil surface, or in near-surface soil layers without ponding water. We denote the corresponding sink terms as $\gamma^o_{evap}(\theta^o, \theta^s)$ and $\gamma^s_{evap}(\theta^o, \theta^s)$, respectively. Finally, the source/sink terms $\gamma_{bgc}(\theta^s)$ and $\eta_{bgc}(\theta^s, c^s)$ for the subsurface system represent the net result of the biogeochemical processes in the soil with respect to water and solutes, respectively.

5.2.1 Hydrology problem formulation

Summing up the above considerations, we state the hydrology problem as the following system of partial differential equations:

$$\partial_t \theta^{\mathrm{o}} + \nabla \cdot \mathbf{q}^{\mathrm{o}} = q_{\mathrm{S}} \mathbf{e}_z + \gamma_{\mathrm{prec}} + \gamma^{\mathrm{o}}_{\mathrm{evap}}(\theta^{\mathrm{o}}, \theta^{\mathrm{s}}) \qquad \text{on } \Gamma_{\mathrm{S}} \times (0,T) \tag{5.7a}$$

$$\partial_t (c^{\mathrm{o}} \theta^{\mathrm{o}}) + \nabla \cdot (c^{\mathrm{o}} \mathbf{q}^{\mathrm{o}}) = c(q_{\mathrm{S}}) q_{\mathrm{S}} \mathbf{e}_z + c_{\mathrm{prec}} \gamma_{\mathrm{prec}} \qquad \text{on } \Gamma_{\mathrm{S}} \times (0,T) \tag{5.7b}$$

$$\partial_t \theta^{\mathrm{s}} + \nabla \cdot \mathbf{q}^{\mathrm{s}} = \gamma^{\mathrm{s}}_{\mathrm{evap}}(\theta^{\mathrm{o}}, \theta^{\mathrm{s}}) + \gamma_{\mathrm{bgc}}(\theta^{\mathrm{s}}) \qquad \text{in } \Omega \times (0,T) \tag{5.7c}$$

$$\partial_t (c^{\mathrm{s}} \theta^{\mathrm{s}}) + \nabla \cdot (c^{\mathrm{s}} \mathbf{q}^{\mathrm{s}}) = \eta_{\mathrm{bgc}}(\theta^{\mathrm{s}}, c^{\mathrm{s}}) \qquad \text{in } \Omega \times (0,T) \tag{5.7d}$$

$$\mathbf{q}^{\mathrm{s}} = q_{\mathrm{S}} \mathbf{e}_z \qquad \text{on } \Gamma_{\mathrm{S}} \times (0,T) \tag{5.7e}$$

$$c^{\mathrm{s}} \mathbf{q}^{\mathrm{s}} = c(q_{\mathrm{S}}) q_{\mathrm{S}} \mathbf{e}_z \qquad \text{on } \Gamma_{\mathrm{S}} \times (0,T) \tag{5.7f}$$

$$\mathbf{q}^{\mathrm{o}} = \mathbf{q}^{\mathrm{o}}_{\mathrm{out}} \qquad \text{on } \Gamma^{\mathrm{o}}_{\mathrm{out}} \times (0,T) \tag{5.7g}$$

$$\psi^{\mathrm{s}}_w = \psi^{\mathrm{s}}_{\mathrm{out}} \qquad \text{on } \Gamma_{\mathrm{out}} \times (0,T) \tag{5.7h}$$

$$\mathbf{q}^{\mathrm{o}} = 0 \qquad \text{on } \Gamma^{\mathrm{o}}_0 \times (0,T) \tag{5.7i}$$

$$\mathbf{q}^{\mathrm{s}} = 0 \qquad \text{on } \Gamma_0 \times (0,T) \tag{5.7j}$$

$$\theta^{\mathrm{o}}(0) = 0 \qquad \text{on } \Gamma_{\mathrm{S}} \tag{5.7k}$$

$$c^{\mathrm{o}}(0) = 0 \qquad \text{on } \Gamma_{\mathrm{S}} \tag{5.7l}$$

$$\theta^{\mathrm{s}}(0) = \theta^{\mathrm{s}}_0 \qquad \text{in } \Omega \tag{5.7m}$$

$$c^{\mathrm{s}}(0) = 0 \qquad \text{in } \Omega \tag{5.7n}$$

5.2.2 Discretization

We use a cell-centered finite volume method for the spatial discretization of (5.7). It is based on a grid $\Omega_h = \bigcup_{i=1}^{N^{\mathrm{s}}} C_i$ of N^{s} non-overlapping polyhedral cells C_i $(i = 1, ..., N^{\mathrm{s}})$ covering the domain Ω. Let N^{o} be the number of cells in the top layer which have a face F_i $(i = 1, ..., N^{\mathrm{o}})$ at the soil surface. The vertical projection of F_i onto the plane $z = 0$ is denoted A^{o}_i. The volume of overland water on face F_i and the volume of soil water in cell C_i are

$$w^{\mathrm{o}}_i = \int_{A^{\mathrm{o}}_i} \theta^{\mathrm{o}} \, d\sigma \quad (i = 1, ..., N^{\mathrm{o}}) \text{ and}$$

$$w^{\mathrm{s}}_i = \int_{C_i} \theta^{\mathrm{s}} \, dx \quad (i = 1, ..., N^{\mathrm{s}}) \text{, respectively.}$$

Analogously, the amount of solute on face F_i and the amount of solute in cell C_i are

$$s^{\mathrm{o}}_i = \int_{A^{\mathrm{o}}_i} c^{\mathrm{o}} \theta^{\mathrm{o}} \, d\sigma \quad (i = 1, ..., N^{\mathrm{o}}) \text{ and}$$

$$s^{\mathrm{s}}_i = \int_{C_i} c^{\mathrm{s}} \theta^{\mathrm{s}} \, dx \quad (i = 1, ..., N^{\mathrm{s}}) \text{, respectively.}$$

The average concentration of the solute in the overland water on face F_i and in the soil water in cell C_i is

$$c^{\mathrm{o}}_i = \left\{ \begin{array}{c} \frac{s^{\mathrm{o}}_i}{w^{\mathrm{o}}_i} \text{ , if } w^{\mathrm{o}}_i > 0 \\ 0 \text{ , else} \end{array} \right\} \text{ and } c^{\mathrm{s}}_i = \left\{ \begin{array}{c} \frac{s^{\mathrm{s}}_i}{w^{\mathrm{s}}_i} \text{ , if } w^{\mathrm{s}}_i > 0 \\ 0 \text{ , else} \end{array} \right\} \text{, respectively.}$$

Integrating Eq. (5.7a) over a face F_i and applying the divergence theorem [41] yields

$$\partial_t w_i^{\mathrm{o}} + \int_{\partial F_i} \mathbf{n} \cdot \mathbf{q}^{\mathrm{o}} \, d\varsigma = \int_{F_i} \Big[\mathbf{n} \cdot \mathbf{q}_{\mathrm{S}} + \gamma_{\mathrm{prec}} + \gamma_{\mathrm{evap}}^{\mathrm{o}}(\theta^{\mathrm{o}}, \theta^{\mathrm{s}}) \Big] \, d\sigma \ .$$

Let I_i^{o} denote the index set of the adjacent faces to F_i. The boundary integral above turns into a sum of fluxes to the adjacent faces

$$\int_{\partial F_i} \mathbf{n} \cdot \mathbf{q}^{\mathrm{o}} \, d\varsigma = \sum_{j \in I_i^{\mathrm{o}}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} \ ,$$

where q_{ij}^{o} denotes the volumetric flux from F_i to F_j over the edge $\partial F_i \cap \partial F_j$ with cross-sectional width A_{ij}^{o}. According to Manning's formula (5.4), the flow q_{ij}^{o} is taken as

$$q_{ij}^{\mathrm{o}} = \frac{\mathrm{sign}(z_i - z_j)}{n_{\mathrm{M}}} \sqrt{S_{ij}} \Big(h_{ij}^{\mathrm{o}} \Big)^{\frac{5}{3}} \tag{5.8}$$

where the $S_{ij} = \frac{|h_i^{\mathrm{o}} - h_j^{\mathrm{o}}|}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}$ is the slope of the energy line, i.e. the overland water potential difference, between the faces F_i and F_j, and

$$h_i^{\mathrm{o}} = w_i^{\mathrm{o}} / \mathrm{vol}(A_i^{\mathrm{o}}) \quad , \quad h_{ij}^{\mathrm{o}} = \begin{cases} w_i^{\mathrm{o}} / \mathrm{vol}(A_i^{\mathrm{o}}) & \text{if } z_i \geq z_j \ , \\ w_j^{\mathrm{o}} / \mathrm{vol}(A_j^{\mathrm{o}}) & \text{if } z_i < z_j \end{cases}$$

is the average water height on the face where the flux originates from. For the advection of solutes, a similar treatment of Eq. (5.7b) yields

$$\int_{\partial F_i} \mathbf{n} \cdot (c^{\mathrm{o}} \mathbf{q}^{\mathrm{o}}) \, d\varsigma = \sum_{j \in I_i^{\mathrm{o}}} c_{ij}^{\mathrm{o}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} \ ,$$

where c_{ij}^{o} is defined as

$$c_{ij}^{\mathrm{o}} = \begin{cases} c_i^{\mathrm{o}} & \text{if } q_{ij}^{\mathrm{o}} > 0 \ , \\ c_j^{\mathrm{o}} & \text{if } q_{ij}^{\mathrm{o}} < 0 \ , \\ 0 & \text{else} \ , \end{cases}$$

which is the average concentration of the solute on the face where the flux originates from.
We skip the details of the finite volume discretization for the subsurface flow since it uses the same techniques. We only give the approximation of the flow according to Eq. (5.5)

$$q_{ij}^{\mathrm{s}} = - \frac{2}{1/K(W_i^{\mathrm{s}}) + 1/K(W_j^{\mathrm{s}})} \frac{h_i^{\mathrm{s}} - h_j^{\mathrm{s}}}{\|\mathbf{x}_i - \mathbf{x}_j\|_2} \ , \tag{5.9}$$

where

$$W_i^{\mathrm{s}} = \frac{w_i^{\mathrm{s}}}{\text{volume of the pore space in } C_i}$$

is the wetness and $h_i^{\mathrm{s}} = \psi_m(W_i^{\mathrm{s}}) + \rho g z_i$ is the average water potential in cell C_i.
At the interface Γ_{S}, let $C_{k(i)}$ be that cell of Ω_h which has $F_i \subset \Gamma_{\mathrm{S}}$ as a face. The flux through F_i is approximated as

$$q_{\mathrm{S},i} = - \frac{2}{1/K_{\mathrm{sat}} + 1/K(W_{k(i)}^{\mathrm{s}})} \frac{h_i^{\mathrm{o}} - \psi_m(W_{k(i)}^{\mathrm{s}})}{0.5 \times \mathrm{height}(C_{k(i)})} \ , \tag{5.10}$$

where K_{sat} denotes the saturated conductivity of cell $C_{k(i)}$. The solute concentration is denoted

$$c_{\mathrm{S},i} = \begin{cases} c_i^{\mathrm{o}} & \text{if } q_{\mathrm{S},i} < 0 \ , \\ c_{k(i)}^{\mathrm{s}} & \text{if } q_{\mathrm{S},i} > 0 \ , \\ 0 & \text{else} \ . \end{cases}$$

The spatially discretized system reads

$$\dot{w}_i^{\mathrm{o}} + \sum_{j \in I_i^{\mathrm{o}}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} = \bar{\gamma}_i^{\mathrm{o}} \qquad \text{in } (0,T) \qquad (i = 1, ..., N^{\mathrm{o}}) \ , \tag{5.11}$$

$$\dot{s}_i^{\mathrm{o}} + \sum_{j \in I_i^{\mathrm{o}}} c_{ij}^{\mathrm{o}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} = \bar{\eta}_i^{\mathrm{o}} \qquad \text{in } (0,T) \qquad (i = 1, ..., N^{\mathrm{o}}) \ , \tag{5.12}$$

$$\dot{w}_i^{\mathrm{s}} + \sum_{j \in I_i^{\mathrm{s}}} q_{ij}^{\mathrm{s}} A_{ij}^{\mathrm{s}} = \bar{\gamma}_i^{\mathrm{s}} \qquad \text{in } (0,T) \qquad (i = 1, ..., N^{\mathrm{s}}) \ , \tag{5.13}$$

$$\dot{s}_i^{\mathrm{s}} + \sum_{j \in I_i^{\mathrm{s}}} c_{ij}^{\mathrm{s}} q_{ij}^{\mathrm{s}} A_{ij}^{\mathrm{s}} = \bar{\eta}_i^{\mathrm{s}} \qquad \text{in } (0,T) \qquad (i = 1, ..., N^{\mathrm{s}}) \ , \tag{5.14}$$

$$w_i^{\mathrm{o}}(0) = w_{i,0}^{\mathrm{o}} \ , \quad s_i^{\mathrm{o}}(0) = s_{i,0}^{\mathrm{o}} \qquad (i = 1, ..., N^{\mathrm{o}}) \ , \tag{5.15}$$

$$w_i^{\mathrm{s}}(0) = w_{i,0}^{\mathrm{s}} \ , \quad s_i^{\mathrm{s}}(0) = s_{i,0}^{\mathrm{s}} \qquad (i = 1, ..., N^{\mathrm{s}}) \ , \tag{5.16}$$

where the terms on the right hand sides of Eqs. (5.11) - (5.14) are

$$\begin{aligned} \bar{\gamma}_i^{\mathrm{o}} &= q_{\mathrm{S},i} A_i^{\mathrm{o}} + \int_{F_i} \gamma_{\text{prec}} + \gamma_{\text{evap}}^{\mathrm{o}} \, d\sigma \ , \\ \bar{\eta}_i^{\mathrm{o}} &= c_{\mathrm{S},i} q_{\mathrm{S},i} A_i^{\mathrm{o}} + \int_{F_i} c_{\text{prec}} \gamma_{\text{prec}} \, d\sigma \ , \\ \bar{\gamma}_i^{\mathrm{s}} &= \int_{F_i} \gamma_{\text{evap}}^{\mathrm{s}} + \gamma_{\text{bgc}} \, d\sigma \ , \\ \bar{\eta}_i^{\mathrm{s}} &= \int_{F_i} \eta_{\text{bgc}} \, d\sigma \ . \end{aligned}$$

For simplicity, we introduce the following short-cut notation for the spatially discretized system (5.11) - (5.14):

$$\partial_t (w, s) = f_{\text{hyd}}(w, s, t) \quad \text{in } (0,T) \ , \tag{5.17}$$

where $(w, s) = (w^{\mathrm{o}}, w^{\mathrm{s}}, s^{\mathrm{o}}, s^{\mathrm{s}}) \in \mathbb{R}^{2N^{\mathrm{o}}+2N^{\mathrm{s}}}$ represents the vector of water and solute states, and

$$f_{\text{hyd}}(w, s, t) = \begin{bmatrix} \bar{\gamma}_i^{\mathrm{o}} - \sum_{j \in I_i^{\mathrm{o}}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} & (i = 1, ..., N^{\mathrm{o}}) \\ \bar{\gamma}_i^{\mathrm{s}} - \sum_{j \in I_i^{\mathrm{s}}} q_{ij}^{\mathrm{s}} A_{ij}^{\mathrm{s}} & (i = 1, ..., N^{\mathrm{s}}) \\ \bar{\eta}_i^{\mathrm{o}} - \sum_{j \in I_i^{\mathrm{o}}} c_{ij}^{\mathrm{o}} q_{ij}^{\mathrm{o}} A_{ij}^{\mathrm{o}} & (i = 1, ..., N^{\mathrm{o}}) \\ \bar{\eta}_i^{\mathrm{s}} - \sum_{j \in I_i^{\mathrm{s}}} c_{ij}^{\mathrm{s}} q_{ij}^{\mathrm{s}} A_{ij}^{\mathrm{s}} & (i = 1, ..., N^{\mathrm{s}}) \end{bmatrix}$$

denotes the complete right-hand-side function of the hydrology problem.

Problem (5.17) is highly nonlinear due to the nonlinear dependency of the conductivity on the soil water content [11, 73, 104]. Additionally, it shows multi-scale nature in terms of the temporal scale of the various hydrological processes including subsurface flow under arid conditions, steep infiltration fronts, and surface runoff upon intense rainfall. Therefore, powerful numerical integration schemes are needed to accurately compute the temporal evolution of the hydrological model. We use an integration scheme with a time step size control mechanism based on the CVODE solver package for ordinary differential equation initial value problems [22, 42]. CVODE is used in its stiff integration mode. In this mode, CVODE represents an implicit backward differentiation formula (BDF) of variable order q between 1 and 5 with variable time step size. It uses Newton's method to solve the nonlinear system for each time step, employing preconditioned Krylov subspace methods [93] for the solution of the linear system in each Newton iteration. Inside the linear solver, the matrix vector product of the Jacobian matrix $J = \nabla f_{\text{hyd}}$ and a vector v, i.e. the directional derivative, is approximated by a finite difference as explained above. We employ CVODE's parallel vector data structure for evaluating the right-hand-side function f_{hyd} and all internal computations of the CVODE routines. We denote the hydrology integrator by F_{hyd} and the corresponding procedure by $\Phi^{F_{\text{hyd}}}$, so that the computation of one time step for the hydrology model reads

$$(\hat{w}_n, \hat{s}_n) = \Phi^{F_{\text{hyd}}}(f_{\text{hyd}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})\,. \tag{5.18}$$

Here we denote the solution for the new time instance with the hat symbol to emphasize its role as preliminary result used in a concurrent operator splitting scheme to form the global result.

5.2.3 Implementation and parallelization of the hydrology model

We use Catchment Modelling Framework (CMF) [54] to realize the spatial finite volume discretization presented above. It is a C++ library for creating hydrological simulation models. It offers a variety of classes and functions which represent the ingredients of a finite volume discretization as proposed by Qu and Duffy [81]. Hydrological models are set up as a network using node and connection objects within the soil. The nodes of the network represent the cells of the spatial discretization in Ω_h. They have the hydrological meaning of a water storage keeping track of the water content. They are equipped with material specific constants, boundary conditions and a position in the geometry. The connection objects represent the edges of the network between adjacent cells. They contain the definition of the flux approximation, like the Richards approximation for subsurface flow between adjacent cells, kinematic wave approximation for overland flow, and evaporation and rainfall on the soil surface.
For modeling the transport of dissolved substances like nitrate (NO_3^-) or dissolved carbon (DOC), CMF attaches solute storage objects to each water storage. The solute storages keep track of the solute content in the cells. Both storage types provide methods to calculate the time derivative of their state. For each cell $C_i \in \Omega_h$, the corresponding water storage computes $\dot{w}_i$, and the attached solute storage computes $\dot{s}_i$ by iterating over the flux connections and evaluating the contributions using the flow approximations (5.8), (5.9) and (5.10). Thus, the CMF model calculates the right hand side function f_{hyd} of Equation (5.17). Furthermore, it offers routines to transfer the values of the hydrological model states from the CMF model core into vector form and vice versa.

For simulations with CMF, one usually employs a numerical integration scheme to compute the temporal evolution of (5.17). The use of explicit schemes is straight forward since they rely on evaluations of f_{hyd} with known arguments from already computed past time steps. However, explicit schemes may be restricted to small time step sizes and therefore possibly require a large number of time steps resulting in a large number of function evaluations. Implicit schemes often

allow for larger time step sizes, but in general they require the solution of a nonlinear system of equations to compute the new time step. The nonlinear system is often solved by means of a Newton iteration, which approximates the solution in a sequence of linear problems involving the Jacobian matrix $J(w,s,t) = \nabla f_{\mathrm{hyd}}(w,s,t)$. Since the Jacobian matrix is not available in CMF, implicit schemes need to rely on a finite difference approximation of the directional derivative $J(w,s,t)[\hat{w},\hat{s}] \approx \frac{1}{h}[f_{\mathrm{hyd}}(w+h\hat{w}, s+h\hat{s}, t) - f_{\mathrm{hyd}}(w,s,t)]$, again resulting in a large number of evaluations of f_{hyd}. Either way, a great portion of the overall computational effort for CMF simulations is spent on function evaluations. Therefore, the target of our parallelization approach for CMF is the parallelization of the function evaluation of f_{hyd} and a corresponding parallelization of the numerical integrator and the vector data structures.
Our concept to enable parallel computations on distributed memory machines using MPI [5] is based on a horizontal domain decomposition of the computational domain Ω_h. Since the hydrological catchment which we consider as our computational domain has by far greater extent in the horizontal than in vertical direction, we use a two-dimensional horizontal domain decomposition. We assume that all grid cells are aligned in vertical direction, such that those cells which lie upon another can be viewed as a vertical soil column. This conceptual lateral grid of vertical soil columns is split into non-overlapping sub-domains $\Omega_r = \bigcup_{i \in I_r} C_i$ $(r = 1, ..., p)$ according to the number p of MPI processes, so that each MPI process is assigned to one sub domain. I_r denotes the index set of all cells C_i belonging to sub domain Ω_r. The partitioning is computed with the help of the graph partitioner tool METIS [51] to obtain a balanced decomposition $\Omega = \bigcup_{r=1}^{p} \Omega_r$ with a similar number of cells in each sub domain Ω_r. The MPI processes have access only to the cells of their own sub domain, and each MPI process is responsible for evaluating those component functions of f_{hyd} which correspond to any of its water or solute states.
Looking at the structure of the component functions which describe the temporal variation of the water and solute state in a cell $C \subset \Omega_r$ belonging to process r, it is necessary to compute the water and solute fluxes between the cell C and all adjacent cells. This is a straight forward calculation if all relevant cells are accessible by the MPI process r, i.e. if all relevant cells belong to the same sub domain Ω_r. However, if C lies at the sub domain boundary of Ω_r and an adjacent cell belongs to the sub domain Ω_s of another MPI process $s \neq r$, fluxes across sub domain boundaries have to be calculated. To account for fluxes between different sub-domains, each sub domain is extended by ghost cells which mirror the adjacent cells on neighboring sub-domains. MPI processes use the ghost cells in read-only mode to calculate the fluxes across sub domain boundaries which contribute to the temporal variation of the water and solute states in the cells of their own sub domain. Before the calculation, the MPI processes need to communicate the values of water and solute states at sub domain boundaries to keep the ghost cells up to date. Such ghost update is necessary before any function evaluation performed by the integrator. Therefore, the integrator is equipped with an appropriate communication functionality. It derives the communication pattern based on the adjacency of the sub-domains during initialization. The actual transfer of ghost update values is implemented with the non-blocking MPI send and receive routines `MPI_Isend` and `MPI_Irecv`, respectively. This allows to overlap communication and computation in the integrator.

5.3 Simulation with OpenPALM using a hybrid operator splitting method

Our goal is to employ the concurrent operator splitting scheme in order to benefit from its inherent parallelism on the level of the coupling scheme by running both models concurrently

on separate sets of processors. However, care must be taken with respect to the feasibility of the results in terms of their physical meaning. Even if both models safely respect the physics individually, the concurrent operator splitting scheme may yield unphysical results. This is due to the concurrent scheme's nature of propagating both models independently and afterwards combining their individual results as

$$(w_n, s_n) = \Phi^{F_{\text{hyd}}}(f_{\text{hyd}}, t_n, t_{n-1}, w_{n-1}, s_{n-1}) + \Phi^{F_{\text{bgc}}}(f_{\text{bgc}}, t_n, t_{n-1}, w_{n-1}, s_{n-1}) - (w_{n-1}, s_{n-1})\,.$$

The separation of the model propagation from the subsequent formation of the global result implies the possibility of yielding results which are mathematically correct, but unphysical. In our case, negative solute concentrations may result from the concurrent scheme if the sum of transported and consumed solute from the hydrology and the biogeochemistry model during one time step exceed the available amount before that time step. Provided that the continuous multiphysics system and the individual model integrators are physically sound, the issue of possible unphysical results of the concurrent scheme is clearly only a matter of the time step size since it is first order consistent and convergent as we proved in Sec. 2.4.1. Therefore, one possibility to prevent from unphysical results is to reduce the global time step size. However, we refrain from varying the global time step size in our application for practical reasons. Instead, we employ a hybrid form of the concurrent and consecutive operator splitting scheme which also prevents from unphysical results. Provided that the continuous multiphysics system and the individual model integrators are physically sound, the consecutive scheme maintains the physical feasibility since it does not separate the model propagation from the formation of the global result. Therefore, we use the consecutive scheme as fallback position to recompute a time step whenever we encounter unphysical results from the concurrent scheme. This hybrid operator splitting scheme is stated in Alg. 22.

Algorithm 22 Hybrid operator splitting scheme for the hydrology biogeochemistry coupling.

1: Given initial states w_0, s_0.
2: **for** $n = 1, 2, \ldots$ **do**
3: Compute one time step for the coupled models using the concurrent operator splitting scheme $(w_n, s_n) = \Phi^{F_{\text{hyd}}+F_{\text{bgc}}}(f_{\text{hyd}}, f_{\text{bgc}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})$.
4: **if** $s_n < 0$ in some component **then**
5: Recompute the time step for the coupled models using the consecutive operator splitting scheme $(w_n, s_n) = \Phi^{F_{\text{hyd}} \circ F_{\text{bgc}}}(f_{\text{hyd}}, f_{\text{bgc}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})$.
6: **end if**
7: **end for**

We designed two OpenPALM units, a CMF unit and a LandscapeDNDC unit, which implement the hybrid operator splitting scheme of Alg. 22 from the hydrology model side and from the biogeochemistry model side. The CMF unit is stated in Alg. 24, and the LandscapeDNDC unit is stated in Alg. 23. Figure 5.1 shows the PrePALM graphical user interface with the coupling scheme we used for our simulations. The blue and green vertical bars depict the two independent execution branches of the application. The boxes sitting on the branches represent the units. Each unit has a variety of plugs which are connected to each other. Plugs on the top side of a unit represent input data, and plugs on the bottom side represent output data. By connecting the plugs, units can communicate with each other. OpenPALM's communication routines are used in the units' source code to perform the actual exchange of the data. As we run the hydrology model with internal parallelization, the data to be exchanged is distributed among the corresponding MPI processes on the hydrology side. OpenPALM derives the communication paths between the

Algorithm 23 LandscapeDNDC unit implementing the hybrid operator splitting scheme of Alg. 22 from the biogeochemistry model side. The LandscapeDNDC unit interacts with the CMF unit in Alg. 24 by means of the OpenPALM communication routines.

1: Read input data and build the LandscapeDNDC project.
2: `PALM_Put` cells and topology information.
3: `PALM_Put` initial values w_0, s_0, vegetation parameters and climate data.
4: **for** $n = 1, 2, ...$ **do**
5: Compute one time step for the biogeochemical model to obtain $(\tilde{w}_n, \tilde{s}_n) = \Phi^{F_{\text{bgc}}}(f_{\text{bgc}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})$, where $\tilde{w}_n = w_{n-1}$ according to (5.2).
6: `PALM_Put` the solute states $\tilde{s}_n$ of the biogeochemical model, vegetation parameters and climate data.
7: `PALM_Get` water and solute states w_n, s_n.
8: **end for**

individual processes of the units. This avoids to collect the distributed data on one process before the communication, and to broadcast it afterwards. Instead, the data is transferred in portions according the intersection of the individual data distributions in the units. The OpenPALM driver and the two units are each compiled into their own executable. The simulations are then carried out as a multiple program multiple data (MPMD) application. All input data necessary for initializing and driving the simulations are read by the LandscapeDNDC unit from files. This includes the geometric and topological information of the cells, the soil parameters, the initial states of the models, the time series of climate and vegetation parameters, and the time series of land use and land management actions. The LandscapeDNDC unit uses these data to initialize and drive its biogeochemistry model computations. Furthermore, it sends the initialization and driving data which are required by the hydrology model to the CMF unit. Table 5.1 lists the initialization data transferred from the LandscapeDNDC unit to the CMF unit in the setup

Algorithm 24 CMF unit implementing the hybrid operator splitting scheme of Alg. 22 from the hydrology model side. The CMF unit interacts with the LandscapeDNDC unit in Alg. 23 by means of the OpenPALM communication routines.

1: `PALM_Get` cells and topology information.
2: Build CMF project and determine domain decomposition.
3: `PALM_Distributor_set` the data distribution in CMF.
4: `PALM_Get` initial values w_0, s_0, vegetation parameters and climate data.
5: **for** $n = 1, 2, ...$ **do**
6: Compute one time step for the hydrology model to obtain $(\hat{w}_n, \hat{s}_n) = \Phi^{F_{\text{hyd}}}(f_{\text{hyd}}, t_n, t_{n-1}, w_{n-1}, s_{n-1})$.
7: `PALM_Get` the solute states $\tilde{s}_n$ of the biogeochemical model, vegetation parameters and climate data.
8: Combine the states of both models as $(w_n, s_n) = (\hat{w}_n, \hat{s}_n) + (\tilde{w}_n, \tilde{s}_n) - (w_{n-1}, s_{n-1})$, where $\tilde{w}_n = w_{n-1}$ according to (5.2).
9: **if** $s_n < 0$ in some component **then**
10: Recompute the time step for the hydrology model to obtain $(w_n, s_n) = \Phi^{F_{\text{hyd}}}(f_{\text{hyd}}, t_n, t_{n-1}, \tilde{w}_n, \tilde{s}_n)$.
11: **end if**
12: `PALM_Put` water and solute states w_n, s_n.
13: **end for**

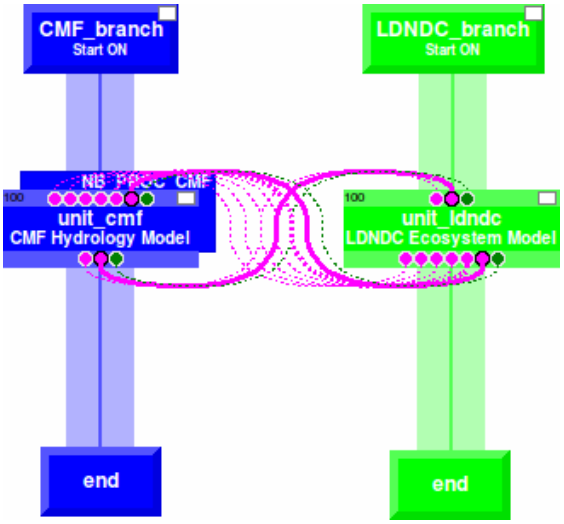


Figure 5.1: Graphical representation of the coupling algorithm in PrePALM.

phase of the simulation. From the LandscapeDNDC unit point of view, this is done by calling the `PALM_Put` routine in steps 2 and 3 of Alg. 23. From the CMF unit point of view, the corresponding calls to `PALM_Get` are in steps 1 and 4 of Alg. 24. During the time stepping loop, the LandscapeDNDC unit computes one time step for the biogeochemical model in step 5 of Alg. 23, and then sends its new solute states and the climate and vegetation parameters for the next time step to the CMF unit by calling `PALM_Put` in step 6. The data transferred from the biogeochemistry model to the hydrology model is detailed in Tab. 5.2. The CMF unit concurrently computes one time step for the hydrology model in step 6 of Alg. 24, and then receives the solute states and the climate and vegetation parameters from the LandscapeDNDC unit in step 7. Furthermore, the CMF unit combines the states of both models in step 8, checks for possible negative concentrations and recomputes the time step if necessary in step 10. Finally, it sends the new global water and solute states by calling `PALM_Put` in step 12 of Alg. 24, which is complemented by the LandscapeDNDC unit's call to `PALM_Get` in step 7 in Alg. 23. The data transferred from the hydrology model to the biogeochemistry model is detailed in Tab. 5.3.

quantity transferred CMF ⟵ LandscapeDNDC	units [·] and conversion factor α [CMF] ⟵ [LandscapeDNDC] × α				
soil layer depth	m	⟵	m		
soil saturated conduct.	$\frac{\mathrm{m}}{\mathrm{day}}$	⟵	$\frac{\mathrm{cm}}{\mathrm{min}}$	×	14.4
soil porosity	$\frac{\mathrm{m}^3}{\mathrm{m}^3}$	⟵	$\frac{\mathrm{m}^3}{\mathrm{m}^3}$		
water on surface	m^3	⟵	$\frac{\mathrm{m}^3}{\mathrm{m}^2}$	×	cell surface area
water in layers	$\frac{\mathrm{m}^3}{\mathrm{m}^3}$	⟵	$\frac{\mathrm{m}^3}{\mathrm{m}^3}$		
solutes on surface	kg	⟵	$\frac{\mathrm{kg}}{\mathrm{m}^2}$	×	cell surface area
solutes in layers	kg	⟵	$\frac{\mathrm{kg}}{\mathrm{m}^2}$	×	cell surface area
solutes in precipitation	$\frac{\mathrm{kg}}{\mathrm{m}^3}$	⟵	$\frac{\mathrm{g}}{\mathrm{m}^3}$	×	10^{-3}
solute retention factor	%	⟵	%		
harvest event	$\{0,1\}$	⟵	$\{0,1\}$		
leaf area index (LAI)	$\frac{\mathrm{m}^2}{\mathrm{m}^2}$	⟵	$\frac{\mathrm{m}^2}{\mathrm{m}^2}$		
vegetation height	m	⟵	m		
canopy closure	%	⟵	%		
root fraction	%	⟵	%		
ice fraction	%	⟵	%		
current air temp.	°C	⟵	°C		
max. air temp.	°C	⟵	°C		
min. air temp.	°C	⟵	°C		
ground temp.	°C	⟵	°C		
relative vapor pressure	Pa	⟵	% rel. humidity	×	10^{-2} abs. vapor pressure
wind speed	$\frac{\mathrm{m}}{\mathrm{s}}$	⟵	$\frac{\mathrm{m}}{\mathrm{s}}$		
radiation	$\frac{10^6\mathrm{J}}{\mathrm{day}\ \mathrm{m}^2}$	⟵	$\frac{\mathrm{W}}{\mathrm{m}^2}$	×	0.0864
precipitation	$\frac{10^{-3}\mathrm{m}}{\mathrm{day}}$	⟵	$\frac{\mathrm{m}}{\Delta t}$	×	1000 $\frac{\Delta t}{\mathrm{day}}$

Table 5.1: Parameters and model states transferred from the LandscapeDNDC biogeochemistry model to the CMF hydrology model in the setup phase of the simulation, with the corresponding unit conversion where applicable. The LandscapeDNDC biogeochemistry model sends the data in step 3 of Alg. 23, and the CMF hydrology model receives the data in step 4 of Alg. 24.

quantity transferred CMF ⟵ LandscapeDNDC	units [·] and conversion factor α [CMF] ⟵ [LandscapeDNDC] × α				
solutes on surface	kg	⟵	$\frac{\text{kg}}{\text{m}^2}$	×	cell surface area
solutes in layers	kg	⟵	$\frac{\text{kg}}{\text{m}^2}$	×	cell surface area
solutes in precipitation	$\frac{\text{kg}}{\text{m}^3}$	⟵	$\frac{\text{g}}{\text{m}^3}$	×	10^{-3}
harvest event	$\{0, 1\}$	⟵	$\{0, 1\}$		
leaf area index (LAI)	$\frac{\text{m}^2}{\text{m}^2}$	⟵	$\frac{\text{m}^2}{\text{m}^2}$		
vegetation height	m	⟵	m		
canopy closure	%	⟵	%		
root fraction	%	⟵	%		
ice fraction	%	⟵	%		
current air temp.	°C	⟵	°C		
max. air temp.	°C	⟵	°C		
min. air temp.	°C	⟵	°C		
ground temp.	°C	⟵	°C		
relative vapor pressure	Pa	⟵	% rel. humidity	×	10^{-2} abs. vapor pressure
wind speed	$\frac{\text{m}}{\text{s}}$	⟵	$\frac{\text{m}}{\text{s}}$		
radiation	$\frac{10^6 \text{J}}{\text{day m}^2}$	⟵	$\frac{\text{W}}{\text{m}^2}$	×	0.0864
precipitation	$\frac{10^{-3}\text{m}}{\text{day}}$	⟵	$\frac{\text{m}}{\Delta t}$	×	1000 $\frac{\Delta t}{\text{day}}$

Table 5.2: Parameters and model states transferred from the LandscapeDNDC biogeochemistry model to the CMF hydrology model in each time step of the simulation, with the corresponding unit conversion where applicable. The LandscapeDNDC biogeochemistry model sends the data in step 6 of Alg. 23, and the CMF hydrology model receives the data in step 7 of Alg. 24.

quantity transferred CMF ⟶ LandscapeDNDC	units [·] and conversion factor α [CMF] ⟶ [LandscapeDNDC] × α				
water on canopy	m^3	⟶	$\frac{\text{m}^3}{\text{m}^2}$	×	1 / cell surface area
snow on surface	m^3	⟶	$\frac{\text{m}^3}{\text{m}^2}$	×	1 / cell surface area
water on surface	m^3	⟶	$\frac{\text{m}^3}{\text{m}^2}$	×	1 / cell surface area
water in layers	$\frac{\text{m}^3}{\text{m}^3}$	⟶	$\frac{\text{m}^3}{\text{m}^3}$		
ice fraction	%	⟶	%		
solutes on surface	kg	⟶	$\frac{\text{kg}}{\text{m}^2}$	×	1 / cell surface area
solutes in layers	kg	⟶	$\frac{\text{kg}}{\text{m}^2}$	×	1 / cell surface area
infiltration flux	$\frac{\text{m}^3}{\text{day}}$	⟶	$\frac{\text{m}}{\Delta t}$	×	Δt / (day × cell surface area)
water throughfall	$\frac{\text{m}^3}{\text{day}}$	⟶	$\frac{\text{m}}{\Delta t}$	×	Δt / (day × cell surface area)
solute throughfall	$\frac{\text{kg}^3}{\text{day}}$	⟶	$\frac{\text{kg}}{\Delta t\ \text{m}^2}$	×	Δt / (day × cell surface area)
water discharge	$\frac{\text{m}^3}{\text{day}}$	⟶	$\frac{\text{m}}{\Delta t}$	×	Δt / (day × cell surface area)
solute discharge	$\frac{\text{kg}}{\text{day}}$	⟶	$\frac{\text{m}}{\Delta t}$	×	Δt / (day × cell surface area)

Table 5.3: Parameters and model states transferred from the CMF hydrology model to the LandscapeDNDC biogeochemistry model in each time step of the simulation, with the corresponding unit conversion where applicable. The CMF hydrology model sends the data in step 12 of Alg. 24, and the LandscapeDNDC biogeochemistry model receives the data in step 7 of Alg. 23.

5.4 Numerical experiments

This section is devoted to the numerical experiments with our hydrology-biogeochemistry coupling. We present two scenarios, one focusing on nitrous oxide greenhouse gas emissions, and one focusing on nitrate leaching to water bodies. Furthermore, we present parallel performance tests of the coupled model system. Both scenarios and the performance tests have already been published in our works [108, 64, 92]. We review these results in the following sub-sections in the chronological order of their publication. We present the nitrous oxide emission scenario in Sec. 5.4.1, the performance tests in Sec. 5.4.2, and the nitrate leaching scenario in Sec. 5.4.3.

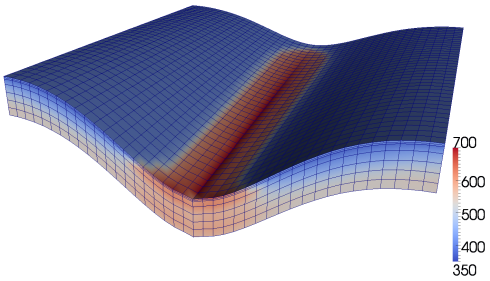
5.4.1 Soil N_2O emission scenario

The first scenario was designed as a numerical experiment to demonstrate the feasibility of our coupling approach. Here we present the conception and the results of this numerical experiment, details can be found in our previously published work [108].
The computational domain was chosen as a virtual landscape in the form of a valley with a slope of 5%. The discretization details are stated in Tab. 5.4. A Dirichlet boundary condition with hydraulic head set to 0.1 m below surface was used to model an outflow at the lowest part of the domain. All other soil boundary parts were equipped with no-flow conditions. At the

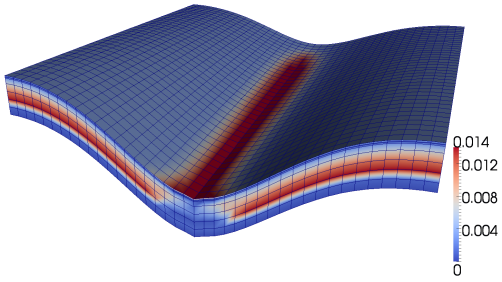
	entities	physical extent per entity
horizontal	40×41 soil columns	10m×10m
vertical	canopy, surface water, 8 soil layers	soil layers top to bottom: 5, 5, 20, 30, 30, 30, 30, 50 cm

Table 5.4: Discretization parameters of the computational domain shown in Figs. 5.2a and 5.2b.

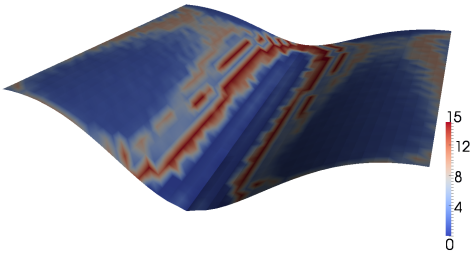
surface, open water and canopy water elements were attached to the top soil layer to account for surface runoff, rainfall interception, and infiltration. In this scenario we considered transport only for NO_3^- as it was sufficient to demonstrate the effect of lateral exchange on N_2O emissions. Figures 5.2a to 5.2d show the results after 420 simulated days. Precipitation and subsequent infiltration and redistribution of soil water created a saturated zone at the bottom of the valley as shown in Fig. 5.2a. Nitrate was produced through mineralization and nitrification of plant litter in upper soil layers, and transported into deeper layers and towards the valley (Fig. 5.2b). Notably, NO_3^- concentrations were highest in medium depths of the flanks, and near the valley surface. These were exactly those target regions where nitrate was transported to, but which were not saturated, i.e. aerobic regions where nitrification was active and denitrification was inhibited due to the presence of oxygen. In contrast, NO_3^- concentrations were very low in the deepest layers of the flanks and in all but the top layers of the valley. This was due to the high saturation of these regions inducing anaerobic conditions where nitrification is inhibited, but denitrification was active. Figure 5.2c shows the nitrous oxide emission pattern. The emission pattern was clearly pronounced at the boundary of the saturated zone in the valley. This was exactly the transition zone between aerobic and anaerobic conditions where the denitrification pathway was intermitted and N_2O was formed. The dinitrogen (N_2) emission pattern in Fig. 5.2d was concentrated at the saturated region in the valley where denitrification can completely reduce nitrate.
This scenario successfully demonstrated the feasibility of our coupling approach using the biogeochemical model system LandscapeDNDC together with the hydrology model CMF. The explicit consideration of lateral movement of water and transport of solutes results in more realistic emission patterns. In particular, our model coupling approach enables to consider the indirect



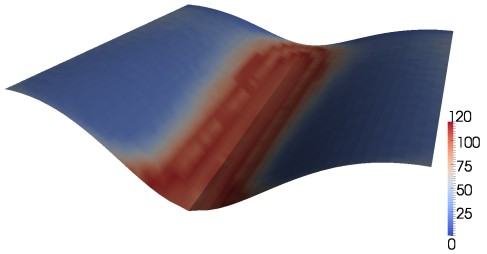
(a) Soil water content [mm/m^3].



(b) Nitrate (NO_3) concentration [kg N/ha] in the soil water.



(c) Spatial distribution of accumulated nitrous oxide (N_2O) emissions [kg N/ha].



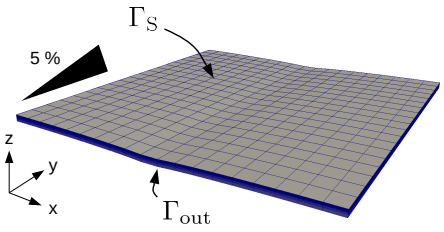
(d) Spatial distribution of accumulated dinitrogen (N_2) emissions [kg N/ha].

Figure 5.2: Results after 420 simulated days.

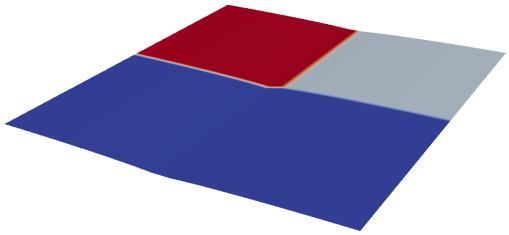
effects which are known, but which were so far hardly reflected in models. Moreover, the combination of a detailed biogeochemistry model coupled to a dedicated hydrology model allows for the previously unmatched capability of catching local emission events, i.e. hot spots and hot moments.

5.4.2 Parallel performance tests

We briefly review our parallel performance tests published in [64]. As shown in Fig. 5.3a, we used a similar domain as in the N_2O emission scenario described above. Our goal was to asses the ability or our coupling approach using OpenPALM to address the issue of the models' different computational demands. To this end, we ran test series on two different domain sizes with varying parallel configurations of the hydrology model, since it requires by far more computing power than the biogeochemistry model. The spatial discretization lead to 34,800 unknowns for the small test case, and to 870,000 unknowns for the large test case. Total simulation time was three years with a time step size of one hour, resulting in 26,280 time steps. Figures 5.4 and 5.5 show the state of the system after more than 1.5 years.
For our performance tests, we ran the simulation both with our new concurrent operator splitting scheme, as well as with the consecutive scheme for comparison. We report average runtimes in seconds per time step taken from the first 96 hours of the simulation. For both test cases, we measured the runtime of the consecutive and of the concurrent operator splitting scheme. In addition, we report the individual runtime of the biogeochemistry model. Figures 5.6a and 5.6b show the runtimes for the two test cases. The green line represents the individual runtime of the biogeochemistry model, and the blue line represents the runtime of the coupled application



(a) Computational domain of the small test case with outflow boundary at the forefront.



(b) Anthropogenic land use on the three domain parts. Grassland (blue), intensive maize cultivation (red) and extensive maize cultivation (gray).

Figure 5.3: Domain and land use.

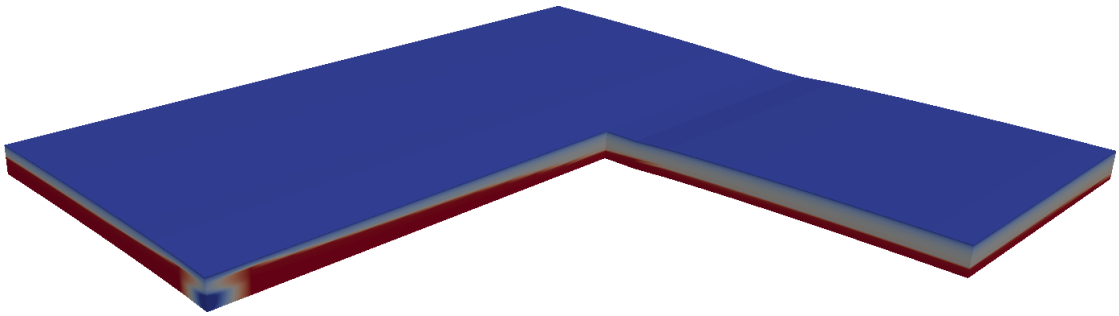


Figure 5.4: Cut view into the domain showing the water content of the soil after 16,348 hours. The lower part of the soil is fully saturated (red color) and the water table balances the slope of the domain. The saturation drops (gray to blue color) near the soil surface and at the outlet.

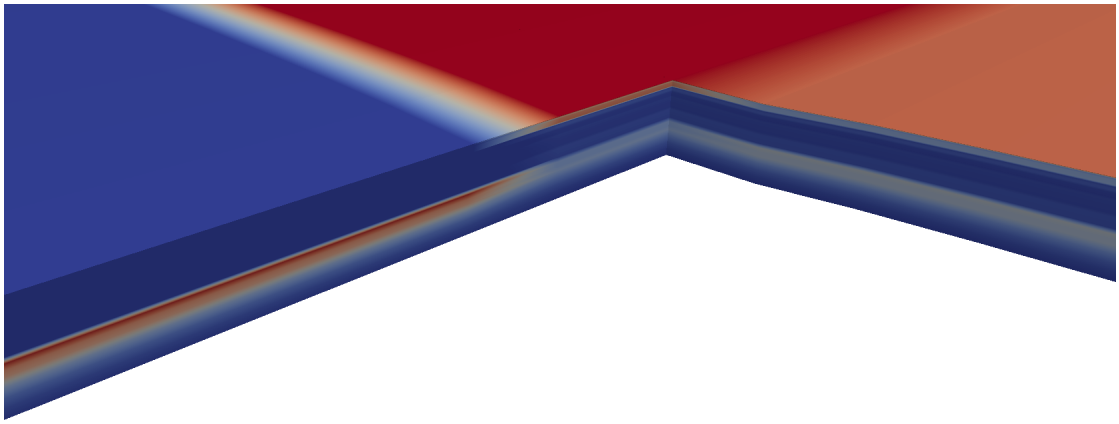
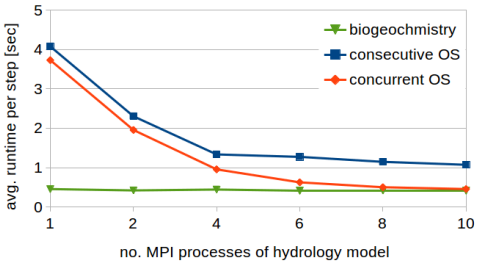
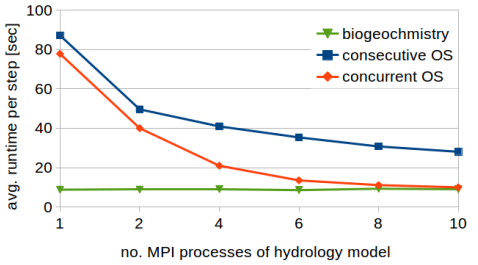


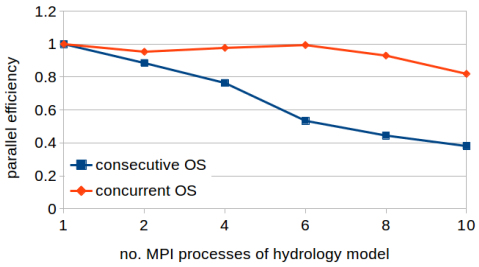
Figure 5.5: Zoomed cut view into the domain showing the nitrate (NO_3^-) concentration after 15,013 hours. High concentration near the soil surface in the area with intensive maize cultivation (dark red), medium concentration in the area with extensive maize cultivation (light red), and low concentration on the grassland (blue). Percolation and downstream transport of nitrate can cause high concentrations in lower soil layers.



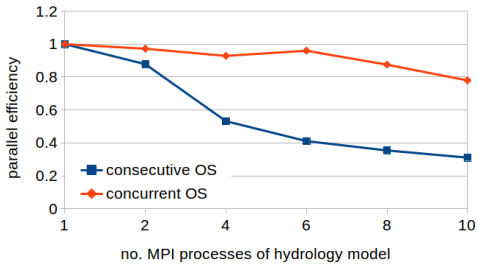
(a) Average runtime per time step for the small test case.



(b) Average runtime per time step for the large test case.



(c) Parallel efficiency for the small test case.



(d) Parallel efficiency for the large test case.

Figure 5.6: Runtimes and efficiency of the consecutive and the concurrent operator splitting schemes for the small and large test cases.

using the consecutive scheme. Therefore, the difference between the green and the blue line is the individual runtime of the hydrology model. The time needed for the data transfer between the models is negligible compared to the computation time. From the $p = 1$ cases, one can see that the hydrology model demands for approximately 8 to 9 times the computational effort of the biogeochemistry model. When using more processes for the hydrology model, the runtimes for the coupled applications decrease for both operator splitting schemes. The consecutive scheme has a runtime which is the sum of the two models. In contrast, the concurrent scheme has a runtime equal to the slower of the two models. For $p = 10$, the concurrent scheme shows nearly the same runtime for coupled application as for the biogeochemistry model alone. That means, the parallelization of the hydrology model yields a speedup for its execution such that the runtimes of both models are balanced. We calculated the parallel efficiency from the runtime measurements as $E(p) = T(1)/\big[pT(p)\big]$, where p is the number of MPI processes of the hydrology model, and $T(p)$ is the corresponding runtime of the coupled application. Shown in Figures 5.6c and 5.6d, the graphs illustrate the advantage of the concurrent scheme. It maintains an efficiency of approximately 80 % for the $p = 10$ case where the runtimes of the two models are balanced, while the efficiency of the consecutive scheme drops to less than 40 %.

5.4.3 Vegetated buffer strip scenario

In our second scenario we address the impact of vegetated buffer strips on the lateral transport of nutrients and the related input to water bodies through leaching, and on N-trace gas emissions. Here, we briefly introduce the setup of the study which has been published in [92]. In the sense of this work, vegetated buffer strips are patches of land covered with vegetation which separate

areas under cultivation from water bodies such as streams or rivulets, or from other areas. Due to their capability to retain nitrate and other nutrients, vegetated buffer strips are an option to mitigate nitrate leaching from agricultural soils to water bodies. The scenario domain is a 100 $\times$100 meter domain of one meter depth with 5% downslope towards south-east. The study area is subject to agricultural land management including different crops and N fertilizer application. Vegetated buffer strips are located along the south and east boundary of the domain. Different buffer strip sizes ranging from zero (no buffer) to 20 meters were tested. The main results reported in [92] are largely increased denitrification rates and corresponding N_2 emissions from buffer strips with high water content, and reduced nitrate leaching.
The aforementioned work provides quantitative results on N_2 and N_2O emission inventories and nitrate loss to water bodies.It used the consecutive operator splitting scheme, and a time step size of one hour. We complement the study by using the concurrent operator splitting scheme implemented with OpenPALM, and the domain decomposition parallelization of CMF in our simulations. This allows for a refinement of the time step size to 30 minutes while at the same time speeding up the simulations. Figure 5.7 shows qualitative results for three model runs

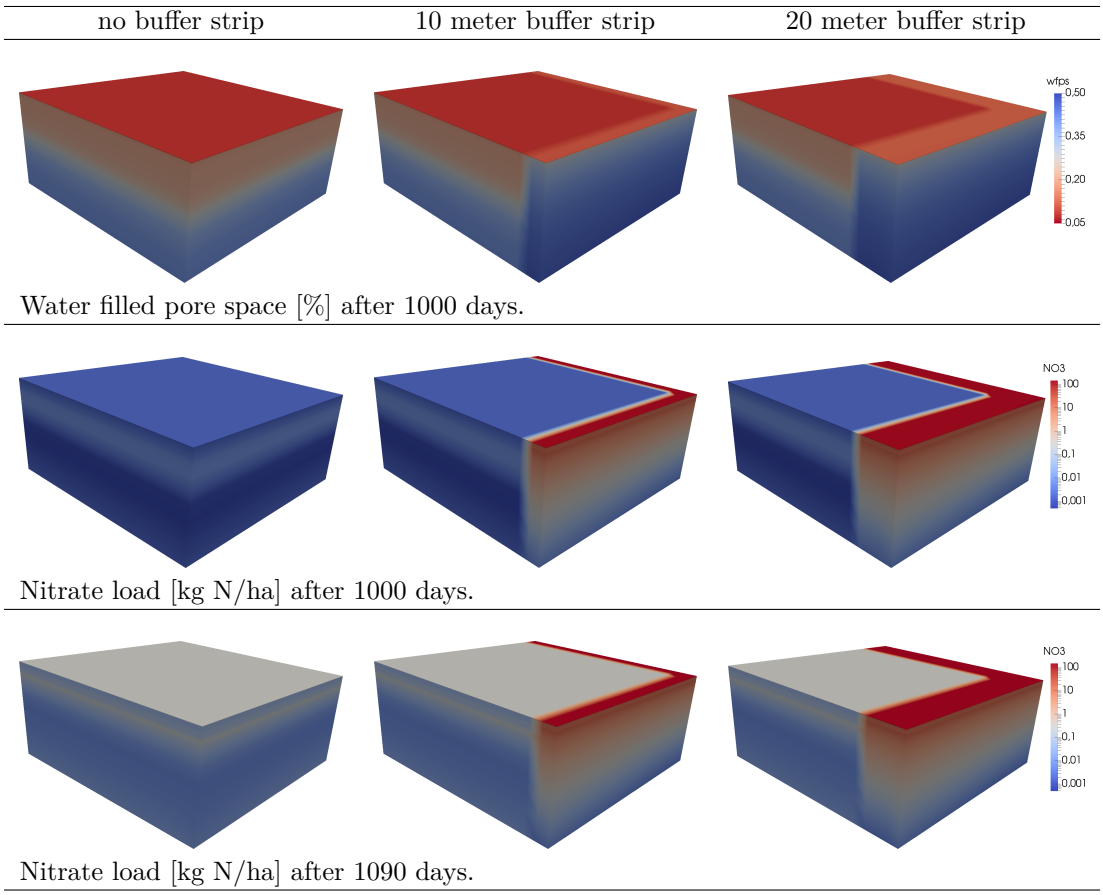


Figure 5.7: Simulation results for buffer strip sizes 0, 10 and 20 meters (left to right). Top row: water filled pore space after 1000 days. Middle row: nitrate load after 1000 days. Bottom row: nitrate load after 1090 days. The plots are scaled by factor 10 in vertical direction for better visibility.

without buffer, or with 10 or 20 meters buffer width, respectively. One can see that the buffer strip holds significantly more moisture than other parts of the soil. Also nitrate retention in the buffer strip is clearly visible. The high water content and nitrate availability in the buffer strip causes increased denitrification rates. This leads to the reported higher N_2 emissions and reduced nitrate leaching.

5.5 Résumé

We applied our multiphysics model coupling approach for the simulation of nutrient cycles in soils. We showed the coupling of the hydrology model CMF with the biogeochemical model LandscapeDNDC to assess nutrient and emission inventories in different scenarios. In this Chapter we presented new developments and qualitative results, and we reviewed our previously published contributions. We demonstrated the ability of our approach to capture indirect and local effects in ecosystem simulations. The spatial extent of our scenarios investigated so far was moderate, which is natural in this early stage of development, but nevertheless appropriate to give a proof of concept. Moreover, performing simulations for the relevant temporal extent of years to decades with small time steps of hours or less is evenly challenging. We showed that this can be successfully addressed with the proposed multiphysics simulation technique due to its ability to use high performance computing effectively. Possible future work could address the parallelization of all model parts including LandscapeDNDC. This would allow for simulations of larger domains and higher spatial resolution.

6 Conclusion

In this work we presented a new approach to nutrient cycle simulations for terrestrial ecosystems. We considered a coupling of dedicated models for the involved biogeochemical and hydrological processes leading to a system of ordinary and partial differential equations. The system describes the temporal evolution of a common state variable under the simultaneous influence of the coupled models. For the time integration of the coupled models, we proposed specific operator splitting schemes by means of a composition of one step methods. These schemes use one step methods as local model integrators to compose global solutions on a global time grid for the coupled system. We established two variants of composed one step methods, a consecutive operator splitting and a concurrent operator splitting, respectively. In the consecutive variant, the output of one model integrator is used as input to another model integrator. In the concurrent variant, models are integrated independently of each other until the next synchronization point is reached. The consecutive operator splitting resembles the well-known Lie(-Trotter) splitting in our setup, whereas the concurrent operator splitting has no classical counterpart. The main results of Chap. 2 are the proofs of first order in time consistency and convergence of the composed one step methods with respect to the global system. These proofs solely assume continuity, boundedness and Lipschitz conditions for the functional model representation in the abstract formulation, and consistency and Lipschitz continuity for the one step methods used on the local model level. We presented numerical experiments on the convergence of the composed one step methods using a natural convection scenario in Chap. 3. The results accurately show the expected first order in time convergence for both the consecutive and the concurrent variant in different flow regimes.

To facilitate the model coupling, we employed the OpenPALM software coupler tool. It enables the coupling of dedicated model codes, controls their execution and manages data transfer between models. We extended the data transfer mechanism of OpenPALM by means of dynamic distributors, which is the main result of Chap. 4. Our development allows to dynamically change data sizes and data distributions among models during the simulation, while at the same time keeping OpenPALM's internal routing table consistent. This feature supports the online configuration and adaption of the data transfer mechanism during runtime. Without dynamic distributors, the configuration needed to be derived from offline tests before the simulation could be started, and it could not be adapted during runtime. Furthermore, the OpenPALM coupling technology allows to allocate adequate computing resources to the individual models. This is essential to effectively use high performance computing and to achieve a balanced parallel setup if models demand for different computing resources. We presented performance tests using the natural convection scenario where we achieved substantial improvements of the parallel efficiency with the concurrent operator splitting scheme compared to a monolithic solver.

In Chap. 5, we presented the modeling and implementation of the biogeochemical and hydrological processes which we considered in our nutrient cycle simulations. We used the composed one step methods and the dynamic distributors feature of OpenPALM to realize the model coupling approach in a greenhouse gas emission scenario, and in a nitrate leaching scenario. We demonstrated that our approach allows to capture indirect and local effects which was previously unfeasible using standard simulation techniques. Furthermore, we showed that our coupling technique enables an effective use of high performance computing.

Our techniques are not restricted to the simulation of nutrient cycles, but they are rather usable for model coupling in general, using any number and granularity of models. The composed one step methods allow to propagate the models separately between global time steps. They only require the local model integrators to represent one step methods with respect to the global time grid. This allows to use any appropriate numerical integration scheme, and also to compute sub-steps, as long as the local model integrators synchronize at the global time steps. It is also possible to stack the composed one step methods, which might be useful if two models require a tight coupling and a third model only needs to synchronize with larger time steps. We proved first order in time consistency and convergence of the composed one step schemes, provided that the local model integrators are of first order or higher. Since the biogeochemical model integrator used in the scenarios is of first order, we did not investigate if higher order composed one step methods are possible. As stated in the concluding remarks of Chap. 2, we assume that higher order consecutive schemes are possible, while the existence of higher order concurrent schemes is unclear.
The dynamic distributors feature of OpenPALM is also not limited to nutrient cycle simulations, but a general purpose technique. We used this feature in the hydrology model for the online configuration of the parallel communication routines to match the domain decomposition which is determined at runtime. Other use cases include coarsening or refinement, and load balancing of the coupled models during runtime. The dynamic distributors are integrated into the communication routines and thus available to any software coupling application in the framework of OpenPALM.
To conclude, our developments to advance nutrient cycle simulations resulted in general purpose techniques for multiphysics model coupling. The issues we encountered are typical for model coupling problems, and the proposed methodology is suitable to address them in a wide range of applications. In this sense, our approach is an enabling technique to tackle interdisciplinary challenges by means of multiphysics simulations.

Bibliography

[1] ISO/IEC 1539:1980 Fortran 77 Standard, 1980.

[2] ISO/IEC 1539:1991 Fortran 90 Standard, 1991.

[3] BLAS Technical Forum Standard. *International Journal of High Performance Computing Applications*, 2001.

[4] ISO/IEC 9899:2011 Programming Language C, 2011.

[5] MPI: A Message-Passing Interface Standard, Version 3.0, 2012.

[6] ISO/IEC 14882:2014(E) Programming Language C++, 2014.

[7] R. Aris. *Vectors, Tensors, and the Basic Equations of Fluid Mechanics*. Dover, 1989.

[8] S.A. Blagodatsky and O. Richter. Microbial growth in soil and nitrogen turnover: a theoretical model considering the activity state of microorganisms. *Soil Biol. Biochem.*, 30(13):1743–1755, 1998.

[9] D. Braess. *Finite Elemente*. Springer, 2007.

[10] H.W. Broer, G.B. Huitema, and M.B. Sevryuk. Quasi-Periodic Motions in Families of Dynamical Systems. *Lect. Notes Math.*, 1996.

[11] R.H. Brooks and A.T. Corey. Hydraulic Properties of Porous Media. *Hydrol. Pap. No. 3, Colorado State University*, 1964.

[12] W. Brutsaert. *Hydrology*. Cambridge University Press, 2005.

[13] S. Buis, A. Piacentini, D. Declat, and the PALM Group. Palm: A computational framework for assembling high-performance computing applications. *Concurr. Comput. Pract. Exp.*, 18:231–245, 2006.

[14] K. Butterbach-Bahl, E.M. Baggs, M. Dannenmann, R. Kiese, and S. Zechmeister-Boltenstern. Nitrous oxide emissions from soils: how well do we understand the processes and their controls? *Philos. Trans. R. Soc. B*, 368, 2013.

[15] F. Stange K. Butterbach-Bahl H. Papen C. Li, J. Aber. A process-oriented model of N_2O and NO emissions from forest soils: 1. Model development. *J. Geophys. Res.*, 105(D4):4369–4384, 2000.

[16] R. Harris C. Li, S. Frolking. Modeling carbon biogeochemistry in agricultural soils. *Glob. Biogeochem. Cycles*, 8(3):237–254, 1994.

[17] T.A. Frolking C. Li, S. Frolking. A Model of Nitrous Oxide Evolution From Soil Driven by Rainfall Events: 1. Model Structure and Sensitivity. *Geophys. Res.*, 97:9759–9776, 1992.

[18] J.R. Cannon. *The One-Dimensional Heat Equation*. Addison-Wesley, 1984.

[19] P.J. Channell and W.F. Neri. *Integration Algorithms and Classical Mechanics*, chapter A brief introduction to symplectic integrators, pages 45–58. AMS, 1996.

[20] V.T. Chow. *Open-Channel Hydraulics*. McGraw-Hill, 1959.

[21] J.C. Chrispell, V.J. Ervin, and E.W. Jenkins. A fractional step θ-method for convection-diffusion problems. *J. Math. Anal. Appl.*, 333:204–218, 2007.

[22] S.D. Cohen and A.C. Hindmarsh. CVODE, A Stiff/Nonstiff ODE Solver in C. *Comput. Phys.*, 1996.

[23] R. Conrad. Soil Microorganisms as Controllers of Atmospheric Trace Gases. *Microbiol. Reviews*, 60(4):609–640, 1996.

[24] S. Klatt I. Santabarbara-E. Haas R. Wassmann-C. Werner-R. Kiese K. butterbach-Bahl D. Kraus, S. Weller. How well can we assess impacts of agricultural land management changes on the total greenhouse gas balance (CO_2, CH_4 and NO_2) of tropical rice-cropping systems with a biogeochemical model? *Agriculture Ecosys. Environ.*, 224:104–115, 2016.

[25] V. Dorodnitsyn. *Symmetries and Integrability of Difference Equations*, chapter Continuous symmetries of finite-difference evolution equations and grids, pages 103–112. AMS, 1996.

[26] J. Douglas and H.H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Trans. Amer. Math. Soc.*, 82:421–439, 1956.

[27] A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Springer, 2004.

[28] D. Etling. *Theoretische Meteorologie*. Springer, 2008.

[29] M.E. Fenn J. Simunek F. Yuan, T. Meixner. Impact of transient soil water simulation to estimated nitrogen leaching and emission at high- and low-deposition forest sites in Southern California. *Geophys. Res.*, 116:G03040, 2011.

[30] K. Feng. Symplectic, contact and volume-preserving algorithms. In Z.C. Shi and T. Ushijima, editors, *Proc. 1st China-Japan Conf. Numer. Math.* World Scientific, 1993.

[31] K. Feng. Contact algorithms for contact dynamical systems. *J. Comput. Math.*, 1998.

[32] J.A.C. Fortescue. *Environmental Geochemistry: A Holistic Approach*. Springer, 1980.

[33] F.-M. Breon W. Collins J. Fuglestvedt J. Huang D. Koch J.-F. Lamarque-D. Lee B. Mendoza T. Nakajima A. Robock G. Stephens T. Takemura H. Zhang G. Myhre, D. Shindell. *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, chapter Anthropogenic and Natural Radiative Forcing. Cambridge University Press, 2013.

[34] L. Bakken P. Leffelaar L.E. Haugen G. Schurgers, P. Dörsch. Modelling soil anaerobiosis from water retention characteristics and soil respiration. *Soil Biol. Biochem.*, 38:2637–2644, 2006.

[35] R. Glowinski and J. Periaux. Numerical methods for nonlinear problems in fluid dynamics. *Proc. Intern. Seminar on Scientific Supercomputing, Paris, Feb. 2-6*, 1987.

[36] E. Haas, S. Klatt, A. Froehlich, P. Kraft, C. Werner, R. Kiese, R. Grote, L. Breuer, and K. Butterbach-Bahl. LandscapeDNDC: a process model for simulation of biosphere-atmosphere-hydrosphere exchange processes at site and regional scale. *Landsc. Ecol.*, 28:615–636, 2013.

[37] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 2008.

[38] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, 2010.

[39] B.C. Hall. *Lie Groups, Lie Algebras, and Representations*. Springer, 2003.

[40] M. Heinen. Simplified denitrification models: Overview and properties. *Geoderma*, 133:444–463, 2006.

[41] H. Heuser. *Lehrbuch der Analysis, Teil 2*. Teubner, 12 edition, 2002.

[42] A.C. Hindmarsh and R. Serban. User Documentation for CVODE v2.7.0, 2012.

[43] G.A. O'Connor H.L. Bohn, B.L. McNeal. *Soil Chemistry*. John Wiley & Sons, 1979.

[44] J. Holst, R. Grote, C. Offermann, J.P. Ferrio, A. Gessler, H. Mayer, and H. Rennenberg. Water fluxes within beech stands in complex terrain. *Int. J. Biometeorol.*, 54:23–36, 2010.

[45] O. Ippisch. *Contributions to the large-scale Simulation of Flow and Transport in Heterogeneous Porous Media*. Habilitation, Department of Mathematics and Computer Science, Heidelberg University, Germany, 2016.

[46] A. Iserles, H.Z. Munthe-Kaas, S.P. Norsett, and A. Zanna. Lie group methods. *Acta Numerica*, 9:215–365, 2000.

[47] G. Sun C. Trettin J. Cui, C. Li. Linkage of MIKE SHE to Wetland-DNDC for carbon budbudget and anaerobic biogeochemistry simulation. *Biogeochem.*, 72:147–167, 2005.

[48] T. Jahnke and C. Lubich. Error bounds for exponential operator splittings. *BIT Numerical Mathematics*, 40:735–744, 2000.

[49] C.A. Federer J.D. Aber. A generailzed, lumped-parameter model of photosynthesis, evapotranspiration and net primary production in temperate and boreal forest ecosystems. *Oecologia*, 92:463–474, 1992.

[50] A.J.A. Vinten J.R.M. Arah. Simplified models of anoxia and denitrification in aggregated and simple-structured soils. *Europ. J. Soil Sci.*, 46:507–517, 1995.

[51] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1999.

[52] D.E. Keyes, L.C. McInnes, C. Woodward, W.D. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. Peters, D. Reynolds, B. Riviere, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth. Multiphysics Simulations: Challenges and Opportunities. Tech. Rep. ANL/MCS-TM-321, 2011.

[53] Y. Kim, Y. Seo, D. Kraus, S. Klatt, E. Haas, J. Tenhunen, and R. Kiese. Estimation and mitigation of N2O emission and nitrate leaching from intensive crop cultivation in the Haean catchment, South Korea. *Sci. Total Environ.*, 529:40–53, 2015.

[54] P. Kraft, K. B. Vaché, H.-G. Frede, and L. Breuer. A hydrological programming language extension for integrated catchment models. *Environ. Model. Softw.*, 26(6):828–830, 2011.

[55] D. Kraus, S. Weller, S. Klatt, E. Haas, R. Wassmann, R. Kiese, and K. Butterbach-Bahl. A new LandscapeDNDC biogeochemical module to predict CH4 and N2O emissions from lowland rice and upland cropping systems. *Plant Soil*, 386:125–149, 2015.

[56] S. Julich H.-G. Frede L. Breuer, K.B. Vache. Current concepts in nitrogen dynamics for mesoscale catchments. *Hydrol. Sci.*, 53(5):1059–1074, 2008.

[57] T. Lagarde, A. Piacentini, and O. Thual. A new representation of data-assimilation methods: The PALM flow-charting approach. *Q. J. R. Meteorol. Soc.*, 127:189–207, 2001.

[58] J.S.W. Lamb. Time-reversal symmetry in dynamical systems. *Physica D*, 112:1–328, 1998.

[59] P.A. Leffelaar. Dynamics of partial anaerobiosis, denitrification, and water in a soil aggregate: simulation. *Soil Sci.*, 146(6), 1988.

[60] P.A. Leffelaar and W.W. Wessel. Denitrification in a Homogeneous, Closed System: Experiment and Simulation. *Soil Sci.*, 146(5), 1988.

[61] C. Li. Modeling trace gas emissions from agricultural ecosystems. *Nutr. Cycl. Agroecosys.*, 58:259–276, 2000.

[62] M.J. Lighthill and G.B. Whitham. On kinematic waves, part I. Flood movement in long rivers. *Proc. R. Soc. Lond., Ser. A, Math. Phys. Sci.*, 229:281–316, 1955.

[63] A. Ostermann M. Hochbruck. *Exponential integrators*. Cambridge University Press, 2010.

[64] S. Klatt D. Kraus E. Haas R. Kiese K. Butterbach-Bahl P. Kraft L. Breuer M. Wlotzka, V. Heuveline. Parallel multiphysics simulations using OpenPALM with application to hydro-biogeochemistry coupling. In *accepted for Proceedings of 6th Int. Conf. on High Perform. Sci. Comput., March 16-20, 2015, Hanoi, Vietnam*, 2016.

[65] R.I. McLachlan, M. Perlmutter, and G.R.W. Quispel. On the Nonlinear stability of symplectic integrators. *Preprint*, 2001.

[66] R.I. McLachlan and G.R.W. Quispel. *Foundations of Computational Mathematics*, chapter Six lectures on geometric integration, pages 155–210. Cambridge University Press, 2001.

[67] R.I. McLachlan, G.R.W. Quispel, and N. Robidoux. Geometric Integration using discrete gradients. *Phil. Trans. Roy. Soc. A*, 357:1021–1046, 1999.

[68] R.I. McLachlan, G.R.W. Quispel, and G.S. Turner. Numerical integrators that preserve symmetries and reversing symmetries. *SIAM J. Numer. Anal.*, 35:586–599, 1998.

[69] E.A. Davidson M.K. Firestone. *Exchange of Trace Gases between Terrestrial Ecosystems and the Atmosphere*, chapter Microbial Basis of NO and N_2O Production and Consumption in Soil, pages 7–21. John Wiley & Sons, 1989.

[70] S. Molina-Herrera, R. Grote, I. Santabarbara-Ruiz, D. Kraus, S. Klatt, E. Haas, R. Kiese, and K. Butterbach-Bahl. Simulation of CO2 Fluxes in European Forest Ecosystems with the Coupled Soil-Vegetation Process Model LandscapeDNDC. *Forests*, 6, 2015.

[71] P. Molino. *Riemannian Foliations*. Birkhäuser, 1988.

[72] T. Morel, F. Duchaine, A. Thevenin, A. Piacentini, M. Kirmse, and E. Quemerais. *OpenPALM coupler version 4.1.4: User guide and training manual*, 2013.

[73] Y. Mualem. A New Model for Predicting the Hydraulic Conductivity of Unsaturated Porous Media. *Water Resources Research*, 12:513–522, 1976.

[74] A. Murua and J.M. Sanz-Serna. Order conditions for numerical Integrators obtained by composing simpler intergators. *R. Soc. London Philos. Trans. A*, 357:1079–1100, 1999.

[75] T.N. Narasimhan. Buckingham, 1907: An Appreciation. *Vadose Zone Journal*, 4:434–441, 2005.

[76] OpenMP Architecture Review Board. OpenMP Application Program Interface, Version 4.0, 2013.

[77] J.M. Ortega and W.C. Reinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[78] R. Scholes P. Lavelle, R. Dugdale. *Ecosystems and Human Well-being: Current State and Trends, Volume 1*, chapter Nutrient Cycling, pages 331–353. Islnd Press, 2005.

[79] D.W. Peaceman and H.H. Rachford. The numerical solution of parabolic and elliptic differential equations. *J. Soc. Indust. Appl. Math.*, 3(1):28–41, 1955.

[80] A. Piacentini and the PALM Group. PALM: A Dynamic Parallel Coupler. *Lecture Notes in Computer Science*, 2565:479–492, 2003.

[81] Y. Qu and C.J. Duffy. A semidiscrete finite volume formulation for multiprocess watershed simulation. *Water Resources Research*, 43:W08419, 2007.

[82] G.R.W. Quispel and D.I. McLaren. Explicit volume-preserving and symplectic Integrators for trigonometric polynomial flows. *Preprint*, 2002.

[83] C. Werner S. Wochele R. Grote K. Butterbach-Bahl R. Kiese, C. Heinzeller. Quantification of nitrate leaching from German forest ecosystems by use of a process oriented biogeochemical model. *Environ. Pollut.*, 159:3204–3214, 2011.

[84] R. Rannacher. Numerische Mathematik 1: Numerik Gewöhnlicher Differentialgleichungen, 2014.

[85] S. Reich. Numerical integration of the generalized Euler equations. *Technical report 93-20, Dept. Computer Science, U British Columbia*, 1993.

[86] S. Reich. Backward error Analysis for numerical integrators. *SIAM J. Numer. Anal.*, 36:1549–1570, 1999.

[87] E. Pattey R.F. Grant. Mathematical modeling of nitrous oxide emissions from an agricultural field during spring thaw. *Global Biogeochem. Cycles*, 13(2):679–694, 1999.

[88] G.R.W. Quispel R.I. McLachlan. What kinds of dynamics are there? Lie pseudogroups, dydynamic systems, and geometric integration. *Nonlinearity*, 2001.

[89] G.R.W. Quispel R.I. McLachlan. Splitting methods. *Acta Numerica*, pages 341–434, 2002.

[90] L.A. Richards. Capillary Conduction of Liquids through Porous Mediums. *Physics*, 1:318–333, 1931.

[91] R. Kiese C. Werner K. Butterbach-Bahl S. Blagodatsky, R. Grote. Modelling of microbial carbon and nitrogen turnover in soil with special emphasis on N-trace gases emission. *Plant Soil*, 346:297–330, 2011.

[92] P. Kraft L. Breuer M. Wlotzka-V. Heuveline-E. Haas-R. Kiese K. Butterbach-Bahl S. Klatt, D. Kraus. Exploring impacts of vegetated buffer strips on nitrogen cycling using a spatially explicit hydro-biogeochemical modmodel approach. *accepted for Environ. Model. Softw.*, 2016.

[93] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2 edition, 2000.

[94] W.E. Schiesser. *The Numerical Method of Lines: Integration of Partial Differential Equations*. Academic Press, 1991.

[95] Z.J. Shang. Resonant and diophantine step sizes in computing invariant tori of Hamiltonian systems. *Nonlinearity*, 13:299–308, 2000.

[96] J. Shi and Y.T. Yan. Explicitly integrable polynomial HHamiltonian and evaluation of Lie transformations. *Phys. Rev. E*, 48:3943–3951, 1993.

[97] C.T. Simmons. Henry Darcy (1803-1858): Immortalised by his scientific legacy. *Hydrogeology Journal*, 16:1023–1038, 2008.

[98] H. Sohr. *The Navier-Stokes Equations: An EleElement Functional Analytic Approach*. Birkhäuser, 2001.

[99] J.H. Spurk and A. Nuri. *Fluid Mechanics*. Springer, 2008.

[100] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5(3), 1968.

[101] M. Suzuki. Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations. *Phys. Lett. A*, 146:319–323, 1990.

[102] R. Temam. *Navier-Stokes Equations*. AMS Chelsea Publishing, 2001.

[103] Z. Tsuboi and M. Suzuki. Determining equations for higher-order decompositions of exponential operators. *Int. J. Mod. Phys. B*, 25:3241–3268, 1995.

[104] M.T. van Genuchten. A Closed Form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils. *Soil Sci. Soc. Am. J.*, 44:892–898, 1980.

[105] M. Zeeman P. Michna A. Zingg-N. Buchmann-L. Emmenegger W. Eugster, K. Zeyer. Methodical study of nitrous oxide eddy covariance measurements using quantum cascade laser spectrometery over a Swiss forest. *Biogeosciences*, 4:927–939, 2007.

[106] C. Werner, E. Haas, R. Grote, M. Gauder, S. Graeff-Hönninger, W. Claupein, and K. Butterbach-Bahl. Biomass production potential from Populus short rotation systems in Romania. *GCB Bioenergy*, 4(6):642–653, 2012.

[107] M. Wlotzka and V. Heuveline. A parallel solution scheme for multiphysics evolution problems using OpenPALM. *EMCL Prepr. Ser.*, 1, 2014.

[108] M. Wlotzka, V. Heuveline, S. Klatt, E. Haas, D. Kraus, K. Butterbach-Bahl, P. Kraft, and L. Breuer. *Handbook of Geomathematics*, chapter Simulation of Land Management Effects on Soil N_2O Emissions using a Coupled Hydrology-Biogeochemistry Model on the Landscape Scale. Springer, 2014.

[109] M. Wlotzka, T. Morel, A. Piacentini, and V. Heuveline. New features for advanced dynamic parallel communication routines in OpenPALM: Algorithms and documentation. *EMCL Prepr. Ser.*, 4, 2017.

[110] H Yoshida. Construction of higher order symplectic integrators. *Phys. Lett. A*, 150:262–268, 1990.