Geosci. Model Dev., 5, 1395–1405, 2012
www.geosci-model-dev.net/5/1395/2012/
doi:10.5194/gmd-5-1395-2012

Geoscientific Model Development

Implementation of multirate time integration methods for air pollution modelling

M. Schlegel[1], **O. Knoth**[1], **M. Arnold**[2], **and R. Wolke**[1]

[1]Leibniz Institute for Tropospheric Research, Permoserstraße 15, 04318 Leipzig, Germany
[2]Martin Luther University Halle-Wittenberg, Institute of Mathematics, 06099 Halle (Saale), Germany

Correspondence to: R. Wolke (wolke@tropos.de)

Received: 11 July 2011 – Published in Geosci. Model Dev. Discuss.: 15 November 2011
Revised: 5 September 2012 – Accepted: 9 October 2012 – Published: 12 November 2012

Abstract. Explicit time integration methods are characterised by a small numerical effort per time step. In the application to multiscale problems in atmospheric modelling, this benefit is often more than compensated by stability problems and step size restrictions resulting from stiff chemical reaction terms and from a locally varying Courant-Friedrichs-Lewy (CFL) condition for the advection terms. Splitting methods may be applied to efficiently combine implicit and explicit methods (IMEX splitting). Complementarily multirate time integration schemes allow for a local adaptation of the time step size to the grid size. In combination, these approaches lead to schemes which are efficient in terms of evaluations of the right-hand side. Special challenges arise when these methods are to be implemented. For an efficient implementation, it is crucial to locate and exploit redundancies. Furthermore, the more complex programme flow may lead to computational overhead which, in the worst case, more than compensates the theoretical gain in efficiency. We present a general splitting approach which allows both for IMEX splittings and for local time step adaptation. The main focus is on an efficient implementation of this approach for parallel computation on computer clusters.

1 Introduction

Atmospheric processes can be described using advection-diffusion-reaction equations. The advection term describes transport due to wind, diffusion describes turbulent mixing on spatial scales below the cell size. Both of these terms can efficiently be solved using explicit Runge-Kutta (RK) methods. So called Runge-Kutta-Chebychev methods were developed for the coupled treatment of advection-diffusion problems (Verwer, 1996), (Verwer et al., 2004). Depending on the specific simulation, the reaction terms may describe microphysical processes or chemical reactions of pollutants including source terms. These terms are usually stiff as characteristic times of involved processes differ significantly. Employing explicit methods is not efficient in this case as stability requirements limit the time step size to unpractically small values. Implicit methods have proven to be much more efficient for the chemistry problem.

Implicit/explicit (IMEX) splittings have been developed which allow for an efficient solution of advection-diffusion-reaction equations. Complementarily since the 1980s explicit multirate methods have been developed (e.g., Osher and Sanders, 1983; Tang and Warnecke, 2005) which allow for efficient solution of problems which can be split into non-stiff sub-systems with differing characteristic times, for example, advection on locally refined grids. Generally, multirate schemes result only in a significant reduction of computational costs if the amount to compute the part which requires smaller time steps is low in comparison to the total effort.

In the majority of current atmospheric models simple operator splittings are employed while the usage of multirate methods is rare. In Schlegel et al. (2009, 2012) we presented a generic splitting approach which can be employed to construct a multirate-IMEX scheme. The current paper is concerned with the efficient implementation of this scheme in the state-of-the-art Multiscale Atmospheric Chemistry and Transport model COSMO-MUSCAT (Wolke et al., 2004; Hinneburg et al., 2009) developed at the Institute for Tropospheric Research in Leipzig.

The remainder of this paper is structured as follows. First, we will mathematically outline a generic splitting scheme. The subsequent section is concerned with the focus of this paper, i.e., details of a practical implementation. We shall present the programme flow, details on data exchange and a balancing approach in separate subsections. Finally, we present two scenarios and discuss the obtained reduction of computational cost for each of them.

2 Mathematical preliminaries

In Schlegel et al. (2009) we presented a general splitting that may be employed to generate multirate methods, called *recursive flux splitting multirate* (RFSMR). The approach is based on an IMEX splitting presented by Knoth and Wolke (1998a), which we briefly outline here. These schemes can be seen as a higher-order generalisation of the popular source splitting approach (Verwer et al., 2002). Consider an equation

$$w' = F(w) + G(w).$$

In the context of this paper G represents advection with comparatively low Courant numbers. The other term, F, may represent diffusion-reaction or advection with higher Courant numbers.

Denote the time substeps in the explicit method by $\tau_i = t_0 + \Delta t c_i$ with c_i monotonically increasing with i. Then the algorithm computing an approximate solution w_1 for time t_1 from an approximate solution w_0 at time t_0 can be outlined as follows

$$W_1 = w_0, \tag{1}$$

$$r_i = \sum_{j=1}^{i-1} \left(a_{ij} - a_{i-1,j}\right) G\left(W_j\right), \tag{2}$$

$$v_i\left(\tau_{i-1}\right) = W_{i-1}, \tag{3}$$

$$\frac{dv_i}{\mathrm{d}\tau} = \frac{1}{c_i - c_{i-1}} r_i + F\left(v_i\right), \quad \tau \in \left[\tau_{i-1}, \tau_i\right], \ i = 2, ..., s+1, \tag{4}$$

$$W_i = v_i\left(\tau_i\right), \tag{5}$$

$$w_1 = W_{s+1}, \tag{6}$$

with s denoting the number of stages of an explicit Runge-Kutta (ERK) method with parameters (a, b, c) in standard RK notation. Additionally r_i denotes a source term correlated to the advection term G. For simplicity we define $a_{s+1,j} = b_j$, thus, avoiding a separate treatment of the summation stage. For stages i with $c_i = c_{i-1}$ we replace (3),...,(5) with a purely explicit step:

$$W_i = W_{i-1} + \Delta t r_i = W_{i-1} + \Delta t \sum_{j=1}^{i-1} \left(a_{ij} - a_{i-1,j}\right) G\left(W_j\right), \tag{7}$$

which we will call a *correction step*.

Defining $F(v) \equiv 0$ we obtain the underlying explicit method, subsequently called the *outer* method. Generally we require $w' = G(w)$ to be non-stiff.

Solving the *inner* system Eq. (4) with an implicit integrator leads to an IMEX splitting. The system $v' = r_i + F(v)$ then may be stiff. To obtain an explicit multirate method the inner system must be solved using an explicit Runge-Kutta method. In the latter context $v' = r_i + F(v)$ is required to be non-stiff, but it may impose stricter time step restrictions than G. This situation arises e.g., when transport is simulated on a locally refined grid. To make a distinction possible, we denote the parameters of the outer method with a superscript "O" in contrast to the parameters of the inner method (employed for the solution of Eq. 4) denoted by a superscript "I". An explicit multirate method based on the above splitting then reads

$$W_1 = w_0, \tag{8}$$

$$r_i = \sum_{j=1}^{i-1} \tilde{a}_{ij}^{\mathrm{O}} G\left(W_j\right), \tag{9}$$

$$V_{i,1} = W_{i-1}, \tag{10}$$

$$V_{i,k} = V_{i,k-1} + \Delta t \tilde{c}_i^{\mathrm{O}} \sum_{j=1}^{k-1} \tilde{a}_{kj}^{\mathrm{I}} \left(\frac{1}{\tilde{c}_i^{\mathrm{O}}} r_i + F\left(V_{i,j}\right)\right), \tag{11}$$

$$i = 2, ..., s^{\mathrm{O}} + 1, \ k = 2, ..., s^{\mathrm{I}} + 1, \tag{12}$$

$$W_i = V_{i,s^{\mathrm{I}}+1}. \tag{13}$$

with the tilde parameters denoting the increments per RK stage

$$\tilde{a}_{ij} = \begin{cases} a_{ij} - a_{i-1,j} & \text{if } i < s+1 \\ b_j - a_{s,j} & \text{if } i = s+1 \end{cases},$$

$$\tilde{c}_i = \begin{cases} c_i - c_{i-1} & \text{if } i < s+1 \\ 1 - c_s & \text{if } i = s+1 \end{cases},$$

for an explicit base method with s stages.

Note that the time step ratio R, i.e., the ratio of the time step of the outer method (macro time step) to the time step of the inner method (micro time step), depends only on the nodes c_i^{O} of the outer method. Formally the inner method is executed s^{O} times with time step sizes of $(\Delta t(c_2^{\mathrm{O}} - c_1^{\mathrm{O}}), \Delta t(c_3^{\mathrm{O}} - c_2^{\mathrm{O}}), ...)$. This is no severe limitation, however, as the inner method may for instance be replaced by a composition of n steps of size $\Delta t/n$ to obtain the desired time step ratio, see Table 1. Finally the splitting may be applied recursively to obtain a multirate-IMEX splitting, see Schlegel et al. (2012).

Based on a partitioned Runge-Kutta (PRK) formulation (see for instance Hairer et al., 1987) we have proven that the methods constructed using the algorithm (1),...,(6) are second order accurate in time if all of the employed base methods are at least of second order accuracy (Schlegel et al., 2012). Even third order accuracy can be obtained if the base methods themselves satisfy third order conditions and one

Table 1. RFMSMR(RK2a) – an example for a 2nd order multirate scheme with time step ratio $R = 2$; redundant stages omitted.

0	
1	1
	1/2 1/2

(RK2a)

outer base method

0				
1/2	1/2			
1/2	1/4	1/4		
1	1/4	1/4	1/2	
	1/4	1/4	1/4	1/4

inner base method

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

"slow" part

0					
1/2	1/2				
1/2	1/4	1/4			
1	1/4	1/4	1/2		
1	1/4	1/4	1/4	1/4	
	1/4	1/4	1/4	1/4	0

"fast" part

additional order condition is satisfied by the outer method,

$$\sum_{i=1}^{s^{\mathrm{O}}} \left(c_{i+1}^{\mathrm{O}} - c_i^{\mathrm{O}}\right) \sum_{j=1}^{i-1} \left(a_{i+1,j}^{\mathrm{O}} + a_{i,j}^{\mathrm{O}}\right) c_j^{\mathrm{O}} = \frac{1}{3}.$$

Third order accuracy in time has been documented by numerical tests and has also been proven formally. Order conditions for partitioned Runge-Kutta methods can be found for instance in Jackiewicz and Vermiglio (1998).

In order to apply this multirate approach to the advection equation, the advection operator must be split. Commonly a splitting by components is employed. Unfortunately the methods generated via RFSMR generally have unequal summation weights $b^{\mathrm{fast}} \neq b^{\mathrm{slow}}$ (see Table 1 for an example), so that the methods do not preserve the linear invariants of the system such as the total mass of pollutants. Mass conservation, however, is a strict requirement for atmospheric models. The solution of this problem is to employ a splitting by fluxes. Applied to a decomposition of the domain into slow and fast blocks, flux splitting means that every block computes the fluxes leaving its cells. Thus, fast fluxes leaving cells with a stricter local Courant-Friedrichs-Lewy (CFL) restriction are updated more frequently per macro time step than slow fluxes. As the individual cell outfluxes are computed exactly the same way and based on the same concentration vector as the corresponding cell influxes, this kind of splitting guarantees mass conservation independent from the partitioned time integration scheme.

3 Implementation details

As the name already suggests the recursive flux splitting multirate algorithm is especially suitable for recursive implementation. This kind of implementation makes it simple to exploit redundancies. The primary aim of multirate methods is to reduce computational cost, usually measured in evaluations of the right-hand side. An objection obvious to programmers is that the bottleneck in modern hardware is memory access rather than the actual computations on the CPU. This holds the more if a multi-node cluster is employed and data has to be exchanged across the network. Fortunately RFSMR can be implemented with only little memory overhead; additionally communication workload is reduced.

The algorithm is implemented in the *multi-scale atmospheric chemistry and transport model* MUSCAT, (Wolke and Knoth, 2000; Knoth and Wolke, 1998b) developed at the Leibniz Institute for Tropospheric Research in Leipzig. This model is used for scientific process studies as well as online coupled with a meteorological model for several air quality applications in local and regional areas. MUSCAT describes the transport as well as chemical transformations for several gas phase species and particle populations in the atmosphere. The spatial discretisation of the mass balance equations is performed by finite volume techniques on a hierarchical grid structure. A second order IMEX scheme is applied for the time integration. The step size control during the implicit integration leads to load imbalances. Therefore, a dynamic workload balancing is implemented (Wolke et al., 2004). This is done using the ParMetis libraries (Karypis et al., 2011; Karypis, 1999). The MUSCAT code is mainly written in FORTRAN90/95 and uses a few additional C libraries.

This section is organised as follows: first, we shall explain how data is organised in our model, then present the programme flow of local computations and finally explain data exchange strategies. Since the workload balancing gets more complicated as the programme complexity increases, we will also comment on this issue.

3.1 Data organization and spatial structure

In MUSCAT data is organised hierarchically: the three dimensional computational domain is decomposed statically into rectangular blocks. This decomposition is applied in

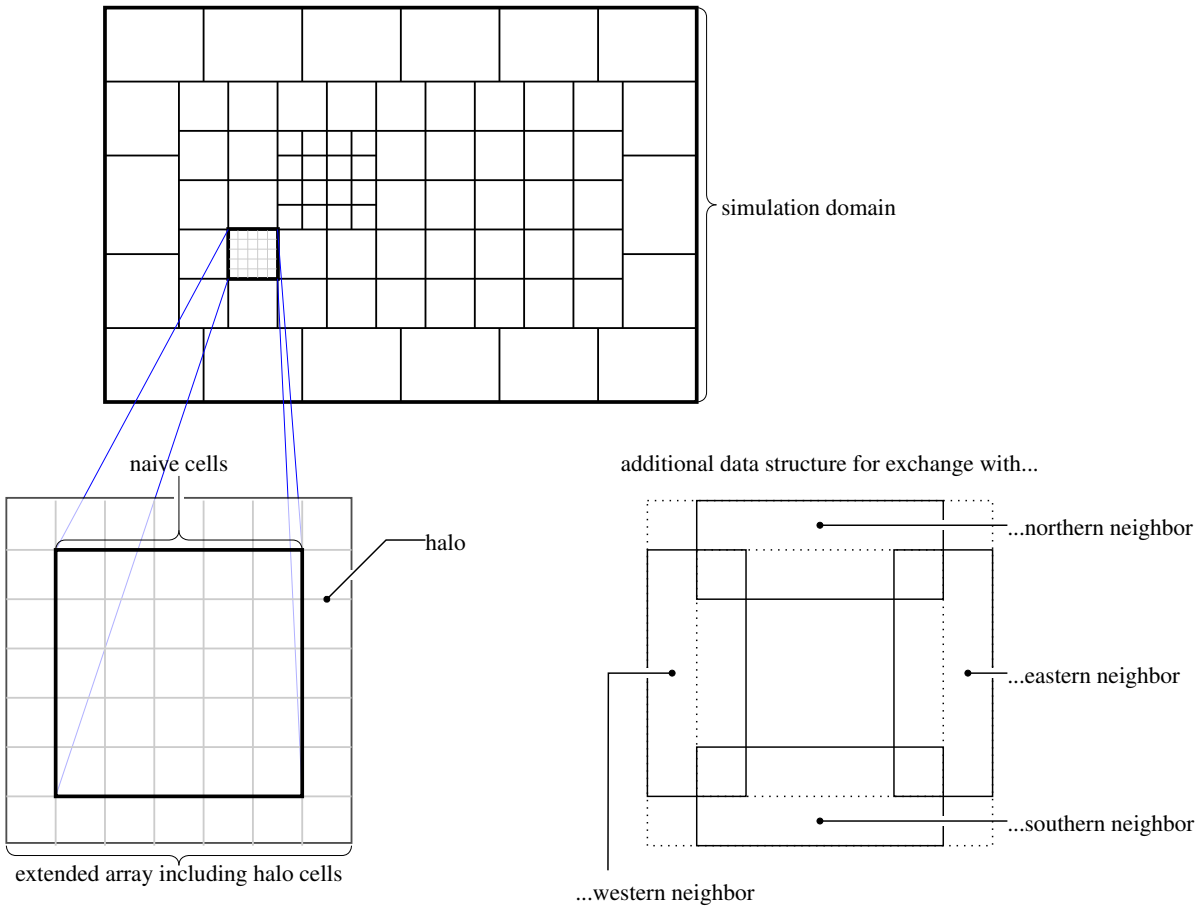

Fig. 1. Illustration of block structure for simulation domain as well as one generic block in MUSCAT.

horizontal direction only so that every block ranges from ground level through the top of the domain to simulate. An important feature is that each block may have its own spatial refinement level. Thus, selected regions may be examined in more detail. The cell size always is the macro cell size divided by an integer power of two. If two blocks are adjacent, their spatial refinement level is allowed to differ by a factor of two maximum. Cell size, adjacency to other blocks, temporal refinement level, etc., form the block meta data. Even for parallel processing meta data of all blocks is present on all processors. The main part of the data consists of concentration arrays and a number of *difference* arrays associated with the stage vectors of the employed explicit Runge-Kutta method. Opposed to the meta data, the main data of the block is present only on the processor the block is associated with. Additional variables include geometric data per cell (volume, extend per axis etc.) defined on initialisation and meteorological data such as wind speed or density of air. The latter may be provided by the COSMO model (Schättler et al., 2011; Steppeler et al., 2003) of the German weather service via online coupling or by a simple test driver.

The declaration of the concentration array is done in such a way that cell local values, i.e., the concentrations of the different tracers or species inside of a cell are directly adjacent in physical memory. The cell data in turn is organised so that cells within one column (i.e., cells at the same horizontal position) have minimal distance in memory. This layout is advantageous for the computations executed most frequently, i.e., cell local chemistry and column local (vertical) diffusion. A number of fully coupled implicit chemistry diffusion steps have to be performed per advection step. This part contributes mainly to the overall computational cost. All arrays have the same shape and dimension making vectorised operations possible.

We employ an *extended array* declaration for the individual blocks where the extended array includes both the actual cells of the block and the surrounding halo or ghost cells which are needed for coupling with adjacent blocks. Thus, all block local computations may be done on a logically cartesian array. The extended arrays are only used where necessary. For instance the *difference* arrays associated with the stage vectors are only defined for the inner cells. Additional data structures for the exchange of mass fluxes with

neighbouring blocks are needed along the respective boundaries, see Fig. 1. These data structures correspond to the source term r occurring in Eqs. (4) and (11). Employing a limited third order upwind spatial discretisation (Hundsdorfer and Verwer, 2003), the halo has to be one cell wide while the additional data structures overlap with the halo cells and the outermost row of actual cells.

Though the grid of the block is logically cartesian, its geometrical interpretation may be different. First of all, the simulation domain usually represents a volume above a spherical surface. Furthermore, perpendicularly to the interface the ghost cells have the extent of the actual cells they overlap with, see Fig. 2. The possible relations between actual cells and ghost cells representing the same physical volume are also marked in Fig. 2. The respective volume may be represented by:

a. one inner cell and one ghost cell,

b. two inner cells and one ghost cell of a coarser neighbour,

c. one inner cell and two ghost cells of a finer neighbour or

d. inner cell(s) and ghost cell(s) for each of two neighbours.

Case (d) only occurs at block corners and may be complicated for neighbour blocks with different spatial refinement levels. These different possible relations have to be taken into account when data is exchanged between blocks.

Keep in mind that the domain is decomposed in the horizontal direction only, so that all of the above holds not only for cells in one vertical layer, but also for the columns, ranging from the bottom through the top of the simulation domain.

3.2 Programme flow

In this section we shall discuss the MUSCAT programme flow. For simplicity we shall concentrate on the main integration loop, omitting initialisation, finalisation and output routines. Algorithm (1),...,(6) translates directly into an implementation. The following pseudo code evolves all blocks on the given time level from t_0 to $t_0 + \Delta t$. Here and subsequently we will employ the terms "time level" and "temporal refinement level" synonymously. In the pseudo code we also shortly write "level". The macro time step is equivalent to the lowest level.

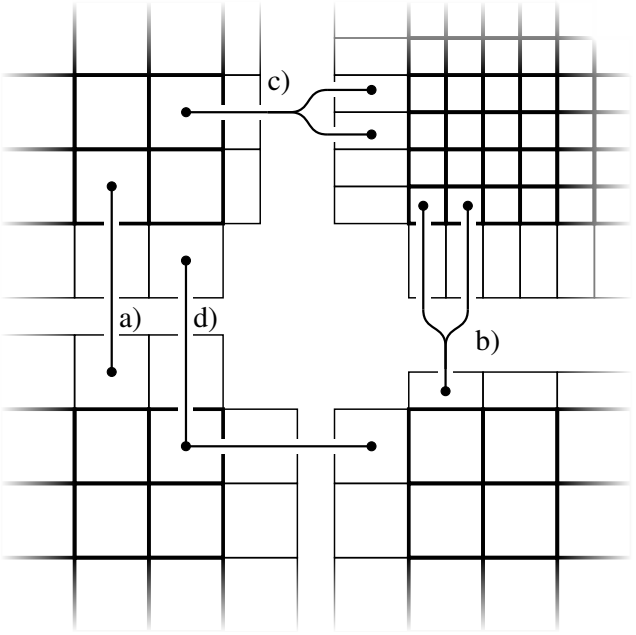

Fig. 2. Geometrical cell structure of adjacent blocks. Halo cells depicted with thinner contours. Connectors between blocks indicate multiple representations of the same physical volume.

```
1:  procedure RFSMR (t_0, Δt, level)
2:    for i = 2, ..., s + 1 ▷Loop over Runge-Kutta stages
3:      for all Processor-local blocks k do
4:        if [timelevel of Block k]= level then
5:          Compute local advective fluxes f_{i-1}
6:          r := Σ_{j=1}^{i-1} (a_{ij} − a_{i-1,j}) f_j
7:          r := r + r_(slow) · (c_i − c_{i-1})
8:          exchange fluxes with neighbour blocks
            on same level r_(equal) := r^neighbour
9:          r := r + r_(equal)
10:       end if
11:     end for
```

At this point, all advective fluxes on time level *level* are computed and known on all blocks containing the corresponding cell boundary. Note that for exchanges we write the variables without superscript if the variable is block local and with a superscript "neighbour" if the variable belongs to another block. The source term r (see line 7) corresponds to the source term r_i as mathematically defined in Eq. (2). Since source terms computed during earlier explicit stages are not needed anymore, we employ a single variable. Source terms $r_{(\text{slow})}$ computed on a lower time level are defined before the routine is called. Now a case distinction has to be made, whether or not the forward step in time associated with the current Runge-Kutta stage $\Delta t\,(c_i - c_{i-1})$ is greater than zero.

```
12:       if c_i > c_{i-1} then
13:         if there are blocks on a higher level then
14:           substeps: = ⌈2(c_i − c_{i-1})⌉
15:           send weighted boundary fluxes to neighbours
                on higher level: r^{neighbour}_{(slow)} := r/(c_i − c_{i-1})
16:           for l = 1, ...,substeps do
17:             RFSMR(t_0 + Δt · c_{i-1}+
                (l − 1) · Δt · (c_i − c_{i-1})/substeps,
                Δt · (c_i − c_{i-1})/substeps,
                level+1)
18:           end for
19:         end if
20:         for all Processor-local blocks k do
21:           if [timelevel of Block k]= level then
22:             DiffReact(Y, r, t_0 + Δt · c_{i-1},
                Δt · (c_i − c_{i-1}))
23:             exchange boundary concentration with
                  neighbours on same level
24:           end if
25:         end for
26:       else if c_i ≤ c_{i-1} then
27:         for all Processor-local blocks k do
28:           if [timelevel of Block k]= level then
29:             Y = Y + Δt · r
30:             exchange boundary concentration with
                  neighbours on same level
31:             send boundary concentration to
                  neighbours on lower level
32:           end if
33:         end for
34:       end if
35:     end for ▷ Loop over Runge-Kutta stages
36:   end procedure
```

In this pseudo code we assume that DiffReact$(Y, r, \tau_0, \Delta\tau)$ solves the initial value problem given by

$$c(\tau_0) = Y,$$
$$\frac{d}{d\tau}c = r + F(c, \tau),$$
$$\tau \in [\tau_0, \tau_0 + \Delta\tau],$$

with F representing a time dependent diffusion-reaction term and stores the result $c(\tau_0 + \Delta\tau)$ in the variable Y.

The number of substeps to be taken on the next higher time level can be chosen dynamically in line 14. This allows us to use a different base method in the next level to ensure a time step ratio of 2 between successive time levels. For the method (RK2a) given in Table 2 two substeps will be employed between the first and second stage, while no step will be taken between the second stage and the summation stage; this is equivalent to the formal construction as shown in Table 1. For (RK2b) one step on the next temporal level will be done both for the second stage and the summation stage, again leading to a time step ratio of two. Generally this time step ratio is possible for any explicit Runge-Kutta method with monotonically increasing nodes c such that

$$\forall i : c_i \in \{0, 1/2, 1\}.$$

Table 2. Examples of two stage, second order explicit Runge-Kutta methods.

0		
1	1	
	1/2	1/2

(RK2a)

0		
1/2	1/2	
	0	1

(RK2b)

The above synopsis of the actual implementation already shows that due to the recursive calls the programme flow will naturally be structured into multiple phases corresponding to the different time levels. This fact significantly complicates balancing. Furthermore, there are different kinds of exchanges which shall be discussed in the following section.

3.3 Data exchange

In the above listed pseudo code, the following data exchanges are mentioned. Exchanges of fluxes have to be performed between blocks on one level (line 8) or from a lower to a higher time level (line 15). Exchanges of concentrations have to be performed between blocks on a single level (line 23, 30) or from a higher to a lower level (line 31). While the expression "exchange of fluxes" is illustrative it is not exact. Stored and exchanged are not the advective fluxes across cell boundaries, but the time derivatives of the concentrations per cell computed from the net fluxes. If the boundary fluxes are cast as $f_{i\pm1/2,j}$ and $f_{i,j\pm1/2}$ with spatial indexes i, j and cell volume $V_{i,j}$, then the concentration tendency due to advection reads

$$\frac{d}{dt}c_{i,j} = -\frac{f_{i+1/2,j} - f_{i-1/2,j} + f_{i,j+1/2} - f_{i,j-1/2}}{V_{i,j}}.$$

Due to the data structure, i.e., the data available for a block and the algorithm employed for the calculation of block local fluxes, each boundary flux is computed exactly once and sent to the neighbouring block. It is convenient to store the received time derivatives in an additional variable allowing for a distinction between fluxes given by neighbours on the same and on a lower temporal refinement level.

Exchange of concentrations always overwrites the previous concentration of the receiving cell. While flux exchange always involves both the outermost actual cells and the halo cells, exchange of concentrations may occur in different ways. If concentrations are exchanged on a single time level, the exchange is a copy from the outermost actual cells of the sender to the halo cells of the receiver, possibly complicated by inter process communication. If on the other hand concentrations are sent to a block on a different time level concentration from both ghost cells and outermost actual cells are

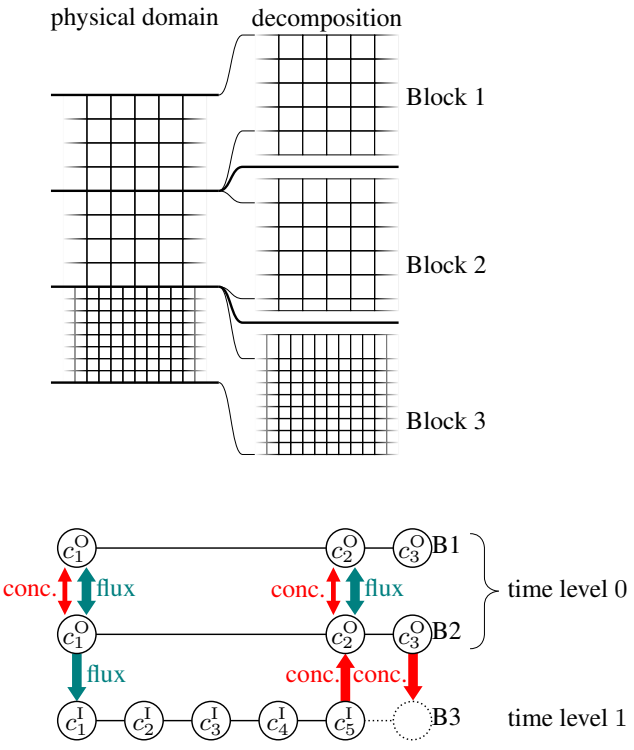

Fig. 3. Exchanges in course of a macro time step for method based on RK2a, see also Table 1. The c_i^{O} and c_i^{I} denote the nodes of the inner and outer base method.

updated from their geometrical counterparts. The rationale for this is that the halo cells of the sending block contain a better approximation than the receiving block's outer cells. An illustration of the exchanges in the course of one macro time step is given in Fig. 3. The last exchange is necessary to update the higher level block's outer cells and halo with the result of the correction stage performed on the macro time level.

Considering the geometrical aspects of the various exchanges, it is important to ensure that multiple representations of the same physical volume contain equivalent data. As all exchanged quantities are either cell average values or time derivatives of cell average values, this can be accomplished by averaging and constant interpolation of these quantities. We interpret the cell volumes $V \subset \Omega$ as subsets of the simulation domain Ω. Further we denote the original quantity and the original volume as given by the sending block with a superscript "S" and the copied quantity of the receiving block and the correlated volume with a superscript "R". The exchanges for a generic quantity q for different configurations then read:

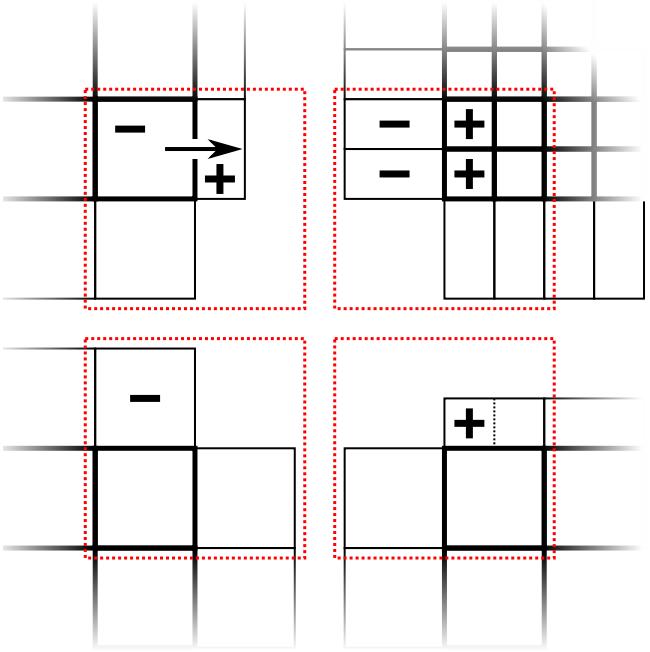

Fig. 4. Multiple representation of the same physical region (marked by red rectangles) and source/sink terms caused by a specific advective flux.

$$V^{\mathrm{S}} = V^{\mathrm{R}} \Rightarrow q^{\mathrm{R}} := q^{\mathrm{S}},$$
$$V_1^{\mathrm{S}} \cup V_2^{\mathrm{S}} = V^{\mathrm{R}} \Rightarrow q^{\mathrm{R}} := \frac{1}{2}\left(q_1^{\mathrm{S}} + q_2^{\mathrm{S}}\right),$$
$$V^{\mathrm{S}} = V_1^{\mathrm{R}} \cup V_2^{\mathrm{R}} \Rightarrow q_1^{\mathrm{R}} = q_2^{\mathrm{R}} := q^{\mathrm{S}},$$

corresponding to the cell relations (a), (b) and (c) as illustrated in Fig. 2. More complicated relations hold for diagonal exchanges. For a better understanding see Fig. 4. A flux across a specific cell boundary is computed from data present in the upper left block. This flux in turn is represented by a source term marked "+" and a sink term marked "–". As the same physical region is represented 4 times, these source and sink terms have to be exchanged to the other blocks shown. The exchange is complicated by the fact that the source term influences "half a ghost cell" of the lower right block. For this specific configuration, this problem can be solved by adding half of the source term to the ghost cell. If as in our implementation the spatial resolution of directly (i.e., not diagonally) adjacent blocks is allowed to differ by a factor of two maximum, nine different configurations of diagonal overlaps have to be distinguished. Note that diagonal exchanges are obsolete if all blocks have the same temporal refinement level, i.e., a classical time integration scheme is employed.

For efficient parallel execution it is desirable to minimise communication cost. Generally it is more efficient to exchange one big chunk of data instead of several small chunks of equal cumulative size. For this reason inter process

exchange is implemented as a gathering or packing of data, exchange using MPI routines and finally unpacking of data. As meta data regarding all blocks including adjacency information is present on all processors every participant of an exchange knows a priori which data to send and/or to receive. To reduce the amount of data to be exchanged interpolation and averaging is done in such a way that as little data as possible is to be transferred. This means that averaging is done before packing while interpolation is done after unpacking.

3.4 Balancing

If a simulation is to be distributed on multiple cores of one processor, multiple processors or even multiple nodes of a computing cluster, the simulation has to be split in several parts. In the context of air pollution modelling this means a decomposition of the simulation domain into blocks. The available computing elements are then to be assigned to these blocks such that idle times are minimised. For classical time integration schemes, this can easily be implemented by performing *workload balancing*. Thus, it can be provided that every processor has approximately the same amount of work to do.

The Metis/ParMetis libraries (Karypis et al., 2011) provide sophisticated balancing algorithms. Balancing problems are interpreted as a class of graph theoretical problems which may be subsumed as "minimise the edge cut without violating node balancing constraints". Blocks of the decomposed simulation domain are mapped to nodes of the graph, necessary data exchanges are mapped to the edges connecting these nodes. Consequently minimising the edge cut is equivalent to minimising the communication between partitions or processors.

The more complex programme flow in the multirate context calls for a more sophisticated approach to balancing. Naive workload balancing is not sufficient to minimise idle times, as the programme flow is structured in multiple phases due to recursive calls, see Fig. 5 for an example. Assuming that the same computational cost is associated with all four mentioned blocks, both presented block distributions are optimal in the sense of workload balancing. However, the worst case distribution will lead to an unnecessarily high amount of idle times.

Optimally not only the overall workload is balanced, but also the workload for each phase, i.e., for each temporal refinement level. The Metis/ParMetis libraries offer the possibility of *multi constraint balancing*, (Karypis, 1999). Classical balancing (single constraint) considers one scalar weight per node and aims at an even distribution of this scalar weight within a margin of tolerance. Opposed to this multi-constraint balancing considers a vector of weights for each node and aims at an even distribution for each vector dimension. For our algorithm that naively means that the weight vector $\boldsymbol{w}$ for a block reads

Setup:

- Blocks #1 and #2 on temporal refinement level 0
- Blocks #3 and #4 on temporal refinement level 1.
- Equal workload per block.

a) Worst case block distribution:

time level	0		1		2,...	1		0	
processor 1	block 1	block 2	[idle]		...	[idle]		block 1	block 2
processor 2	[idle]		block 3	block 4	...	block 3	block 4	[idle]	

b) Best case block distribution:

time level	0	1	2,...	1	0
processor 1	block 1	block 3	...	block 3	block 1
processor 2	block 2	block 4	...	block 4	block 2

Fig. 5. Parallel programme flow for different distributions of four blocks on two processors. Thick lines indicate exchanges.

$$\boldsymbol{w} \in \mathbb{R}^N, \quad w_k = \begin{cases} C \text{ if } k = L \\ 0 \text{ otherwise} \end{cases},$$

with N denoting the number of time levels throughout the simulation, L the block's time level and C the number of columns within the block. Choosing this approach will probably not lead to satisfactory results as the constraints leave only little margin for optimisation. As a compromise we employ three constraints correlated to the highest temporal refinement level $L_{\max}$, the second highest temporal refinement level and the remaining levels. The rationale for this is that in most setups the two highest refinement levels will cause the bigger part of computational cost. Consequently balancing blocks on these levels will lead to an acceptable tradeoff between constraints and idle times. We define the three dimensional weight vector as follows:

$$\boldsymbol{w} = \begin{cases} (C, 0, 0) & \text{if } L = L_{\max} \\ (0, C, 0) & \text{if } L = L_{\max} - 1 \\ \left(0, 0, 2^L C\right) & \text{otherwise} \end{cases}.$$

The factor 2^L in the third case is needed to consider the relative computational cost of blocks on potentially different time levels. The scaling for the first two components of the weighting vector is not necessary, as only their relative weights are taken into account by the ParMetis library. To prioritise balancing of the highest temporal refinement levels over the remaining levels, a smaller and, thus, stricter margin of tolerance is provided for the first component of the weight vector.

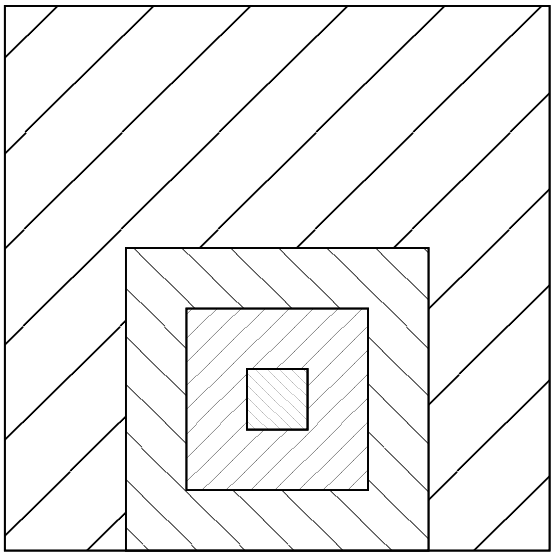

Fig. 6. Illustration of the spatial structure for academic test run. Hatchings correspond to different grid sizes.

Table 3. Synopsis of spatial structure for academic test case.

horizontal cell size	relative area	number of columns	fraction of cells total
4 km	≈69 %	3584	≈18 %
2 km	≈20 %	4096	≈21 %
1 km	≈10 %	8192	≈41 %
500 m	≈1 %	4096	≈21 %

Computational tests with realistic scenarios show ambivalent results. While multi-constraint balancing is suitable to minimise idle times during computation, it generally leads to higher communication cost, as the optimisation of communication is hindered by more constraints. Thus, it is generally recommendable for such simulations in which local computations take significantly more time than data exchange, e.g., simulations involving computationally expensive chemistry or microphysics. If in contrast to that a simulation is communication dominated, relatively little parallelisation speedup can be expected even for optimal distribution of blocks on different processors.

4 Results

The implementation described above was tested with academic and realistic scenarios. Results show good agreement of the solutions obtained with the multirate splitting and classical time integration. We shall present two test cases. The first test is designed to make optimal use of the multirate approach by employing a grid with a small region of interest and a homogeneous, diagonal wind field. The other test case

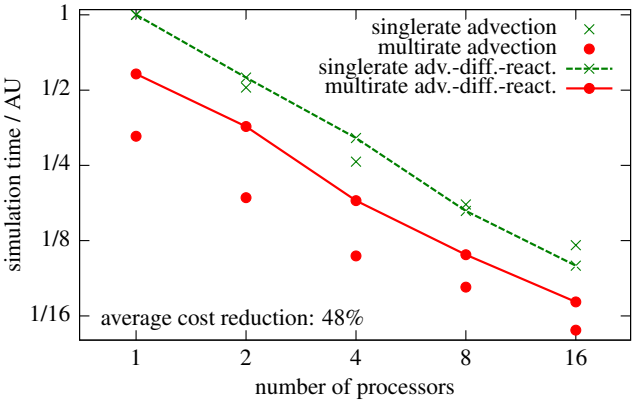

Fig. 7. Computational cost for academic test case. The displayed simulation time was normalised such that the single-rate setup on one processor corresponds to unit. Actual computational cost of the pure advection setup is about 1 % of the full setup.

is taken from an earlier study, with a realistic wind field provided by the COSMO model (Hinneburg et al., 2009). Results of the latter case shall demonstrate the potential of multirate schemes for realistic scenarios as well as show remaining deficiencies.

4.1 Academic test case

An important characteristic of parallel programmes is the speedup[1] when solving the problem on multiple processors. For the scenario discussed here we observed not quite an ideal (i.e., linear) speedup, but the overhead is small enough to justify parallel execution. Furthermore, due to a sophisticated balancing approach making use of ParMetis' multi-constraint partitioning capabilities, the parallelisation speedup is comparable to the one obtained for the much simpler case of single-rate time integration.

The domain is quadratic in horizontal direction. A comparatively small region is refined, see Table 3 and Fig. 6. Sources are located within the region most finely resolved.

In this domain, we tested two kinds of model equations: a pure advection with a uniform wind field and advection-diffusion-reaction with the same wind field, vertical diffusion and a chemistry model involving point sources in the finest region and 258 different chemical reactions of 98 reactants. The overall computational cost of the full system is about a factor of 100 larger than that of the pure advection case. These tests are run on different numbers of processors each. We compare the computational cost of the multirate approach to the computational cost without temporal refinement, denoted *single-rate*. Results are shown in Fig. 7.

As the time step is chosen to be directly proportional to the grid size, the naive cost reduction is approximately 52 %.

[1]Not to be confused with the cost reduction due to application of a multirate scheme.

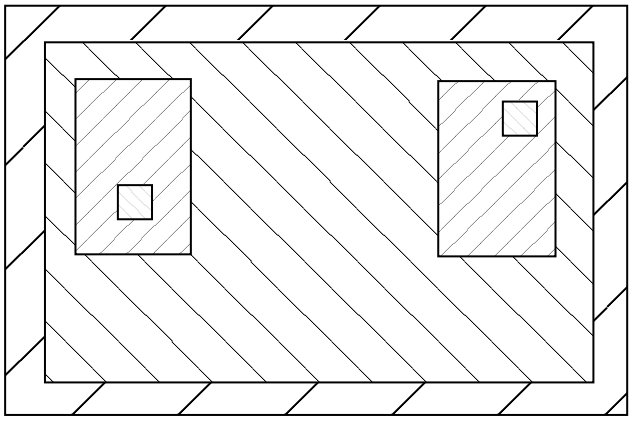

Fig. 8. Illustration of the spatial structure for realistic test run. Hatchings correspond to different grid sizes.

For a pure advection problem, we achieve an average multirate cost reduction of 59 %. Here the exceeding of the naive cost reduction can be explained by reduced communication. The behaviour for the full advection-diffusion reaction system, however, is not intuitive. In the latter case, the multirate approach is applied only to the advection operator whose evaluation contributes about 1 % to the total computational cost. However, there still is a significant cost reduction of about 36 %. The reason for this is the behaviour of the term solved implicitly depending on the source term r, see Eq. (4). Each update of this source term introduces a discontinuity in the right-hand side of the equation. In combination with the error control of the second order implicit solver this causes smaller implicit steps or even makes expensive restarts necessary (Knoth and Wolke, 1998a). Fewer updates take place if the outer system is solved using a larger time step, thus, indirectly improving the efficiency of the implicit solving.

4.2 Realistic test case

The following test case is taken from a earlier study performed by Hinneburg et al. (2009), examining the effects of emissions of two power plants in Germany. One plant is located near Lippendorf, the other near Boxberg, both in the federal country Saxony. Emissions are modelled by point sources which for the larger part represent the chimneys of the power plants, and area sources representing the emissions according to land usage. Meteorological data is provided by the COSMO model via online coupling.

Again we employ a grid with four levels of refinement with small, highly resolved regions around the power plants, totalling about 1 % of the overall area, see Table 4 and Fig. 8. The high resolution in this context is chosen to ensure an accurate description of the near field chemistry around the power plants by reducing numerical diffusion. Exactly equal parts of the total area are on the coarsest and second coarsest

Table 4. Synopsis of spatial structure for realistic test case.

horizontal cell size	relative area	number of columns	fraction of cells total
2.8 km	≈41.0 %	1748	≈7.3 %
1.4 km	≈41.0 %	6992	≈29.2 %
700 m	≈16.6 %	11312	≈47.2 %
350 m	≈1.4 %	3904	≈16.3 %

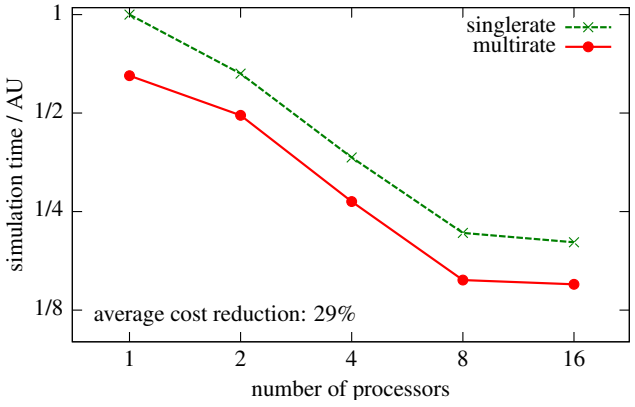

Fig. 9. Computational cost for realistic test case. The displayed simulation time was normalised such that the single-rate setup on one processor corresponds to unit.

refinement level. Chemical reactions are modelled as in the more complex of the academic test cases with 258 different chemical reactions of 98 reactants.

Assuming a homogeneous wind field we would expect a slightly higher reduction of computational cost as for the academic test case with equal diffusion-reaction setup, as a smaller fraction of cells is on the finest refinement level. The actually obtained cost reduction is lower, see also Fig. 9. This results from the fact that the wind fields provided by the COSMO model exhibit strong variability in all of the computational domain. A more sophisticated time step selection based on the characteristic times of the individual blocks rather than based solely on the spatial resolution can be constructed with relative ease. Complications arise for parallel execution – if the temporal refinement level of a block is changed, a redistribution of the blocks is necessary. This holds for either of the balancing approaches discussed in Sect. 3.4.

A further defect concerns the parallelisation speedup: while the problem scales well for up to eight processors, employing sixteen processors yields only very little improvement. At least in part this results from inhomogeneities due to the distribution of the point sources. The strongest point sources are inhomogeneously distributed on the blocks with the highest temporal and spatial refinement level. Each of the point sources induces significantly increased cost for the

chemistry solver. For up to eight processors, the blocks containing the point sources are distributed evenly on all processors; for more processors this is not the case. This problem is even more complicated due to the temporally varying wind field. Due to higher concentrations the speed of chemical reactions inside of the plume is significantly higher than in free air. If the plume is transported into a previously empty cell, computational cost of this cell is increased for the next time step. This effect can not easily be taken into consideration a priori. However, employing dynamic repartitioning based on the measured workload per block seems to be a promising way to tackle this problem.

Since the correction of both of the mentioned defects involves the implementation of a complex repartitioning routine, it seems recommendable to implement a conjunctive solution.

5 Conclusions

In this paper, we presented details on an efficient implementation of a general splitting approach. This approach is employed to obtain a multirate-IMEX splitting, i.e., advection for different physical domains is solved with different explicit time steps depending on the grid size while diffusion-reaction equations are solved implicitly. We have shown that the presented implementation is efficient in the sense that a good fraction of the theoretical speedup can be obtained practically. As practical tests have shown even the efficiency of the implicit solving is improved due to synergetic effects. A reasonable parallel speedup can be achieved by employing the multi-constraint balancing approach as described in Sect. 3.4.

Further work should include a more sophisticated selection of the temporal refinement level, as opposed to choosing the temporal refinement level equal to the spatial refinement level. Additionally, the system could be improved by implementation of dynamic repartitioning.

Acknowledgements. This research is funded by the German research foundation (Deutsche Forschungsgemeinschaft – DFG). Furthermore, we thank the DWD Offenbach and the NIC Jülich for supporting the work.

Edited by: A. Sandu

References

Hairer, E., Norsett, S., and Wanner, G.: Solving ordinary differential equations, vol. I, Springer Verlag, 1987.

Hinneburg, D., Renner, E., and Wolke, R.: Formation of secondary inorganic aerosols by power plant emissions exhausted through cooling towers in Saxony, Environ. Sci. Pollut. Res., 16, 25–35, 2009.

Hundsdorfer, W. and Verwer, J.: Numerical solution of time-dependent advection-diffusion reaction equations, Springer Series in Computational Mathematics, Springer, Berlin, Heidelberg, New York, 2003.

Jackiewicz, Z. and Vermiglio, R.: Order conditions for partitioned Runge–Kutta methods, Appl. Math., 45, 301–316, 1998.

Karypis, G.: Multilevel algorithms for multi-constraint hypergraph partitioning, Technical Report 99–034, University of Minnesota, Department of Computer Science/Army HPC Research Center, 1999.

Karypis, G., Schloegel, K., and Kumar, V.: ParMetis – Parallel graph partitioning and sparse matrix ordering library, Version 3.1, available at: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview, 2011.

Knoth, O. and Wolke, R.: Implicit-explicit Runge-Kutta methods for computing atmospheric reactive flows, Appl. Numer. Math., 28, 327–341, 1998a.

Knoth, O. and Wolke, R.: An explicit-implicit numerical approach for atmospheric chemistry-transport modelling, Atmos. Environ., 32, 1785–1797, 1998b.

Osher, S. and Sanders, R.: Numerical approximations to nonlinear conservation laws with locally varying time and space grids, Math. Comput., 41, 321–336, 1983.

Schlegel, M., Knoth, O., Arnold, M., and Wolke, R.: Multirate Runge–Kutta schemes for advection equations, J. Comput. Appl. Math., 226, 345–357, 2009.

Schlegel, M., Knoth, O., Wolke, R., and Arnold, M.: Numerical solution of multiscale problems in atmospheric modelling, Appl. Numer. Math., 62, 1531–1543, 2012.

Schättler, U., Doms, G., and Schraff, C.: A description of the nonhydrostatic regional COSMO model, Part I: Dynamics and numerics, Technical Report, Deutscher Wetterdienst, Offenbach, available at: http://www.cosmo-model.org, 2011.

Steppeler, J., Doms, G., Schättler, U., Bitzer, H., Gassmann, A., Damrath, U., and Gregoric, G.: Meso-gamma scale forecasts using the nonhydrostatic model LM, Meteorol. Atmos. Phys., 107, 75–96, 2003.

Tang, H. and Warnecke, G.: A class of high resolution schemes for hyperbolic conservation laws and convection – diffusion equations with varying time and space grids, SIAM J. Sci. Comput., 26, 1415–1431, 2005.

Verwer, J. G.: Explicit Runge–Kutta methods for parabolic partial differential equations, Appl. Numer. Math., 22, 359–379, 1996.

Verwer, J. G., Hundsdorfer, W., and Blom, J. G.: Numerical time integration for air pollution models, Surv. Math. Indust., 10, 107–174, 2002.

Verwer, J. G., Sommeijer, B. P., and Hundsdorfer, W.: RKC time-stepping for advection–diffusion–reaction problems, J. Comput. Phys., 201, 61–79, 2004.

Wolke, R. and Knoth, O.: Implicit-explicit Runge-Kutta methods applied to atmospheric chemistry-transport modelling, Environ. Modell. Softw., 15, 711–719, 2000.

Wolke, R., Knoth, O., Hellmuth, O., Schröder, W., and Renner, E.: The parallel model system LM-MUSCAT for chemistry-transport simulations: Coupling scheme, parallelization and applications, in: Parallel Computing: Software Technology, Algorithms, Architectures, and Applications, edited by: Joubert, G. R., Nagel, W. E., Peters, F. J., and Walter, W. V., Elsevier, 363–370, 2004.