Geosci. Model Dev. Discuss., 4, 2937–2972, 2011
www.geosci-model-dev-discuss.net/4/2937/2011/
doi:10.5194/gmdd-4-2937-2011
© Author(s) 2011. CC Attribution 3.0 License.

Geoscientific
Model Development
Discussions

Implementation of splitting methods for air pollution modeling

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

Received: 11 July 2011 – Accepted: 18 September 2011 – Published: 15 November 2011

Correspondence to: M. Schlegel (schlegel@tropos.de)

Published by Copernicus Publications on behalf of the European Geosciences Union.

Abstract

Explicit time integration methods are characterized by a small numerical effort per time step. In the application to multiscale problems in atmospheric modeling, 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 program 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 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 this sub 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 as e.g. advection on locally refined grids. Generally multirate schemes allow for a more significant reduction of computational cost the fewer components require the smallest time step.

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, 2011) 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 program flow, details on data exchange and a balancing approach in separate sub sections. 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). For a better understanding we shortly outline this approach here. 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.

Defining the nodes of the explicit method $\tau_i = t_0 + \Delta t c_i$ and requiring the c_i to be monotonically increasing with i, the algorithm computing an approximate solution w_1 for time t_1 from an approximate solution w_0 at time t_0 can be cast as follows

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

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

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

$$\frac{dv_i}{d\tau} = \frac{1}{c_i - c_{i-1}} r_i + F\left(v_i\right), \tag{4}$$

$$\tau \in \left[\tau_{i-1}, \tau_i\right] \;, \quad i = 2, \ldots, s+1 \;,$$

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

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

with s denoting the number of stages of an explicit Runge-Kutta (ERK) method with parameters (a,b,c) in common 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 be 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

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

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

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

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

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

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

with the tilde parameters denoting the increments per Runge–Kutta 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 macro time step to the 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 $(\Delta t(c_2^{\mathrm{O}} - c_1^{\mathrm{O}}), \Delta t(c_3^{\mathrm{O}} - c_2^{\mathrm{O}}), \ldots)$. 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. (2011).

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. Even third order accuracy can be obtained if the base methods themselves satisfy third order conditions and one 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

$b^{\text{fast}} \neq b^{\text{slow}}$ (see Table 1 for an example), so that the methods do not preserve the linear invariants of the system. 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 that 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 correlated 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 discretization 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., 2003; Karypis, 1999). The MUSCAT code is mainly written in FORTRAN90/95 with few additional C libraries.

This section is organized as follows: first we shall explain how data is organized in our model, then present the program flow of local computations and finally explain data exchange strategies. As workload balancing grows more complicated with the more complex program flow we shall also embark on this issue.

3.1 Data organization and spatial structure

In MUSCAT data is organized hierarchically: the three dimensional computational domain is decomposed statically into rectangular blocks. This decomposition is applied in 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 multiplied with 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 initialization and meteorological data as wind speed or density of air. The latter may be provided by the COSMO model, (Schättler et al., 2008; 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 organized so that cells within one column (i.e. cells at the same horizontal position) have minimal address distance. This layout is advantageous for the computations executed most frequently, i.e. cell local chemistry and column local (vertical) diffusion. For each advection step a number of fully coupled implicit chemistry-diffusion steps is needed which contribute significantly to the overall computational cost. All arrays have the same structure making vectorized 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. Additional data structures for the exchange of mass fluxes with neighbor blocks are needed along the respective boundaries, see Fig. 1. These data structures correspond to the source term r occurring in Eq. (4) and (11). Employing a limited third order upwind spatial discretization (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 extend of the actual cells they overlap with, see Fig. 2. Also marked in Fig. 2 are the possible relations between actual cells and ghost cells representing the same physical volume. 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 neighbor,

c) one inner cell and two ghost cells of a finer neighbor or

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

Case d) only occurs at block corners and may be complicated for neighbor 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 Program flow

In this section we shall discuss the MUSCAT program flow. For simplicity we shall concentrate on the main integration loop, omitting initialization, finalization 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.

```
1:  procedure RFSMR ($t_0, \Delta t$, level)
2:    for $i = 2, \dots, s+1$ ▹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:          $r := \sum_{j=1}^{i-1} (a_{ij} - a_{i-1,j}) f_j$
7:          $r := r + r_{(\mathrm{slow})} \cdot (c_i - c_{i-1})$
8:          exchange fluxes with neighbor blocks
            on same level $r_{(\mathrm{equal})} := r^{\mathrm{neighbor}}$
9:          $r := r + r_{(\mathrm{equal})}$
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 "neighbor" if the variable belongs to another block. The source term r (see line 7) correspond to the source term r_i as mathematically defined in Eq. (2). As source terms of previous explicit stages are not needed anymore we employ a single variable for this. 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 neighbors
               on higher level: $r^{\text{neighbor}}_{\text{(slow)}} := r/(c_i - c_{i-1})$
16:           for $l = 1,\ldots,$substeps do
17:             RFSMR($t_0 + \Delta t \cdot c_{i-1} +$
                $(l-1) \cdot \Delta t \cdot (c_i - c_{i-1})/substeps$,
                $\Delta t \cdot (c_i - c_{i-1})/substeps$,
                $level + 1$)
18:           end for
19:         end if
20:         for all Processor-local blocks $k$ do
21:           if [timelevel of Block $k$]= $level$ then
22:             DiffReact($Y, r, t_0 + \Delta t \cdot c_{i-1}$,
                $\Delta t \cdot (c_i - c_{i-1})$)
23:             exchange boundary concentration with neighbors on same level
24:           end if
25:         end for
26:       else if $c_i \leq c_{i-1}$ then
27:         for all Processor-local blocks $k$ do
28:           if [timelevel of Block $k$]= $level$ then
29:             $Y = Y + \Delta t \cdot r$
30:             exchange boundary concentration with neighbors on same level
31:             send boundary concentration to neighbors on lower level
32:           end if
33:         end for
34:       end if
35:     end for ▷ Loop over Runge-Kutta stages
36:   end procedure
```

GMDD
4, 2937–2972, 2011
Implementation of splitting methods
M. Schlegel et al.

Title Page
Abstract Introduction
Conclusions References
Tables Figures
◀◀ ▶▶
◀ ▶
Back Close
Full Screen / Esc
Printer-friendly Version
Interactive Discussion

In this pseudo code we assume that DIFFREACT(Y,r,τ_0,$\Delta\tau$) solves the initial value problem given by

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

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

Choosing the number of sub steps to be taken by on the next higher time level dynamically in line 14 allows for a different base method to be used while aiming at a time step ratio of 2 between successive time levels. For method (RK2a) given in Table 2 two sub steps 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

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

The above synopsis of the actual implementation already shows that due to the recursive calls the program 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 pseudo code listed above the following data exchanges are mentioned. Exchanges of fluxes between blocks on one level (line 8) or from a lower to a higher time level (line 15) and exchanges of concentrations 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. Thus an exchange of fluxes always is incremental. It is convenient to store the received time derivatives in an additional variable allowing for a distinction between fluxes given by neighbors 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 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 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:

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

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. There 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 minimize 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 modeling 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 minimized. 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., 2003) provide sophisticated balancing algorithms. Balancing problems are interpreted as a class of graph theoretical problems which may be subsumed as "minimize the edge cut without violating node balancing constraints". Blocks of the decomposed simulation domain are mapped to nodes a the graph, necessary data exchanges are mapped to the edges connecting these nodes. Consequently minimizing the edge cut is equivalent to minimizing the communication between partitions or processors.

The more complex program flow in the multirate context calls for a more sophisticated approach to balancing. Naive workload balancing is not sufficient to minimize idle times, as the program 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 w for a block reads

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

with N denoting the number of time levels throughout the simulation, the block's time level L and the number of columns C within the block. Choosing this approach will probably not lead to satisfying results as the constraints leave only little margin for optimization. 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 few constraints and little idle times. We define the three dimensional weight vector as follows:

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

with L_{max} denoting the maximum temporal refinement level. The factor 2^L in the third case is needed to consider the relative computational cost of blocks on potentially different time levels. It is not needed for the first two components of the vector, as their absolute weights are not taken into account by the ParMetis library. To prioritize 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.

Computational tests with realistic scenarios show ambivalent results. While multi constraint balancing is suitable to minimize idle times during computation it generally leads to higher communication cost, as the optimization of communication is hindered by more constraints. Thus it generally is 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 parallelization 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 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 programs is the speedup[1] when distributing 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 parallelization speedup is comparable to the one obtained for the much simpler case of singlerate 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 *singlerate*. Results are shown in Fig. 7.

As the time step is chosen to be directly proportional to the grid size, the naive speedup is approximately 52%. For a pure advection problem we achieve an average multirate speedup of 59%. Here the exceeding of the naive speedup can be explained by reduced communication. The behavior 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 speedup of about 36 %. The reason for this is the behavior 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

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

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 modeled 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 refinement level. Chemical reactions are modeled 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 multirate speedup 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 speedup 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 parallelization 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. Employing a multi constraint balancing approach the parallel speedup is acceptable.

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 work is funded by the German research foundation (Deutsche Forschungsgesellschaft – DFG).

References

Hairer, E., Norsett, S., and Wanner, G.: Solving ordinary differential equations, vol. I, Springer Verlag, 1987. 2942

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. 2939, 2954, 2956

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. 2945

Jackiewicz, Z. and Vermiglio, R.: Order conditions for partitioned Runge–Kutta methods, Appl. Math., 45, 301–316, 1998. 2942

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. 2944, 2953

Karypis, G., Schloegel, K., and Kumar, V.: ParMetis - Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.1, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview, 2003. 2944, 2952

Knoth, O. and Wolke, R.: Implicit-explicit Runge-Kutta methods for computing atmospheric reactive flows, Appl. Numer. Math., 28, 327–341, doi:http://dx.doi.org/10.1016/S0168-9274(98)00051-8, 1998a. 2939, 2956

Knoth, O. and Wolke, R.: An Explicit-Implicit Numerical Approach for Atmospheric Chemistry-Transport Modeling, Atmos. Environ., 32, 1785–1797, 1998b. 2943

Osher, S. and Sanders, R.: Numerical approximations to nonlinear conservation laws with locally varying time and space grids, Math. Comput., 41, 321–336, 1983. 2939

Schlegel, M., Knoth, O., Arnold, M., and Wolke, R.: Multirate Runge–Kutta schemes for advection equations, JCAM, 226, 345–357, 2009. 2939

Schlegel, M., Knoth, O., Wolke, R., and Arnold, M.: Numerical Solution of Multiscale Problems in Atmospheric Modeling, accepted, Proc. NUMDIFF-12, 2009, 2011. 2939, 2942

Schättler, U., Doms, G., and Schraff, C.: A description of the nonhydrostatic regional COSMO-Model. Part I: Dynamics and numerics, Tech. rep., Deutscher Wetterdienst, Offenbach, available from http://www.cosmo-model.org, 2008. 2944

Steppeler, J., Doms, G., Schttler, 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, doi:10.1007/s00703-001-0592-9, 2003. 2944

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. 2939

Verwer, J. G.: Explicit Runge–Kutta methods for parabolic partial differential equations, Appl. Numer. Math., 22, 359–379, 1996. 2938

Verwer, J. G., Sommeijer, B. P., and Hundsdorfer, W.: RKC time-stepping for advection–diffusion–reaction problems, J. Comput. Phys., 201, 61–79, 2004. 2938

Wolke, R. and Knoth, O.: Implicit-explicit Runge-Kutta methods applied to atmospheric chemistry-transport modelling., Environ. Modell. Softw., 711–719, 2000. 2943

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, 363–370, Elsevier, 2004. 2939, 2944

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

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

(RK2a)

outer base method

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 1/4 & 1/4 & & \\ 1 & 1/4 & 1/4 & 1/2 & \\ \hline & 1/4 & 1/4 & 1/4 & 1/4 \end{array}$$

inner base method

$$\begin{array}{c|ccccc} 0 & & & & & \\ 1/2 & 1/2 & & & & \\ 1/2 & 1/2 & 0 & & & \\ 1 & 1 & 0 & 0 & & \\ 1 & 1 & 0 & 0 & 0 & \\ \hline & 1/2 & 0 & 0 & 0 & 1/2 \end{array}$$

“slow” part

$$\begin{array}{c|ccccc} 0 & & & & & \\ 1/2 & 1/2 & & & & \\ 1/2 & 1/4 & 1/4 & & & \\ 1 & 1/4 & 1/4 & 1/2 & & \\ 1 & 1/4 & 1/4 & 1/4 & 1/4 & \\ \hline & 1/4 & 1/4 & 1/4 & 1/4 & 0 \end{array}$$

“fast” part

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

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array} \qquad \begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array}$$

(RK2a) (RK2b)

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 %

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%	11 312	≈47.2 %
350 m	≈1.4%	3904	≈16.3 %

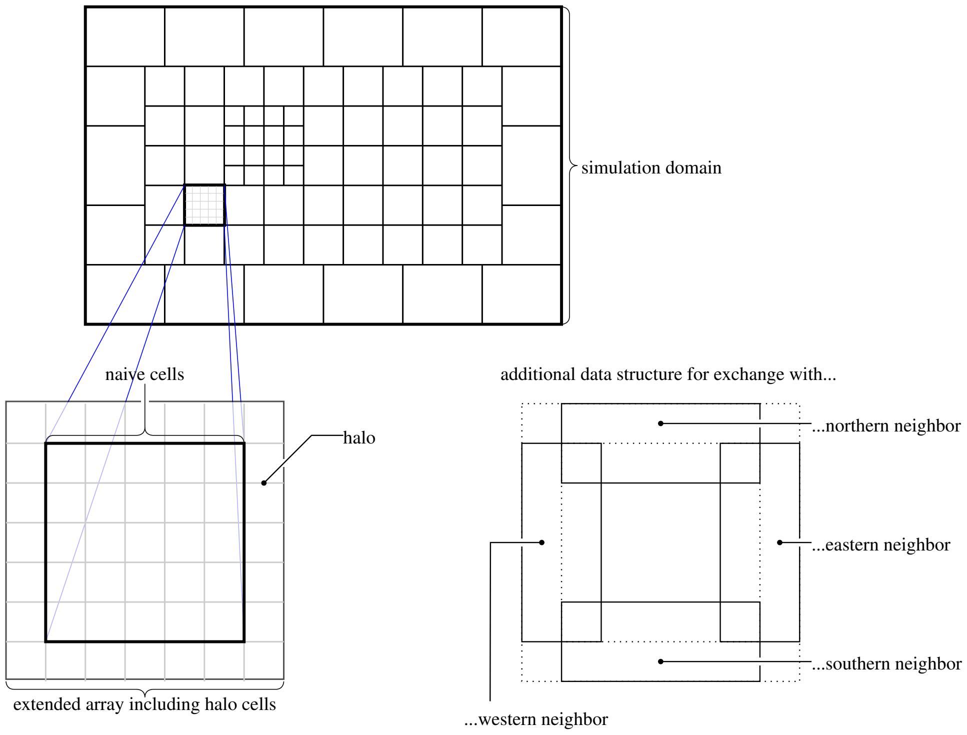

Fig. 1. Illustration of block structure for simulation domain as well as one generic block in MUSCAT.

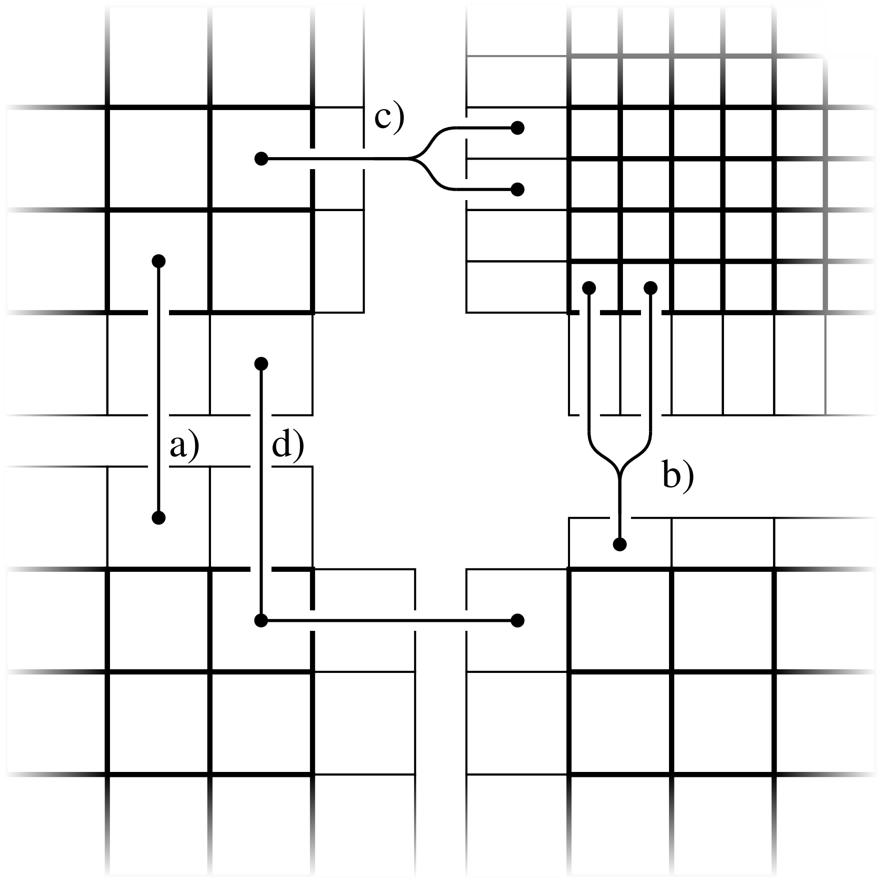

Fig. 2. Geometrical cell structure of adjacent blocks. Halo cells depicted with thinner contours. Connectors between blocks indicate multiple representations of the same physical volume.

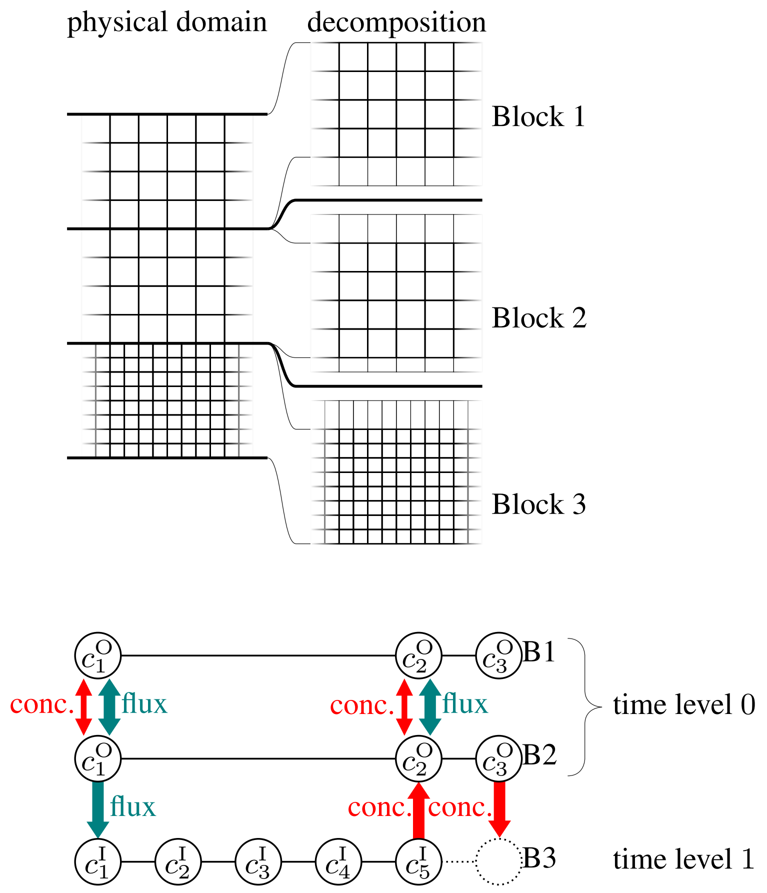

Fig. 3. Exchanges in course of a macro time step for method based on RK2a, see also Table 1. The c_i^O and c_i^I denote the nodes of the inner and outer base method.

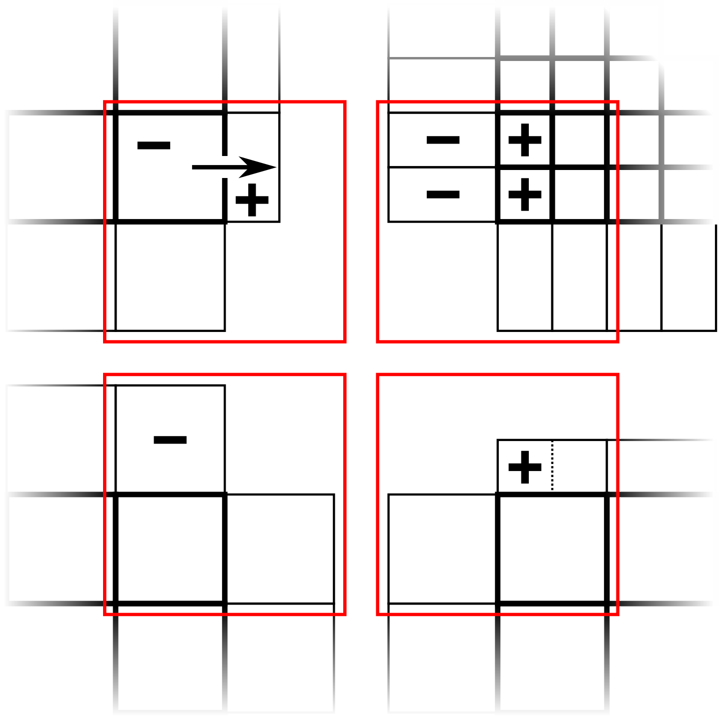

Fig. 4. Multiple representation of the same physical region (marked by red rectangles) and source/sink terms caused by a specific advective flux.

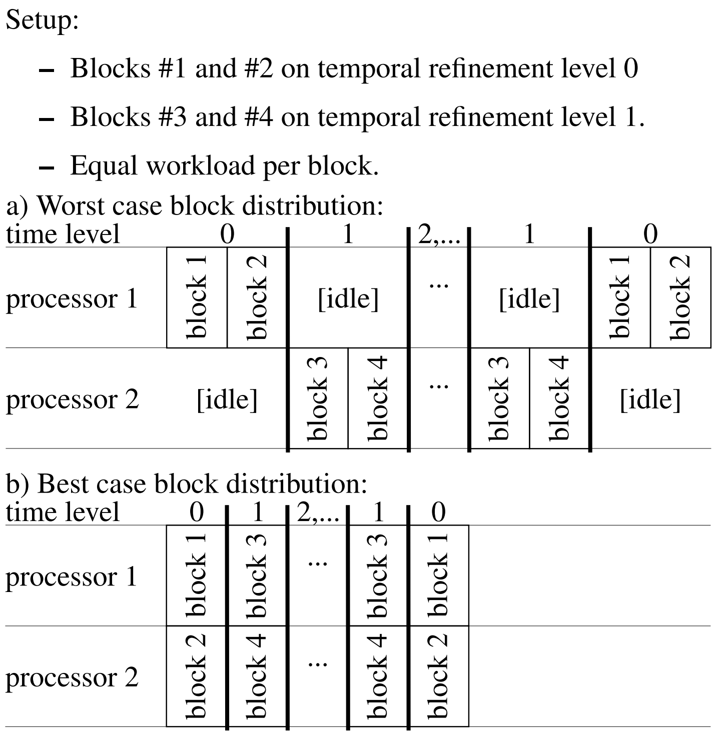

Fig. 5. Parallel program flow for different distributions of four blocks on two processors. Thick lines indicate exchanges.

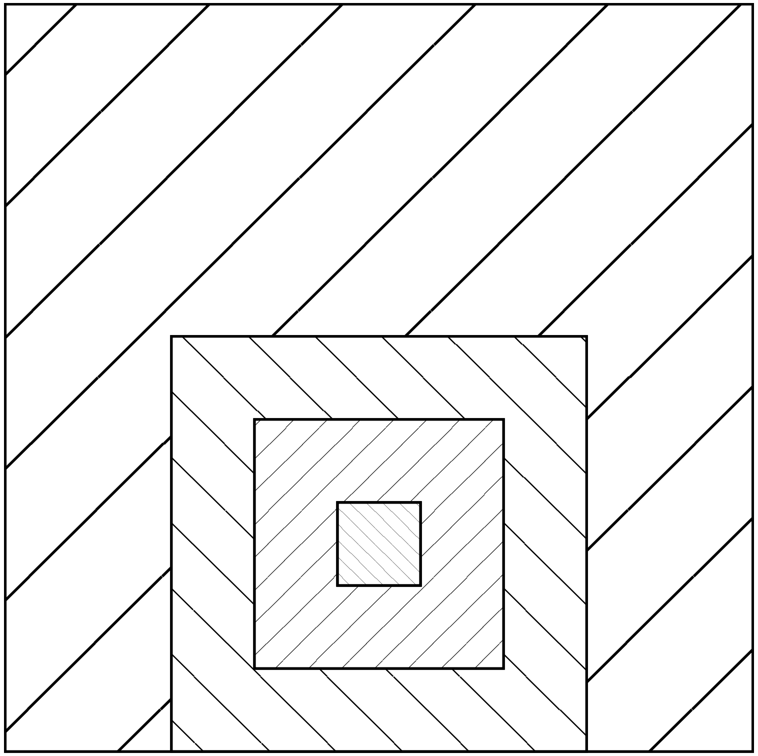

Fig. 6. Illustration of the spatial structure for academic test run. Hatchings correspond to different grid sizes.

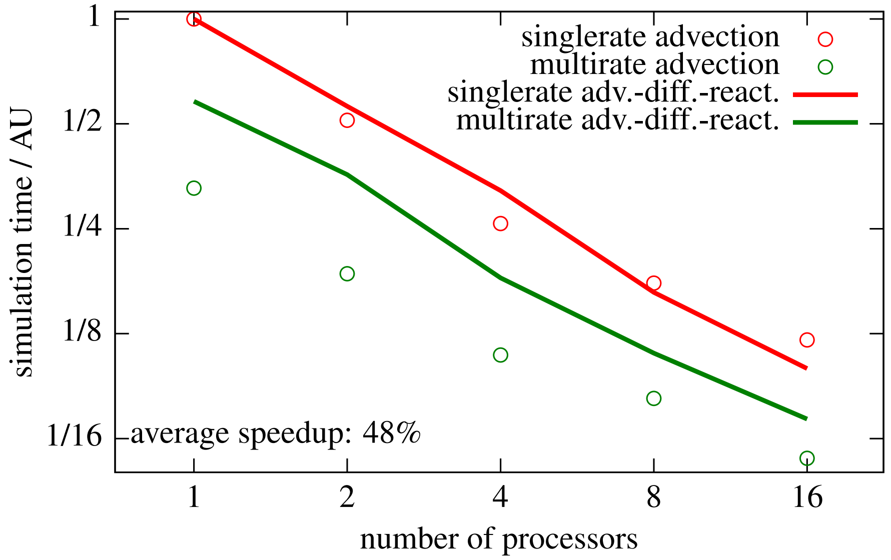

Fig. 7. Computational cost for academic test case. The displayed simulation time was normalized such that the singlerate setup on one processor corresponds to unit. Actual computational cost of the pure advection setup is about 1 % of the full setup.

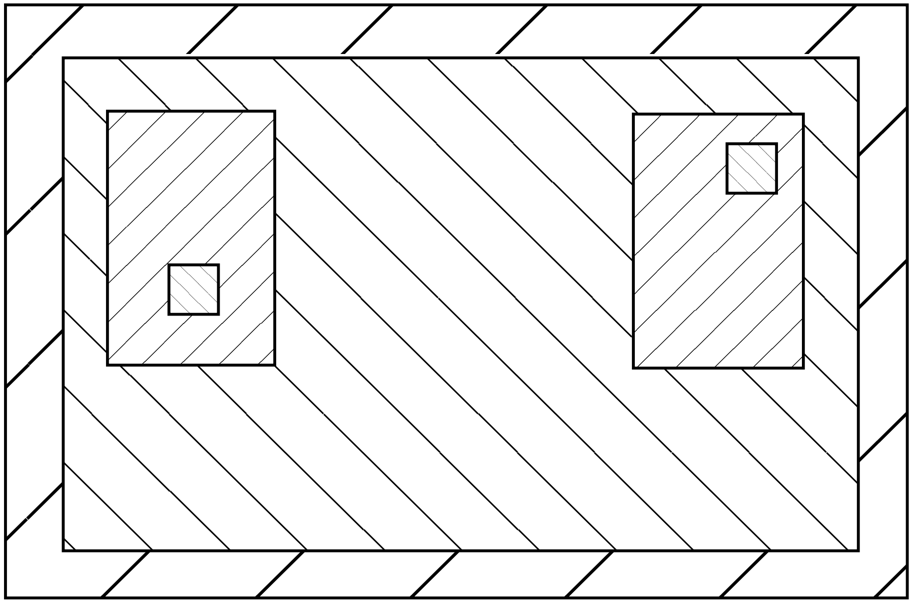

Fig. 8. Illustration of the spatial structure for realistic test run. Hatchings correspond to different grid sizes.

GMDD

4, 2937–2972, 2011

Implementation of splitting methods

M. Schlegel et al.

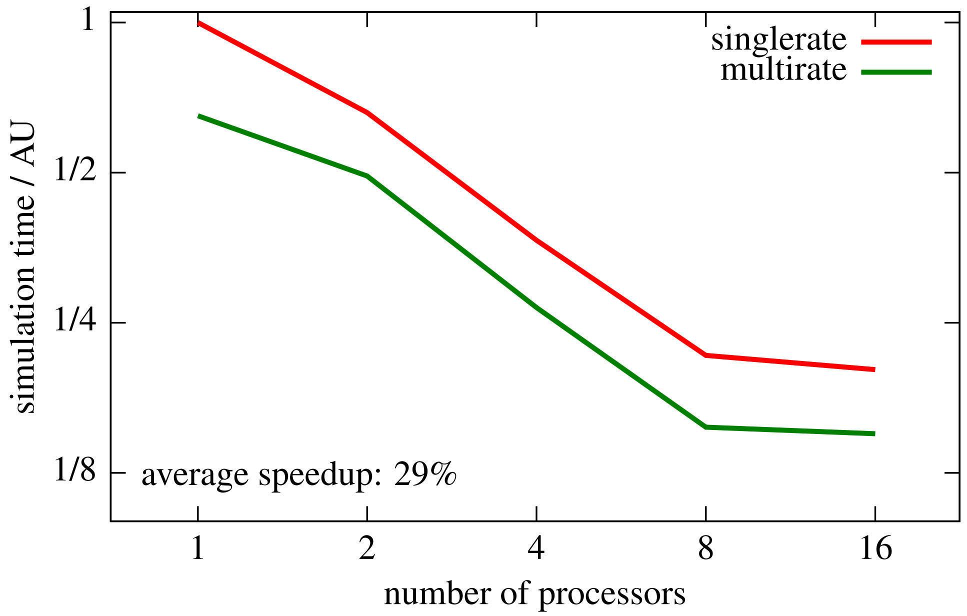

Fig. 9. Computational cost for realistic test case. The displayed simulation time was normalized such that the singlerate setup on one processor corresponds to unit.