ELSEVIER

Applied Numerical Mathematics 21 (1996) 303–320

APPLIED NUMERICAL MATHEMATICS

Splitting methods for three-dimensional bio-chemical transport ☆

B.P. Sommeijer *, J. Kok

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

Splitting methods for the time integration of three-dimensional transport-chemistry models offer interesting prospects: second-order accuracy can be combined with sufficient stability, and the amount of implicitness can be reduced to a manageable level. Furthermore, exploiting the parallelization and vectorization features of the algorithm, a realistic simulation with many species over long time intervals becomes feasible. As an alternative to the usual splitting functions, such as co-ordinate splitting or operator splitting, we discuss in this paper a splitting function that is of hopscotch type. Both for a second-order, symmetric spatial discretization (resulting in a three-point coupling in each direction), and for a third-order, upwind discretization (giving rise to a five-point coupling, in general), we define a particular variant of this hopscotch splitting. These splitting *functions* will be combined with an appropriate splitting *formula*, resulting in second-order (in time) splitting *methods*. A common feature of both hopscotch splitting functions is that we have only coupling in the vertical direction, resulting in a stability behaviour that is independent of the vertical mesh size; this is an important property for transport in *shallow* water. Another characteristic of this hopscotch-type splitting is that it allows for an easy application of domain decomposition techniques in the horizontal directions. Two choices for the splitting formula will be presented. The resulting methods have been applied to a large-scale test problem and the numerical results will be discussed. Furthermore, we show performance results obtained on a Cray C98/4256. As part of the project TRUST (Transport and Reactions Unified by Splitting Techniques), preliminary versions of the schemes are available for benchmarking.

Keywords: Transport models; Shallow water; 3D; Splitting methods; Stability; HPCN

1. Introduction

The mathematical model describing transport processes of salinity, pollutants, etc. in water, combined with their bio-chemical interactions, is defined by an initial-boundary value problem for the system of 3D advection–diffusion–reaction equations

☆ The investigations reported in this paper were supported by the NCF (National Computing Facility Foundation) who provided the authors with a grant (CRG 95-08) from the Cray Research University Grants Program.

* Corresponding author. E-mail: bsom@cwi.nl.

PII S0168-9274(96)00015-3

$$\frac{\partial c_i}{\partial t} + \frac{\partial}{\partial x}(uc_i) + \frac{\partial}{\partial y}(vc_i) + \frac{\partial}{\partial z}(wc_i) = \frac{\partial}{\partial x}\left(\varepsilon_x \frac{\partial c_i}{\partial x}\right) + \frac{\partial}{\partial y}\left(\varepsilon_y \frac{\partial c_i}{\partial y}\right) + \frac{\partial}{\partial z}\left(\varepsilon_z \frac{\partial c_i}{\partial z}\right) + g_i(t, x, y, z, c_1, \ldots, c_m), \quad i = 1, \ldots, m, \tag{1.1}$$

where c_i denote the unknown concentrations of the contaminants. The local fluid velocities u, v, w (to be provided by a hydrodynamical model) and the diffusion coefficients ε_x, ε_y, ε_z are assumed to be given functions. The equations in (1.1) are mutually coupled by means of the functions g_i, which model the (concentration-dependent) bio-chemical reactions and emissions from sources. The definition of the physical domain and of the initial and boundary conditions completes the model. Following the Method of Lines approach, Eq. (1.1), together with the initial condition and the boundary conditions is converted into the semidiscrete initial value problem

$$\frac{\mathrm{d}\boldsymbol{C}(t)}{\mathrm{d}t} = \boldsymbol{F}\big(t, \boldsymbol{C}(t)\big) := \boldsymbol{H}\big(t, \boldsymbol{C}(t)\big) + \boldsymbol{G}\big(t, \boldsymbol{C}(t)\big), \quad \boldsymbol{C}(t_0) = \boldsymbol{C}_0. \tag{1.2}$$

Here, $\boldsymbol{C}$ is a vector of dimension mN containing the m concentrations c_i at the total number of $N := N_x N_y N_z$ grid points (N_x, N_y and N_z denote the number of grid points in the various spatial directions, respectively). The term $\boldsymbol{H}(t, \boldsymbol{C}(t))$ originates from the discretization of the advection–diffusion terms (including the boundary conditions), and $\boldsymbol{G}(t, \boldsymbol{C}(t))$ is the discrete analogue of the reaction terms and emissions. Finally, $\boldsymbol{C}_0$ contains the initial values.

Since the functions $\boldsymbol{H}$ and $\boldsymbol{G}$ have quite a different origin, they give rise to a completely different coupling of the unknowns: in $\boldsymbol{H}$ the concentrations of the various species are uncoupled, but there is of course a coupling in space, due to the underlying spatial differential operators. In $\boldsymbol{G}$ on the other hand, we have in each grid point a local coupling of the concentrations. Another observation is that $\boldsymbol{H}$ is linear in $\boldsymbol{C}$, whereas $\boldsymbol{G}$ is usually nonlinear.

These observations should be taken into account in selecting a suitable time integration method. In this context, "suitable" means that the method should have the following properties:

(i) *Sufficient stability*: In the present application, we are primarily concerned with transport in *shallow* seas, resulting in small values for the mesh size in the vertical direction. As a consequence, stiffness is introduced into the discrete system (1.2). This observation excludes the use of fully explicit methods, since the stability requirements would force the method to take unrealistically small time steps. One possibility to avoid these stability problems is to select a fully implicit method. However, the different nature of $\boldsymbol{H}$ and $\boldsymbol{G}$, as well as the fact that we are dealing with *three* spatial dimensions, result in a complicated coupling in the right-hand side function $\boldsymbol{F}$, and hence in the corresponding Jacobian. As a result, the linear algebra problem to solve the implicit relations is extremely large. This aspect leads us to the second requirement:

(ii) *Manageable level of computational effort*: Based on the above observation, we strive for a reduction of the amount of implicitness, while maintaining sufficient stability. Especially, the coupling in the systems that have to be solved should be modest. For this item it is also relevant to mention that good vectorization and parallelization properties are indispensable to reduce the computational effort.

(iii) *Realistic accuracy*: In this PDE context, high precision results (e.g., produced by high order methods) are usually not necessary. On the other hand, since predictions over long time periods are an essential part in this kind of simulations, first-order accuracy is, in our opinion, too low. Therefore, we restrict our attention to methods that are second-order in time.

(iv) *Storage economy*: Although present-day computers are equipped with large memories, the nature of flow problems, especially in *three* dimensions, still necessitates a careful selection of an algorithm with respect to its storage requirements. A situation in which we are dealing with $N = 10^6$ grid points and $m = 10$ or 20 species is certainly not unusual.

(v) *Domain decomposition*: In many practical situations, different resolutions in space will be required in various regions of the domain. For example, near the coasts and in estuaries a fine grid is unavoidable to capture the physical phenomena. A natural way to efficiently cope with this demand is to apply a domain decomposition approach, in which the various subdomains are discretized with an appropriate resolution. Then the (sub)problems on the various subdomains can be solved in parallel. However, to obtain an efficient process for the overall solution, the coupling of these subproblems should not be too tight, since in that case many iterations would be necessary to match the interface conditions on the boundaries of these subdomains. Therefore, we are aiming at methods that are "loosely coupled in the horizontal direction".

The requirements (i) and (ii) lead us to choose a *splitting method*, which is partly explicit and partly implicit. In this way, we can combine the computational simplicity of an explicit method and the sufficient stability properties of an implicit method. Such splitting methods consist of a splitting *function* and a splitting *formula*.

For the splitting formula we will only consider second-order accuracy, as motivated at item (iii) above. As we will see later, the choice of the splitting formula will also depend on the particular splitting function that we use.

Well-known choices for the splitting function are based on "co-ordinate" (or, dimensional) splitting and operator splitting. In co-ordinate splitting, we create a strong coupling in the horizontal (direction), which conflicts with our requirement (v). By operator splitting we mean that the advection terms and the diffusion terms are treated separately. Since at least one of these terms needs an implicit treatment to satisfy (i), the resulting Jacobian matrix possesses an unpleasant structure, due to the three-dimensional coupling; this would prevent to satisfy condition (ii).

Therefore, we consider in this paper an alternative splitting function, based on the so-called hopscotch type splitting, which corresponds to a special partitioning of the spatial grid points. The basis of this hopscotch idea goes back to Gourlay [5]. An important characteristic of the particular hopscotch type splitting that we have used is that it gives rise to coupling in the vertical direction only. This is a useful property in shallow water flow problems, since the stiffness in (1.2) is mainly introduced by the discretization in the vertical. The partitioning of the points in each horizontal plane depends on how the differential operators with respect to x and y have been discretized: if second-order, central differences are used (resulting in a three-point coupling in both directions), then it suffices to divide the horizontal grid points into two subsets. However, a more sophisticated third-order, upwind-biased discretization needs a partitioning into three subsets. Moreover, as we will see, such a splitting function imposes an extra condition on the splitting formula. Combining both types of hopscotch splitting with appropriate splitting formulae results in splitting methods which were termed Odd–Even Line Hopscotch (OELH) method and Red–Black–White Line Hopscotch (RBWLH) method, respectively.

In [10,11] the numerical treatment of (1.1) has been investigated for the case of a single transport equation (i.e., $m = 1$), using the OELH method. In [12] this method was successfully applied to

a five-species model ($m = 5$). A theoretical stability analysis can be found in [14]. As a result of these studies we may conclude that the OELH method is sufficiently stable to let the time step be dictated by accuracy considerations and not by the stability condition; furthermore, it allows for a very efficient implementation on (multi)vector processors (exploiting the vectorization capabilities of the Cray C98/4256 resulted in a speed-up factor of about 12 with respect to scalar mode).

The reason to consider upwind discretizations as well, is that unwanted wiggles in the numerical solution could be generated if symmetric discretizations are used for the advection terms. Such wiggles can be suppressed to a large extent by using a third-order upwind discretization. This, however, generally results in a five-point coupling in each direction. As a consequence of this extended stencil, we can no longer apply the odd–even line hopscotch ordering and we have to introduce three groups of grid points (say, red, black and white points). Given this splitting function, the time integration requires a splitting formula which allows for multiterm splitting.

In [7] a survey is given of splitting formulae possessing this property. Some of these formulae originate from the literature and a few newly constructed ones are presented as well. For the verification of the time discretization order of these methods, a general framework has been set up in [7]. Starting with a very general class of Runge–Kutta type methods using fractional stages (called RKS methods), the order conditions are easily derived. This approach allows to verify the order of any (one-step) splitting method, however complicated the method may be.

The remainder of the paper is organized as follows. The next section briefly discusses the RKS framework. Both the OELH method and the RBWLH method will be formulated in this format. In Section 3, we will discuss the computational aspects of these methods, taking into account their possibilities for vectorization and parallelization. On the basis of a test example, extensive numerical experiments are presented in Section 4, and, finally, some conclusions are formulated in Section 5.

2. Runge–Kutta splitting methods

In [7] a general framework has been set up to define splitting formulae. The starting point was a Runge–Kutta (RK) type formula.

Based on the multiterm splitting of the right-hand side function $\boldsymbol{F}$ in (1.2) according to

$$\boldsymbol{F}(t, \boldsymbol{C}) = \sum_{k=1}^{\sigma} \boldsymbol{f}_k(t, \boldsymbol{C}), \tag{2.1}$$

the RKS formula studied in [7] is of the general form

$$\begin{aligned} &\boldsymbol{Y}_0 = \boldsymbol{C}_n, \\ &\boldsymbol{Y}_i = \boldsymbol{Y}_{i-1} + \Delta t \sum_{k=1}^{\sigma} \sum_{j=0}^{i} a_{ij}^{(k)} \boldsymbol{f}_k(t_n + \mu_j \Delta t, \boldsymbol{Y}_j), \quad i = 1, \ldots, s, \\ &\boldsymbol{C}_{n+1} = \boldsymbol{Y}_s. \end{aligned} \tag{2.2}$$

Here, s intermediate approximations $\boldsymbol{Y}_i$ have been introduced, which is a typical approach in RK methods. Furthermore, $\boldsymbol{C}_n$ and $\boldsymbol{C}_{n+1}$ represent approximations to the exact solution vector $\boldsymbol{C}(t)$ at

$t = t_n$ and $t = t_{n+1}$, and Δt is the integration step. The parameters $a_{ij}^{(k)}$ and μ_j are free and can be used to give the formula the required numerical properties, such as stability and order of consistency.

The order conditions are easy to verify using the relations derived in [7] (see also [6]). For the stability we refer to [14], where, for one particular choice (viz. the OELH method), the stability is analyzed in detail.

Notice that $\sigma = 1$ yields a conventional (non-splitted) diagonally implicit RK method. The method {(2.1), (2.2)} will be called an RKS method and is completely defined by the splitting function (2.1) and the parameters $a_{ij}^{(k)}$ and μ_j defining the splitting formula. In the next two subsections we will discuss specific choices, leading to the OELH method and the RBWLH method, respectively.

2.1. RKS method based on symmetric spatial discretization

In this subsection we consider the case that the spatial differential operators are replaced by the symmetric, second-order difference stencils

$$u\frac{\partial}{\partial x} \approx \frac{u}{2\Delta x}[-1,\ 0,\ 1], \qquad \frac{\partial^2}{\partial x^2} \approx \frac{1}{(\Delta x)^2}[1,\ -2,\ 1], \tag{2.3}$$

and similar expressions for the derivatives with respect to y and z. Notice that we do not need to discretize the terms $c_i(\partial u/\partial x + \partial v/\partial y + \partial w/\partial z)$ in (1.1) since the velocity field is assumed to be divergence free. As mentioned in the Introduction, symmetric discretization of the advection terms may easily lead to unwanted wiggles and therefore (2.3) should only be used in the case that the solution possesses low spatial activity.

From (2.3) we observe that we have a three-point coupling in each spatial direction and that there is no cross-coupling along the co-ordinate directions. Based on this observation, the grid points in each *horizontal* plane are divided into two categories, let us say the "o-points" and the "+-points", as indicated in Fig. 1.

Notice that each *vertical* grid line contains either "o-points" or "+-points". Furthermore, let the right-hand side function $\boldsymbol{H}$ in (1.2) be split according to

$$\boldsymbol{H}(t, \boldsymbol{C}) := \boldsymbol{H}^{+}(t, \boldsymbol{C}) + \boldsymbol{H}^{\mathrm{O}}(t, \boldsymbol{C}), \tag{2.4a}$$

where $\boldsymbol{H}^{+}$ and $\boldsymbol{H}^{\mathrm{O}}$ have only nonzero values at the grid points + and o, respectively. Then, we may define the Odd–Even Line Hopscotch (OELH) splitting by

$$\boldsymbol{f}_1 := \boldsymbol{H}^{+}, \qquad \boldsymbol{f}_2 := \boldsymbol{H}^{\mathrm{O}}. \tag{2.4b}$$

To handle the reaction and source terms represented by $\boldsymbol{G}$, we simply set

$$\boldsymbol{f}_3 := \boldsymbol{G}. \tag{2.4c}$$

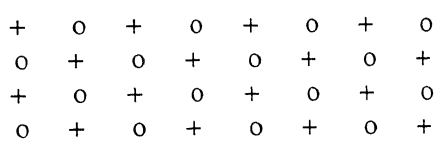

Fig. 1. Two categories of grid points.

Now, the OELH splitting method is defined by

$$\begin{aligned}
&\boldsymbol{Y}_0 = \boldsymbol{C}_n,\\
&\boldsymbol{Y}_1 = \boldsymbol{Y}_0 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{\mathrm{O}}(t_n, \boldsymbol{Y}_0) + \boldsymbol{H}^{+}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_1\big)\big],\\
&\boldsymbol{Y}_2 = \boldsymbol{Y}_1 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{+}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_1\big) + \boldsymbol{H}^{\mathrm{O}}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_2\big)\big],\\
&\boldsymbol{Y}_3 = \boldsymbol{Y}_2 + \tfrac{1}{2}\Delta t\big[\boldsymbol{G}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_2\big) + \boldsymbol{G}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_3\big)\big],\\
&\boldsymbol{Y}_4 = \boldsymbol{Y}_3 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{\mathrm{O}}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_3\big) + \boldsymbol{H}^{+}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_4\big)\big],\\
&\boldsymbol{Y}_5 = \boldsymbol{Y}_4 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{+}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_4\big) + \boldsymbol{H}^{\mathrm{O}}(t_n + \Delta t, \boldsymbol{Y}_5)\big],\\
&\boldsymbol{C}_{n+1} = \boldsymbol{Y}_5.
\end{aligned} \tag{2.5}$$

This scheme can be verified to be second-order indeed. Here we remark that this formulation of the OELH method slightly differs from the one given in [12]. In that paper the advection–diffusion part of the equation (i.e., the $\boldsymbol{H}$-function) was integrated with the OELH method over a full time step; that is, using only the stages for $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, with $\Delta t/4$ replaced by $\Delta t/2$. The resulting output was then used as input for an explicit RK method to integrate the chemical part of the equation (the $\boldsymbol{G}$-part). This so-called "fractional step" approach is, at least formally, only first-order accurate. Following an idea of Strang [13], a combination of two of such steps in a reversed order results in a method of second-order, which is very similar to (2.5). In Section 3, this scheme will be discussed in more detail. First we proceed with specifying the scheme based on upwind discretization.

2.2. *RKS method based on upwind discretization of the convection terms*

As pointed out earlier, wiggles (and resulting negative numerical values for the concentrations) can be largely suppressed by an upwind-biased discretization of the advection terms. Here, we use the so-called $\kappa = 1/3$ discretization [8], defined by

$$u\frac{\partial}{\partial x} \approx \begin{cases} \dfrac{u}{6\Delta x}[1, \ -6, \ \ 3, \ \ 2, \ \ 0], & \text{if } u \geqslant 0,\\[2ex] \dfrac{u}{6\Delta x}[0, \ -2, \ -3, \ \ 6, \ -1], & \text{if } u < 0, \end{cases} \tag{2.6}$$

which is a third-order accurate approximation. For the discretization of the diffusion terms we use the same stencil as given in the preceding subsection.

Again, we have a coupling in each co-ordinate direction only, but now the coupling is extended to five points in general. As a consequence, it is necessary to partition the grid points into three subsets, let us say into "o-points", "+-points", and "*-points". For each *horizontal* plane, these points are positioned as given in Fig. 2. Again, points on one and the same *vertical* line have the same mark.

Splitting the right-hand side function $\boldsymbol{H}$ in (1.2) as

$$\boldsymbol{H}(t, \boldsymbol{C}) := \boldsymbol{H}^{*}(t, \boldsymbol{C}) + \boldsymbol{H}^{+}(t, \boldsymbol{C}) + \boldsymbol{H}^{\mathrm{O}}(t, \boldsymbol{C}), \tag{2.7a}$$

with $\boldsymbol{H}^{*}$, $\boldsymbol{H}^{+}$ and $\boldsymbol{H}^{\mathrm{O}}$ having only nonzero values at the grid points $*$, $+$ and o, respectively, the Red–Black–White Line Hopscotch (RBWLH) splitting is defined by

$$\boldsymbol{f}_1 := \boldsymbol{H}^{*}, \qquad \boldsymbol{f}_2 := \boldsymbol{H}^{+}, \qquad \boldsymbol{f}_3 := \boldsymbol{H}^{\mathrm{O}}, \qquad \boldsymbol{f}_4 := \boldsymbol{G}. \tag{2.7b}$$

o * + o * + o *
* + o * + o * +
+ o * + o * + o
o * + o * + o *

Fig. 2. Three categories of grid points.

The RBWLH method that turned out to be the most promising (see [7]) reads

$$
\begin{aligned}
&\boldsymbol{Y}_0 = \boldsymbol{C}_n,\\
&\boldsymbol{Y}_1 = \boldsymbol{Y}_0 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{\mathrm{O}}(t_n, \boldsymbol{Y}_0) + \boldsymbol{H}^{*}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_1\big)\big],\\
&\boldsymbol{Y}_2 = \boldsymbol{Y}_1 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{*}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_1\big) + \boldsymbol{H}^{+}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_2\big)\big],\\
&\boldsymbol{Y}_3 = \boldsymbol{Y}_2 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{+}\big(t_n + \tfrac{1}{4}\Delta t, \boldsymbol{Y}_2\big) + \boldsymbol{H}^{\mathrm{O}}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_3\big)\big],\\
&\boldsymbol{Y}_4 = \boldsymbol{Y}_3 + \tfrac{1}{2}\Delta t\big[\boldsymbol{G}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_3\big) + \boldsymbol{G}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_4\big)\big],\\
&\boldsymbol{Y}_5 = \boldsymbol{Y}_4 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{\mathrm{O}}\big(t_n + \tfrac{1}{2}\Delta t, \boldsymbol{Y}_4\big) + \boldsymbol{H}^{+}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_5\big)\big],\\
&\boldsymbol{Y}_6 = \boldsymbol{Y}_5 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{+}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_5\big) + \boldsymbol{H}^{*}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_6\big)\big],\\
&\boldsymbol{Y}_7 = \boldsymbol{Y}_6 + \tfrac{1}{4}\Delta t\big[\boldsymbol{H}^{*}\big(t_n + \tfrac{3}{4}\Delta t, \boldsymbol{Y}_6\big) + \boldsymbol{H}^{\mathrm{O}}(t_n + \Delta t, \boldsymbol{Y}_7)\big],\\
&\boldsymbol{C}_{n+1} = \boldsymbol{Y}_7.
\end{aligned}
\tag{2.8}
$$

This method satisfies the second-order conditions given in [7].

3. Computational aspects

In this section we will discuss the computational aspects of the methods defined above. Here, we distinguish between algorithmic and implementational aspects, which will be discussed in the next subsections, respectively.

3.1. Algorithmic aspects

First, we observe that both methods have several features in common. For example, we see that in each stage the intermediate approximation $\boldsymbol{Y}_i$ appears implicitly in only one of the functions $\boldsymbol{H}^{*}$, $\boldsymbol{H}^{+}$, $\boldsymbol{H}^{\mathrm{O}}$ or $\boldsymbol{G}$; this results in a minimal amount of implicitness, which is in accordance with our aim (ii), as formulated in Introduction. Nevertheless, both schemes possess a sufficient stability behaviour. The OELH method (without chemical terms) has been analysed in detail [14], and it turned out that the relevant condition on the time step is of the form

$$\Delta t\left(\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y}\right) \leqslant 4. \tag{3.1}$$

Hence, we observe that the maximal stable stepsize is neither influenced by the terms in the vertical direction nor by the corresponding discretization parameter Δz. This is exactly what we want, for

reason formulated in property (i) in the Introduction. As a matter of fact, this property of hopscotch type splitting was the motivation for constructing this particular "explicit-in-the-horizontal-implicit-in-the-vertical-line-hopscotch-splitting". For the RBWLH method (2.8) a similar stability analysis is in progress; numerical experiments indicate that an analogue of (3.1) is obtained with the number 4 replaced by 2.7.

Owing to this minimal implicitness, the amount of work per stage is quite limited: let us, for example, discuss the first stage of the RBWLH method (2.8) in some detail. First of all, we recall that the concentrations of the various species composing $\boldsymbol{Y}_1$ are not coupled in the $\boldsymbol{H}$-functions, so that these concentrations can be computed in parallel. Furthermore, we observe that only the $*$-points in $\boldsymbol{Y}_1$ have to be calculated implicitly (in this stage, the +-points are simply copied from $\boldsymbol{Y}_0$, and the o-points are calculated explicitly). The total number of these $*$-points equals $\frac{1}{3}N_xN_yN_z$. The positioning of the points in each horizontal plane has been chosen in such a way that the upwind molecule (2.6) does not couple the horizontal $*$-points (see Fig. 2). There is, however, a coupling in the vertical direction, since the vertical grid lines contain points with the same mark. Hence, the work in the implicit part of the stage for $\boldsymbol{Y}_1$ falls apart in solving $\frac{1}{3}N_xN_y$ uncoupled, linear systems, each of which is of dimension N_z. These linear systems are, in general, of pentadiagonal form and allow for an efficient solution. This can be done either in parallel or in vector mode (see the next subsection for a discussion on this topic).

Obviously, the other stages in (2.8) involving $\boldsymbol{H}$-functions require the same amount of work. For the OELH method (2.6) the situation is similar; here, the number of uncoupled, (tridiagonal) linear systems to be solved per stage equals $\frac{1}{2}N_xN_y$ (having the same dimension N_z) for which an efficient solution process has been implemented (see Section 3.2).

A next observation is that some of the stages can be implemented in the so-called "fast-form", by which we mean that some of the explicit $\boldsymbol{H}$-evaluations can be avoided: let us consider the following consecutive stages

$$\boldsymbol{Y}_i = \boldsymbol{Y}_{i-1} + \text{"explicit } \boldsymbol{H}\text{-evaluation in non \#-points"} + \tfrac{1}{4}\Delta t\boldsymbol{H}^{\#}(t_n + \mu\Delta t, \boldsymbol{Y}_i),$$

$$\boldsymbol{Y}_{i+1} = \boldsymbol{Y}_i + \tfrac{1}{4}\Delta t\boldsymbol{H}^{\#}(t_n + \mu\Delta t, \boldsymbol{Y}_i) + \text{"implicit } \boldsymbol{H}\text{-evaluation in non \#-points"},$$

where μ equals $1/4$ or $3/4$, and \# stands for $*$ or + in (2.8) and for + in (2.5). Clearly, the \#-points in $\boldsymbol{Y}_{i+1}$ are defined by

$$\boldsymbol{Y}^{\#}_{i+1} = \boldsymbol{Y}^{\#}_i + \tfrac{1}{4}\Delta t\boldsymbol{H}^{\#}(t_n + \mu\Delta t, \boldsymbol{Y}_i) = \boldsymbol{Y}^{\#}_i + (\boldsymbol{Y}^{\#}_i - \boldsymbol{Y}^{\#}_{i-1}) = 2\boldsymbol{Y}^{\#}_i - \boldsymbol{Y}^{\#}_{i-1}.$$

Furthermore, in both methods this idea can also be used to calculate the o-points in the first stage of a step, using the relation in the last stage of the previous step.

Next, we discuss the "chemical stage", involving the $\boldsymbol{G}$-function. Here, we easily recognize the trapezoidal rule. Although this formula is implicit, the $\boldsymbol{Y}$-vector defined in this stage has been solved by a simple functional iteration process. This approach is motivated by the observation that the chemistry in water is usually quite slow (i.e., the Lipschitz constant of the function $\boldsymbol{G}$ is small). This iteration is continued until the residual (measured in the maximum norm) satisfies a prescribed tolerance. In order to maintain the second-order accuracy of the overall scheme, the residual should be proportional to the local truncation error of the trapezoidal rule. Therefore, in our experiments we have chosen the tolerance parameter to stop the functional iteration of the form $\delta(\Delta t)^3$, with δ sufficiently small.

We have seen that the coupling in the horizontal direction is very weak. This implies that the algorithm is suitable to be used within a Domain Decomposition context: the solution in one domain does not directly influence the solution in another domain, so that "matching" of the conditions on the mutual boundaries can be avoided. This was formulated as aim (v) in the Introduction and results will be reported in the near future.

Finally, we discuss a peculiar property which is inherent to the hopscotch splitting approach. Since the horizontal diffusion terms are treated in a way similar to the Du Fort–Frankel scheme (for parabolic problems), we have *conditional convergence* as $\Delta x, \Delta y \to 0$ and $\Delta t \to 0$. In fact, it can be shown that the global error of the hopscotch schemes is of the form

$$(\Delta t)^2\left(\frac{\varepsilon_x}{(\Delta x)^2}+\frac{\varepsilon_y}{(\Delta y)^2}\right)C+\mathrm{O}\big((\Delta t)^2\big)+\mathrm{O}\big(\Delta^q\big)_{\text{advection}}+\mathrm{O}\big(\Delta^2\big)_{\text{diffusion}}, \tag{3.2}$$

where $\Delta = \max(\Delta x, \Delta y, \Delta z)$, $q = 2$ for OELH and $q = 3$ for RBWLH, and C involves second derivatives (with respect to time) of the solution. For a more detailed discussion on this Du Fort–Frankel deficiency we refer to [14]. The consequence is that reducing all discretization parameters (in space and time) with the same factor, will not increase the resulting accuracy in the case that the first term in (3.2) dominates the global error. Therefore, the proper strategy in applying these methods is to choose the spatial grid as coarse as allowed by the bathymetry, the geometry, and the requirements with respect to resolution (i.e., spatial accuracy). Then, owing to the good stability properties and the second order behaviour in time, the time step can be selected in order to give roughly the same time discretization error.

3.2. *Solving the linear systems*

As indicated in the previous section, the implicit part in each stage requires (per concentration) the solution of $\alpha N_x N_y$ uncoupled linear systems, where $\alpha = 1/2$ for OELH and $\alpha = 1/3$ for RBWLH. In both codes these systems have been solved using a technique initially proposed by Golub and Van Loan [4]. The idea is to perform the successive steps in an LU-factorization (which are recursive, and hence prevent vectorization) for all systems simultaneously; since the systems are uncoupled, the resulting loops are perfectly vectorizable. Furthermore, because $\alpha N_x N_y$ is usually large, we obtain a vector speed close to peak performance. This so-called "vectorization-across-the-systems approach" for solving the tridiagonal systems occurring in OELH, has been extensively discussed in [11,12]. The same idea, however, can equally well be used to solve the pentadiagonal systems arising in RBWLH. We will now briefly comment on this extension.

In RBWLH we use an upwind-biased discretization for the vertical advection terms, yielding a coupling between four successive unknowns. However, depending on the sign of the vertical velocity component, the diagonal element may be the second or the third of these four non-zero elements. To obtain a *regular* pentadiagonal system, additional zero elements have been introduced at the "free" fifth position, as already indicated in (2.6). The costs of computing with these zeros in the co-diagonals are insignificant compared with the advantage of handling matrices with a constant bandwidth.

This "vectorization-across-the-systems" way of solving the linear systems has been implemented in the routines TRI3D (for the tridiagonal systems in OELH) and BAND5 (for the pentadiagonal systems in RBWLH). These routines do not need workspace, since the input arrays are overwritten

Table 1
CPU times (in seconds) and Mflop rates for TRI3D and BAND5

	TRI3D			BAND5		
Mode of calling	CPU time per call	Mflop rate	Speed-up w.r.t. scalar mode	CPU time per call	Mflop rate	Speed-up w.r.t. scalar mode
New Jacobian	$0.96 \cdot 10^{-3}$	610	22	$1.09 \cdot 10^{-3}$	628	20
Only new right-hand side	$0.46 \cdot 10^{-3}$	369	29	$0.45 \cdot 10^{-3}$	420	24

with the results of the decomposition (i.e., with the L and U matrix); similarly, the solution is delivered in the array providing the right-hand side vector. Apart from memory considerations, storing the L and U has an additional advantage: in many calls of TRI3D and BAND5, the coefficient matrix has not changed, so that only a forward/backward substitution is needed. This is especially lucrative if many concentrations are involved, since then the Jacobians for all discretized PDEs (with the same boundary conditions) are the same. Per system of dimension N_z, both routines require N_z divisions, and respectively $7N_z$ and $16N_z$ multiplications/additions in the LU-factorization part. Here we remark that the constants in front of N_z are larger than one might expect for a standard LU-factorization. This is due to the fact that it is the coefficient matrix $I - \frac{1}{4}\Delta t J$ which has to be decomposed when J has changed. To exploit the fact that the coefficient matrix is always of this specific form, its construction is incorporated in the decomposition algorithm. In the appendix, both algorithms are listed. For the subsequent forward/backward substitutions, they require $5N_z$ and $9N_z$ multiplications/additions, respectively. For both routines, Table 1 shows the CPU-times per call, and the Mflop rates (obtained on 1 processor and running in vector mode); here, we distinguish between the case that the Jacobian has been changed, and the case that a previous decomposition can be reused. These results correspond to a grid with $N_x = N_y = 81$ and $N_z = 11$. Notice that (per concentration and per step) 4 calls of TRI3D and 6 calls of BAND5 are needed. To demonstrate the excellent capabilities of these routines to exploit the vectorization facilities, we have included in Table 1 the speed-up factors that are obtained with respect to executing in scalar mode on the Cray C90.

Both routines have (among others) the input parameters `NXR` and `NY`, which can be used for an optimal tuning to the vector capabilities of the Cray, as well as to the availability of parallel processors. The parameters `NXR` and `NY` denote the length of columns and rows of the arrays containing the data corresponding to one horizontal plane of the grid. Optimal vector performance on *one processor* is obtained when the actual values in a call of TRI3D and BAND5 are set to $\alpha N_x N_y$ and 1, respectively. In this way, loops are collapsed, yielding optimal vector speed. For the *multi-processor* variant, the routines are called with the actual values αN_x and N_y for the parameters. In this case, N_y sets of αN_x systems will be distributed among the processors (with the best distribution when N_y is a multiple of the number of processors); per processor, the vectorized loops have length αN_x. In Table 2 we have listed performance results obtained on a grid with $N_x = N_y = 81$ and $N_z = 11$. From this table we observe that the vector speed, *per processor*, is reduced by a factor 1.7 (for OELH) and 2.7 (for BAND5) by changing from single- to multi-processor mode. Hence, for these routines, the number of parallel processors should be at least 2 and 3, respectively, to outweigh the loss of performance due to shorter loops. It should be remarked that the above numbers hold for the aforementioned grid, and that this reduction of vector speed is less pronounced if the grid is refined.

Table 2
Influence of the vector length due to parallelization

	TRI3D		BAND5	
	Length of vectorized loop	Mflop rate	Length of vectorized loop	Mflop rate
Single-processor mode	$\frac{1}{2}N_xN_y$	532	$\frac{1}{3}N_xN_y$	557
Multi-processor mode	$\frac{1}{2}N_x$	308	$\frac{1}{3}N_x$	208

Finally, we remark that we can refrain from using a pivot strategy (which would destroy the vectorization-across-the-systems approach). We have verified that in both cases the requirement on Δt to obtain diagonal dominance is much less stringent than the respective stability conditions on the time step (assuming realistic values for all parameters involved in these conditions).

4. Numerical experiments

The OELH and RBWLH methods described in the Sections 2.1 and 2.2 are applied to the test problem (see also [7])

$$\begin{aligned}\frac{\partial c_1}{\partial t} + \boldsymbol{U}\cdot\nabla c_1 &= \varepsilon\Delta c_1 + g_1(t,x,y,z) - k_1c_1c_2,\\ \frac{\partial c_2}{\partial t} + \boldsymbol{U}\cdot\nabla c_2 &= \varepsilon\Delta c_2 + g_2(t,x,y,z) - k_1c_1 + k_2(1-c_2),\end{aligned} \tag{4.1a}$$

defined on the region $0 \leqslant x, y \leqslant L_h$, $-L_v \leqslant z \leqslant 0$ and for $0 \leqslant t \leqslant T$. Here, $\boldsymbol{U} = (u, v, w)$ denotes the divergence free velocity field, given in analytical form (see [2])

$$\begin{aligned}u(t,x,y,z) &= \{\tilde{y} + 3(\tilde{z}+\tfrac{1}{2})[(\tilde{x}-\tfrac{1}{2})^2 + (\tilde{y}-\tfrac{1}{2})^2 - q^2]\}d(t),\\ v(t,x,y,z) &= \{-\tilde{x} + 3(\tilde{z}+\tfrac{1}{2})[(\tilde{x}-\tfrac{1}{2})^2 + (\tilde{y}-\tfrac{1}{2})^2 - q^2]\}d(t),\\ w(t,x,y,z) &= -3L_v\tilde{z}(\tilde{z}+1)\{(\tilde{x}-\tfrac{1}{2})/L_h + (\tilde{y}-\tfrac{1}{2})/L_h\}d(t),\end{aligned} \tag{4.1b}$$

where we used the scaled co-ordinates $\tilde{x} := x/L_h$, $\tilde{y} := y/L_h$, $\tilde{z} := z/L_v$; furthermore, $q = \frac{1}{3}$ and $d(t) = \cos(2\pi t/T_p)$. The Dirichlet boundary conditions, the initial condition and the functions g_1 and g_2 are chosen in accordance with the prescribed analytical solution, which is of the form

$$c_i(t,x,y,z) = \exp\{\tilde{z}/i - f_i(t) - \gamma_i[(\tilde{x}-r(t))^2 + (\tilde{y}-s(t))^2]\}, \quad i = 1,2, \tag{4.1c}$$

with $f_2(t) = t/(T_b+t)$, $f_1(t) = 4f_2(t)$, $r(t) = [2+\cos(2\pi t/T_p)]/4$, and $s(t) = [2+\sin(2\pi t/T_p)]/4$.

In our experiments, we take the following values for the parameters: $L_h = 20{,}000$, $L_v = 100$, $\varepsilon = 0.5$, $\gamma_1 = 80$, $\gamma_2 = 20$, $T_b = 32{,}400$, and $T_p = 43{,}200$. The length of the integration interval $T = 36\,000$. Realistic values for the reaction rate constants are: $k_1 = k_2 = 10^{-4}$.

The global accuracy of the fully discrete approximation is measured by

$$\mathrm{cd}_i := \text{minimum over all grid points } (-\log_{10}|\text{absolute error for } c_i|), \quad i = 1,2.$$

Hence, cd_i can be considered as the (minimal) number of correct digits for concentration c_i. In the experiments, we used three spatial grids, of increasing resolution:

$\text{Grid}_{\text{coarse}}$ is defined by: $N_x = N_y = 41$, $N_z = 6$, amounting to $\approx 6 \cdot 10^3$ internal grid points;
$\text{Grid}_{\text{middle}}$ is defined by: $N_x = N_y = 81$, $N_z = 11$, amounting to $\approx 5.6 \cdot 10^4$ internal grid points;
$\text{Grid}_{\text{fine}}$ is defined by: $N_x = N_y = 161$, $N_z = 21$, amounting to $\approx 4.8 \cdot 10^5$ internal grid points.

4.1. Algorithmic tests

In this section we show by some experiments the influence of the discretization parameters (in space and time) on the *numerical* behaviour of both methods. The influence on the *performance* will be discussed in the next section.

In Table 3 we present the cd-values for both concentrations, obtained by the OELH method on the three spatial grids and for various values of the time-step. Table 4 contains similar information for the RBWLH method.

Table 3 gives rise to the following remarks and conclusions:

- With respect to the time discretization we observe an increase with 0.6 in the cd_i-values on halving the time step, which is in agreement with the second-order consistency of the OELH method.
- The use of a second-order, three-point discretization in space (cf. (2.3)) is nicely observed if we compare the results on the different grids for extremely small Δt (headed by "$N \to \infty$"), for which the temporal error is negligible compared with the spatial error.
- Since $\max |u(t,x,y,z)| = \max |v(t,x,y,z)| \approx 1.58|d(t)| \leqslant 1.58$, the maximal stable time steps are in accordance with the stability condition (3.1).
- Furthermore we observe that, for fixed values of Δt, a refinement of the spatial grid does not generally result in an increased accuracy. It turned out that this behaviour is caused by the fact that we are dealing with *time-dependent boundary conditions*. It has been known for a long time that splitting methods usually exhibit a reduction of accuracy when the boundary conditions become time-dependent. The explanation for this phenomenon is that, at points adjacent to the boundary, the grid function is not sufficiently smooth, resulting in approximation errors of $O(h^2 + \Delta t^2/h^2)$ in these points, h denoting the distance to the boundary. Already in 1967, Fairweather and Mitchell [3] suggested a boundary-value correction for ADI methods to solve the Laplace equation (see also [9], where these ideas are extended to more general boundary conditions and to LOD methods). In the case of hopscotch type splitting, however, the derivation of these "Fairweather–Mitchell corrections" is much more complicated and we refrained from applying such corrections. In the present application, the "vertical" derivatives of the solution at the water surface and at the bottom are relatively large compared with the "horizontal" derivatives at the boundaries. Consequently, the approximations in the grid points adjacent to the surface and the bottom have an error involving the term $(\Delta z)^{-2}$, causing the behaviour as shown in Table 3.

Table 3
cd_1/cd_2 values for problem (4.1) with $T = 36{,}000$ obtained by the OELH method. N = the number of time steps ($\Delta t = T/N$); an unstable behaviour is indicated by an "*"

Spatial grid	$N = 35$	$N = 70$	$N = 140$	$N = 280$	$N = 560$	$N = 1120$	$N = 2240$	...	$N \to \infty$
$\text{Grid}_{\text{coarse}}$	2.9/1.9	3.3/2.5	3.3/3.1	3.3/3.5	3.3/3.5	3.3/3.5	3.3/3.5		3.3/3.5
$\text{Grid}_{\text{middle}}$	*	3.1/2.0	3.7/2.6	3.9/3.2	3.9/3.8	3.9/4.1	3.9/4.1		3.9/4.1
$\text{Grid}_{\text{fine}}$	*	*	3.1/1.9	3.8/2.6	4.4/3.2	4.5/3.8	4.5/4.4		4.5/4.7

Table 4
cd_1/cd_2 values for problem (4.1) with $T = 36{,}000$ obtained by the RBWLH method. N = the number of time steps ($\Delta t = T/N$); an unstable behaviour is indicated by an "*"

Spatial grid	$N = 35$	$N = 70$	$N = 140$	$N = 280$	$N = 560$	$N = 1120$	$N = 2240$ ...	$N \to \infty$
$\text{Grid}_{\text{coarse}}$	2.8/1.8	3.5/2.4	3.8/3.0	3.8/3.6	3.8/4.2	3.8/4.3	3.8/4.3	3.8/4.3
$\text{Grid}_{\text{middle}}$	*	2.9/1.8	3.6/2.5	4.3/3.0	4.8/3.7	4.8/4.3	4.8/4.9	4.8/5.2
$\text{Grid}_{\text{fine}}$	*	*	*	3.7/2.5	4.3/3.2	5.0/3.8	5.6/4.4	5.8/6.1

For the results produced by the RBWLH method we observe a similar behaviour in time as for the OELH method. The spatial discretization error, however, shows a third-order behaviour as is to be expected from the upwind discretization (2.6) that we used. Apart from reducing wiggles in the numerical solution, this high order discretization has the additional advantage that sufficiently small spatial errors are already obtained on a rather coarse grid. This is a nice property in view of the aforementioned accuracy reduction due to time-dependent boundary conditions.

4.2. *Performance results*

Both the OELH method and the RBWLH method have been implemented on the four-processor CRAY C98/4256 vector computer. In this section we give the performance results in scalar and vector mode for both codes. Vector mode is automatically achieved using the **cf77 -Zv** option of the CF77 compiling system. Megaflop rates (i.e., 10^6 floating point operations per second) and CPU times for the various routines are produced by the package **perfview** [1]. The speed-up that can be achieved owing to *vectorization* aspects of the codes has been tested using one (vector)processor. The optimal vector speed on one processor of the Cray equals 476 Mflops (in the exceptional case that a multiplication and an addition can always be chained, this theoretical peak performance is enhanced by a factor 2). These (vectorization) results are described in the next subsection. The capabilities that the codes offer with respect to *parallelization* are shown in Section 4.2.2. To that end we used the autotask facility of the Cray (activated by **cf77 -Zp**). Instead of running the codes on a dedicated system, we employed the utility **atexpert** [1], which produces predictions of the speed-up factors that can be obtained on a multi-processor system.

4.2.1. *Vectorization results*

We start with a survey of the global performance of the codes, to show the speed-up factors owing to vectorization. Since these factors depend on the vector length (i.e., on the number of grid points), we give results obtained on the three different spatial grids. For the OELH method, Table 5 shows the Megaflop rates for scalar and vector mode, the (average) CPU times needed for one time step and the resulting speed-up. Similar information for the RBWLH method is collected in Table 6.

From Tables 5 and 6 we conclude that:

- The speed-up factors are grid-dependent; in fact, the Mflop rates are reduced on coarser grids, due to the fact that in a number of subroutines vectorization in only one spatial direction is possible (viz., the innermost loop of length N_x). Since the hopscotch splitting also gives rise to a stride (2 for OELH and 3 for RBWLH), it is clear that the vector length on $\text{Grid}_{\text{coarse}}$ (having $N_x = 41$) is too short to achieve a vector speed close to peak performance.

Table 5
Global performance and speed-up factors obtained by the OELH method on various grids. The CPU times are per time step

	$\text{Grid}_{\text{coarse}}$		$\text{Grid}_{\text{middle}}$		$\text{Grid}_{\text{fine}}$	
	CPU (in sec.)	Mflop rate	CPU (in sec.)	Mflop rate	CPU (in sec.)	Mflop rate
Scalar mode	0.31	30.1	1.83	31.5	12.5	31.9
Vector mode	0.034	272	0.16	363	0.89	447
Speed-up factor	9.0		11.6		14.1	

Table 6
Global performance and speed-up factors obtained by the RBWLH method on various grids. The CPU times are per time step

	$\text{Grid}_{\text{coarse}}$		$\text{Grid}_{\text{middle}}$		$\text{Grid}_{\text{fine}}$	
	CPU (in sec.)	Mflop rate	CPU (in sec.)	Mflop rate	CPU (in sec.)	Mflop rate
Scalar mode	0.44	28.8	2.44	30.7	15.9	31.6
Vector mode	0.062	206	0.28	276	1.46	351
Speed-up factor	7.0		8.8		10.9	

– The computational amount of work per step is quite similar for both schemes: they have in common that $2N_xN_y$ linear systems have to be solved, and the "chemical stage" has to be iterated (which turns out to require an equal number of iterations for both schemes). The main difference is that an evaluation of an $\boldsymbol{H}^{\#}$-function in RBWLH is more expensive (due to an extended discretization molecule) and, of course, solving a pentadiagonal system takes more time than a tridiagonal system. From the CPU times given for the scalar mode version we see that this extra work for RBWLH results in an increase of the computational work of 27% on the fine grid to 42% on the coarse grid. However, a comparison of the vector performance of the two codes reveals that the CPU time per step is roughly 70% larger for RBWLH. To a large extent, this is explained by the superior vectorization properties of OELH (compare the speed-up factors).

In Tables 7 and 8 we present the performance results of the main routines in the codes, obtained on the various grids. These routines are: the subroutines H and SOURCE (for computing, per concentration, the discretized advection–diffusion terms, and the inhomogeneous terms g_i, respectively), the subroutine CHEMST (for treating the chemical stage), JACOB (to calculate the Jacobian matrices corresponding to each subset of grid points), TRI3D (in OELH) and BAND5 (in RBWLH) (to solve the linear systems), and the subroutine FCH (to calculate the chemical reaction terms in the right-hand side function). In these tables the symbol # indicates that the number of FCH-calls per step is not constant; obviously, the number of functional iterations to solve the "chemical stage" decreases when Δt is chosen smaller (but this number of iterations does not depend on the resolution of the spatial grid).

4.2.2. Parallelization aspects

As explained in Section 3.2, a few minor modifications in the code are necessary to exploit the *multi*-processor features of the Cray. A typical treatment of the autotasking facility is to collapse as many inner loops as possible for vectorization purposes and to use the next outer loop for parallelization. In

Table 7
Vector performance of the main routines in OELH

Routine	Number of calls per time step	$Grid_{coarse}$ Average time (in sec.)	$Grid_{coarse}$ Mflop rate	$Grid_{middle}$ Average time (in sec.)	$Grid_{middle}$ Mflop rate	$Grid_{fine}$ Average time (in sec.)	$Grid_{fine}$ Mflop rate
H	10	$4.6 \cdot 10^{-4}$	181	$2.5 \cdot 10^{-3}$	295	$1.5 \cdot 10^{-2}$	414
SOURCE	10	$1.6 \cdot 10^{-3}$	236	$7.6 \cdot 10^{-3}$	357	$4.3 \cdot 10^{-2}$	473
CHEMST	1	$3.9 \cdot 10^{-3}$	396	$1.8 \cdot 10^{-2}$	395	$1.0 \cdot 10^{-1}$	392
JACOB	4	$1.9 \cdot 10^{-4}$	69	$9.3 \cdot 10^{-4}$	108	$5.3 \cdot 10^{-3}$	151
TRI3D	8	$9.7 \cdot 10^{-5}$	521	$7.0 \cdot 10^{-4}$	536	$5.5 \cdot 10^{-3}$	530
FCH	#	$1.5 \cdot 10^{-4}$	703	$8.8 \cdot 10^{-4}$	712	$6.1 \cdot 10^{-3}$	706

Table 8
Vector performance of the main routines in RBWLH

Routine	Number of calls per time step	$Grid_{coarse}$ Average time (in sec.)	$Grid_{coarse}$ Mflop rate	$Grid_{middle}$ Average time (in sec.)	$Grid_{middle}$ Mflop rate	$Grid_{fine}$ Average time (in sec.)	$Grid_{fine}$ Mflop rate
H	14	$1.1 \cdot 10^{-3}$	123	$5.9 \cdot 10^{-3}$	200	$3.6 \cdot 10^{-2}$	282
SOURCE	14	$1.5 \cdot 10^{-3}$	175	$6.6 \cdot 10^{-3}$	278	$3.4 \cdot 10^{-2}$	400
CHEMST	1	$3.9 \cdot 10^{-3}$	393	$1.8 \cdot 10^{-2}$	393	$8.4 \cdot 10^{-2}$	394
JACOB	6	$5.9 \cdot 10^{-4}$	59	$3.2 \cdot 10^{-3}$	97	$1.7 \cdot 10^{-2}$	149
BAND5	12	$1.0 \cdot 10^{-4}$	541	$7.8 \cdot 10^{-4}$	557	$6.1 \cdot 10^{-3}$	577
FCH	#	$1.5 \cdot 10^{-4}$	707	$9.1 \cdot 10^{-4}$	692	$6.1 \cdot 10^{-3}$	702

Table 9
Parallel performance of the code RBWLH, estimated by **atexpert**

# processors	2	4	6	8	10	12	14	16
speed-up on $Grid_{coarse}$	1.92	3.63	5.05	5.97	7.00	8.25	8.46	8.64
speed-up on $Grid_{middle}$	1.95	3.75	5.41	6.90	7.80	9.65	9.92	11.17
speed-up on $Grid_{fine}$	1.96	3.75	5.41	6.94	8.14	9.71	10.43	12.06

some subroutines, this approach causes a slight reduction of the vector performance on *each* processor; however, the speed-up factors owing to multi-processing give ample compensation for this effect (see also the discussion in Section 3.2). The results of the **atexpert** utility, when running the code RBWLH on the various grids, are given in Table 9. For the fine grid, we observe a behaviour which is pretty close to linear speed-up. In the case of many processors, the coarser grids show a degradation of the parallel performance. This is due to the fact that the value of N_y (which controls the outer loops that are taken for parallelization) is not large enough to efficiently distribute N_y subtasks over many processors.

5. Concluding remarks

In this paper we have discussed two hopscotch type splitting methods for solving a three-dimensional transport model coupled with bio-chemical terms. The main difference between these methods is that in the first one (termed OELH) symmetric, second-order spatial discretizations have been used, whereas the second method (RBWLH) is based on third-order upwind discretizations. The advantage of the latter approach is that so-called "wiggles" in the solution are reduced to a large extent. This is a highly desired property, since wiggles may lead to negative concentrations, which are "unacceptable" from a physical point of view. The price to pay is, of course, a more complicated (and hence more expensive) spatial discretization, a reduced vector speed, and a reduced stability limit. The first and second aspect accumulate to an increase of the total CPU time by roughly a factor 1.7. If both schemes use their largest possible time step allowed by the respective stability conditions, then an additional factor 1.4 has to be taken into account.

The motive for constructing these particular methods is that a problem of this size can hardly be solved by a standard technique. The only feasible way to perform *realistic* simulations, especially over long real-time intervals, is to tailor the method to this specific application. Several considerations have led to these hopscotch type methods; for example, the fact that we are dealing with *shallow* seas introduces stiffness in the system of ODEs, caused by small values of Δz. This is exactly the reason for choosing a splitting function which treats the vertical terms implicitly. The weak coupling in the horizontal, on the other hand, makes it possible to treat the horizontal terms explicitly (and hence cheaply), without imposing a severe restriction on the time step. In this way, the amount of implicitness has been minimized. Other useful characteristics of these methods are: modest storage requirements, second order accuracy in time, and the possibility to easily embed the methods within a domain decomposition framework (owing to the weak coupling in the horizontal).

Apart from the above advantages, which can be considered as being of algorithmic nature, both methods allow for an efficient implementation on multi-vector computers. In Section 3.2 we showed how to vectorize the solution of the linear systems, resulting in speed-up factors of 20 and more. Also the treatment of the chemical terms, which have been "separated" from the advection/diffusion terms, shows a vector speed close to peak performance (see Section 4.2). Of course, several other routines in the code are less suitable for vectorization, thus reducing the overall vector speed-up. For OELH we obtained an overall speed-up ranging from 9 (obtained on a coarse mesh) to 14 (on a fine mesh). For RBWLH these numbers are 7 and 11, respectively.

On top of this speed-up owing to vectorization, a reduction of the total CPU time can be obtained by exploiting more than one (vector) processor. In Section 4.2.2 we showed that multi-processing leads to a speed-up close to linear.

Hence, combining these three aspects (viz., *algorithmic* tuning of the methods to the problem at hand, and taking care of good *vectorization* and *parallelization*) leads to a resulting code by which realistic simulations become feasible.

Acknowledgements

Professor Dr. P.J. van der Houwen is gratefully acknowledged for his continuous interest in these investigations and for his many valuable suggestions.

Appendix A. The algorithms TRI3D and BAND5

Here we present the algorithms for solving the tridiagonal and pentadiagonal linear systems as they occur in OELH and RBWLH, respectively. In both algorithms we first form the matrix $A := I - \beta J$, where β always equals $\frac{1}{4}\Delta t$ (cf. (2.5) and (2.8)). Then this matrix is decomposed, followed by a forward/backward substitution. The algorithms are designed in such a way that the number of divisions is minimized. In the actual implementation the input matrices and the right-hand side vectors are overwritten by the results of the decompositions and the solution vectors, respectively. For presentation purposes, however, we prefer to describe the methods in an *algorithmic* way, in terms of the 'triangular matrices' L and U. Therefore, in the case that the subscripts are < 1 or $> N_z$, the corresponding element is assumed to have a zero value.

The algorithms for one system $(I - \beta J)x = b$ of dimension N_z read:

TRI3D Algorithm. The matrix A is decomposed into the form $LD^{-1}U$, with the triangular matrices L and U having diagonal elements equal to 1, and D a diagonal matrix. Subsequently the systems in $LD^{-1}Ux = b$ are solved. The number of operations amounts to $8N_z$ floating-point multiplications, $4N_z$ floating-point additions and N_z divisions.

FOR $i = 1(1)N_z$ DO

$A_{i-1,i} := -\beta * A_{i-1,i}$ {we do not need to store the elements of U, however, the upper-diagonal of A must be updated to belong to $I - \beta J$}

$L_{i,i-1} := -\beta * A_{i,i-1} * D_{i-1}$

$D_i := 1/(1 - \beta * A_{i,i} - L_{i,i-1} * A_{i-1,i})$

{we have now obtained the decomposition results L and D}

FOR $i = 1(1)N_z$ DO $y_i := b_i - L_{i,i-1} * y_{i-1}$

FOR $i = N_z(-1)1$ DO $x_i := D_i * (y_i - A_{i,i+1} * x_{i+1})$

BAND5 Algorithm. The matrix A is decomposed into LU-form, with the lower-triangular matrix L having unit diagonal elements, followed by solving the systems in $LUx = b$. The number of operations amounts to $16N_z$ floating-point multiplications, $9N_z$ floating-point additions and N_z divisions.

FOR $i = 1(1)N_z$ DO

$L_{i,i-2} := -\beta * A_{i,i-2} * U^{-1}_{i-2,i-2}$

$U_{i-2,i} := -\beta * A_{i-2,i}$

$L_{i,i-1} := (-\beta * A_{i,i-1} - L_{i,i-2} * U_{i-2,i-1}) * U^{-1}_{i-1,i-1}$

$U_{i-1,i} := -\beta * A_{i-1,i} - L_{i-1,i-2} * U_{i-2,i}$

$U^{-1}_{i,i} := 1/(1 - \beta * A_{i,i} - L_{i,i-2} * U_{i-2,i} - L_{i,i-1} * U_{i-1,i})$

{we have now obtained the decomposition results L, $U^{-1}_{i,i}$ and the codiagonals of U}

FOR $i = 1(1)N_z$ DO $y_i := b_i - L_{i,i-2} * y_{i-2} - L_{i,i-1} * y_{i-1}$

FOR $i = N_z(-1)1$ DO $x_i := U^{-1}_{i,i} * (y_i - U_{i,i+1} * x_{i+1} - U_{i,i+2} * x_{i+2})$

References

[1] Cray Research, Inc., *UNICOS Performance Utilities Manual*, SR-2040, Edition 7.0.
[2] D. Dunsbergen, Particle models for transport in three-dimensional shallow water flow, Ph.D. Thesis, Delft Technical University (1994).
[3] G. Fairweather and R. Mitchell, A new computational procedure for ADI methods, *SIAM J. Numer. Anal.* 4 (1967) 163–170.
[4] G.H. Golub and C.F. van Loan, *Matrix Computations* (John Hopkins, University Press, Baltimore, MD, 2nd ed., 1989).
[5] A.R. Gourlay, Hopscotch: a fast second order partial differential equation solver, *J. Inst. Math. Appl.* 6 (1970) 375–390.
[6] P.J. van der Houwen and J.G. Verwer, One-step splitting methods for semi-discrete parabolic equations, *Computing* 22 (1979) 291–309.
[7] P.J. van der Houwen and B.P. Sommeijer, Splitting methods for three-dimensional transport models with interaction terms, CWI Report NM-R9516; also: *Z. Angew. Math. Mech.* 76 (1996).
[8] B. van Leer, Upwind-difference methods for aerodynamic problems governed by the Euler equations, in: B.E. Engquist, S. Osher and R.C.J. Sommerville, eds., *Proceedings of the 15th AMS–SIAM Summer Seminar on Applied Mathematics*, Scripps Institution of Oceanography, 1983, Lectures in Applied Mathematics 22, Part 2 (AMS, Providence, RI, 1985) 327–336.
[9] B.P. Sommeijer, P.J. van der Houwen and J.G. Verwer, On the treatment of time-dependent boundary conditions in splitting methods for parabolic differential equations, *Internat. J. Numer. Methods Engrg.* 17 (1981) 335–346.
[10] B.P. Sommeijer, P.J. van der Houwen and J. Kok, Time integration of three-dimensional numerical transport models, *Appl. Numer. Math.* 16 (1994) 201–225.
[11] B.P. Sommeijer and J. Kok, Implementation and performance of the time integration of a 3D numerical transport model, *Internat. J. Numer. Methods Fluids* 21 (1995) 349–367.
[12] B.P. Sommeijer and J. Kok, A vector/parallel method for a three-dimensional transport model coupled with bio-chemical terms, CWI Report NM-R9503.
[13] G. Strang, On the construction and comparison of difference schemes, *SIAM J. Numer. Anal.* 5 (1968) 506–517.
[14] J.G. Verwer and B.P. Sommeijer, Stability analysis of an odd-even line hopscotch method for three-dimensional advection–diffusion problems, CWI Report NM-R9422; also: *SIAM J. Numer. Anal.* (1997).