ELSEVIER

Applied Numerical Mathematics 16 (1994) 129–156

APPLIED NUMERICAL MATHEMATICS

VLUGR3: a vectorizable adaptive grid solver for PDEs in 3D, Part I: Algorithmic aspects and applications ★

J.G. Blom*, J.G. Verwer

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

This paper is dedicated to Professor Robert Vichnevetsky to honor him on the occasion of his 65th birthday

Abstract

This paper describes an adaptive-grid finite-difference solver for time-dependent three-dimensional systems of partial differential equations. The robustness and the efficiency of the solver is illustrated by the application of the code to three example problems. The performance of the solver is measured both for vector and scalar processors.

Keywords: Software; Application in physical sciences; Partial differential equations; Method of Lines; Adaptive-grid methods; Nonsymmetric sparse linear systems; Iterative solvers; Vectorization

1. Introduction

To solve time-dependent partial differential equations (PDEs) often a straightforward Method of Lines (MoL) approach is used. The PDE is discretized in space and the resulting system of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) is solved in time with a robust "off the shelf" solver. The spatial grid used can be either uniform or nonuniform but the location of the grid points is invariable for the time interval. For problems with steep and moving fronts this can be an inefficient approach since too many grid points are needed in areas where the solution has at any time large spatial gradients. For this reason many attempts have been made to develop adaptive-grid methods, where the grid is adjusted to the solution either dynamically or

★ This work was supported by Cray Research, Inc., under grant CRG 93.03, via the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF).

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

SSDI 0168-9274(94)00047-6

statically (e.g., after each time step). This introduces of course a certain amount of overhead but when the use of implicit time integrators is profitable or when the number of PDE components is very large such solvers can be more efficient in two dimensions. Often the standard approach will no longer be feasible in three dimensions. The inefficiency of the computations is one reason but it is even more important that the physical memory of most computers will not be large enough to contain the solution values at all grid points plus the work space needed for the solution of the nonlinear system.

In previous work [16–18,22,19,23,15] an adaptive-grid finite-difference method to solve time-dependent two-dimensional systems of PDEs was developed. This method couples Local Uniform Grid Refinement (LUGR) with an implicit ODE solver and proved to be robust and efficient for the target problem class. The code VLUGR2 [3] is a vectorized implementation of this method based on the research code MOORKOP [13] by Trompert. In this paper we describe the algorithmic aspects of VLUGR3 which is the extension of VLUGR2 to the three-dimensional case (see also [5]). This vectorizable adaptive-grid finite-difference code has been designed to solve initial–boundary-value problems for systems that fit in the following master equation

$$\begin{aligned} &\mathcal{F}(t, x, y, z, \boldsymbol{u}, \boldsymbol{u}_t, \boldsymbol{u}_x, \boldsymbol{u}_y, \boldsymbol{u}_z, \boldsymbol{u}_{xx}, \boldsymbol{u}_{yy}, \boldsymbol{u}_{zz}, \boldsymbol{u}_{xy}, \boldsymbol{u}_{xz}, \boldsymbol{u}_{yz}) = 0, \\ &\quad (x, y, z) \in \Omega, \quad t > t_0, \end{aligned} \tag{1}$$

where the solution $\boldsymbol{u}$ may be a vector and the domain Ω an arbitrary domain with a possibly disjunct "brick" structure, i.e., a domain that can be described by right-angled polyhedrons. The boundary conditions belonging to system (1) are formulated as

$$\mathcal{B}(t, x, y, z, \boldsymbol{u}, \boldsymbol{u}_t, \boldsymbol{u}_x, \boldsymbol{u}_y, \boldsymbol{u}_z) = 0, \qquad (x, y, z) \in \partial\Omega, \quad t > t_0, \tag{2}$$

nd the initial conditions satisfy

$$\boldsymbol{u}(t_0, x, y, z) = \boldsymbol{u}_0(x, y, z), \quad (x, y, z) \in \Omega \cup \partial\Omega. \tag{3}$$

The paper is an extension of report [2], where some first test results were discussed. There the nonlinear systems were solved with modified Newton and the linear systems with BiCGStab [20] with ILU preconditioning. Since the test results in [2] indicated that memory demands could be a bottleneck for some problems, we added to our code a matrix-free implementation of the Newton process with linear system solver GCRO [12]. As preconditioner we use in this case a simple (block-)diagonal scaling.

To evaluate the robustness and the efficiency of the adaptive-grid algorithm in 3D as well as the efficiency of the implementation and its performance on a vector processor, we tested the code extensively on a set of three example evolutionary problems in 3D with various solution characteristics, viz.,

- Burgers' equation with an exact solution representing a wave moving skew through the unit cube,
- a rotating sphere problem described by a system of two advection–reaction equations, and
- a model for the simulation of pollution in ground-water flow.

The paper is organized as follows. In Section 2 we give an outline of the Local Uniform Grid Refinement method as implemented in VLUGR3 and the choices made in the code with respect to

the refinement strategy and the time-integration strategy including the solution of the (non)linear systems. For a more detailed discussion on the subject of the implementation of the code and how to use it we refer to the companion paper [5]. Section 3 contains the test results for the three example problems. In this section we also look into the robustness of the code on the basis of some numerical experiments. In Section 4 we discuss the (vector) performance of the solver for the example problems. Finally, Section 5 contains a summary of our findings.

2. The algorithm

2.1. Local Uniform Grid Refinement

There are two reasons to adapt a spatial grid in time to the solution. The first is to speed up the computations since the PDE is evaluated in significantly less points than what would have been the case if one uses a fixed grid. The second reason is to restrict the amount of memory needed. Naturally both goals should be reached without sacrificing overall accuracy.

The advantage of static regridding, and especially of Local Uniform Grid Refinement, is that it is robust and conceptually very simple. The domain is covered by a uniform coarse base grid and nested finer uniform subgrids are recursively created in regions with high spatial activity. So all grids consist of one or more disjunct sets of interconnected grid cells, all having the same size. When a grid of a specific refinement level has been created the corresponding initial boundary value problem is solved on the current time interval.

In short, an implicit solver based on the LUGR method can be described by

(0) Start with the coarse base grid, the initial solution and an initial time step.
(1) Solve the initial–boundary-value problem on the current grid with the current time step.
(2) If the required resolution in space is not yet reached:
  (a) Determine at the forward time level a new embedded grid of the next finer grid level.
  (b) Get solution values at previous time level(s) on the new grid.
  (c) Interpolate internal boundary values from the old grid at the forward time.
  (d) Get initial values for the Newton process at the forward time.
  (e) Goto (1).
(3) Inject fine grid solution values in coinciding coarser grid nodes.
(4) Estimate the error in the time integration. If the time error is acceptable:
  (a) Advance the time level.
(5) Determine the new step size, goto (1) with the coarse base grid as the current grid.

Where interpolation is needed to obtain solution values, linear interpolation is used.

The virtue of an LUGR method lies in the fact that one reaches the accuracy of a fine mesh with considerably less computational effort and memory requirements, since the fine subgrids cover only part of the domain. A necessity for the good functioning of an LUGR method is that the refinement strategy takes care that fine subgrids are created timely and at the correct places. In [17,18,22,23] the refinement strategy is based on a comprehensive error analysis of both the space discretization and the interpolation. Here we use a more simple strategy because the class of equations (1)–(3) is too general for the error analysis to be valid. The refinement strategy is the same as in VLUGR2, i.e.,

based on a curvature monitor.

For each grid point (i, j, k) the space monitor is determined by

$$\begin{aligned}&\mathtt{SPCMON}(i,j,k)\\&\quad:=\max_{ic=1,\mathtt{NPDE}}\mathtt{SPCTOL}(ic)\cdot\left(|\Delta x^2\cdot u_{xx}^{ic}(i,j,k)|+|\Delta y^2\cdot u_{yy}^{ic}(i,j,k)|+|\Delta z^2\cdot u_{zz}^{ic}(i,j,k)|\right),\end{aligned}\tag{4}$$

where Δx, Δy, and Δz are the grid width in the x-, the y- and the z-direction respectively, and

$$\mathtt{SPCTOL}(ic):=\frac{\mathtt{SPCWGT}(ic)}{\mathtt{UMAX}(ic)\cdot\mathtt{TOLS}}.\tag{5}$$

The variables on the right-hand side of (5) are user-specified quantities. Specifically, $0\leqslant\mathtt{SPCWGT}\leqslant 1$ is a weighting factor for the relative importance of a PDE component on the space monitor, UMAX the approximate maximum absolute value for each component, and TOLS the space tolerance. The second-order derivatives in (4) are approximated by second-order finite differences at the internal domain and first-order at the boundaries.

A next level of refinement is created if

$$\max_{(i,j,k)}\mathtt{SPCMON}(i,j,k)>\mathtt{TOLWGT}.\tag{6}$$

TOLWGT acts as a bar against fluctuations of the number of grid levels in subsequent time steps: if the next level of refinement existed at the previous time level TOLWGT = 0.9, otherwise TOLWGT = 1.0.

If a next level of refinement is required, then all grid points with

$$\mathtt{SPCMON}(i,j,k)>\tfrac{1}{4}\tag{7}$$

are flagged together with their 26 direct neighbors. A cell with at least one flagged corner is divided in 8 equal subcells, so we do not apply semi-refinement, i.e., refinement in only one direction parallel to an axis. The next grid level will contain all these refined cells.

2.2. *Discretizing the PDE*

In VLUGR3 standard second-order finite differences are used for the spatial discretization, central on the internal domain and one-sided at the boundaries. Due to the mixed derivatives we thus deal with a 19 point coupled grid stencil.

As time integrator we use the second-order two-step implicit BDF method with variable step sizes

$$\boldsymbol{U}_t=a_0\boldsymbol{U}^{n+1}+a_1\boldsymbol{U}^{n}+a_2\boldsymbol{U}^{n-1},\tag{8a}$$

with

$$a_0=\frac{1+2\alpha}{1+\alpha}\frac{1}{\Delta t},\quad a_1=-\frac{(1+\alpha)^2}{1+\alpha}\frac{1}{\Delta t},\quad a_2=\frac{\alpha^2}{1+\alpha}\frac{1}{\Delta t};\qquad \alpha=\frac{\Delta t}{\Delta t_{\mathrm{old}}}.\tag{8b}$$

In the first time step we apply as usual Backward Euler ($\alpha=0$ in (8)). Like in DASPK [8], we use for all error measurements in the time integration a weighted root-mean-square norm

$$\|\boldsymbol{v}\|_w=\|W\boldsymbol{v}\|_2,\tag{9}$$

where W is a diagonal matrix defined by

$$W = 1/\sqrt{N}\,\mathrm{diag}(w_1, \ldots, w_N). \tag{10}$$

The time integration is controlled by the solution monitor value in time which is computed at each existing grid level

$$\mathtt{TIMMON}(level) := \|\Delta t\, \boldsymbol{u}_t\|_w, \tag{11}$$

where Δt is the current time step size, $\boldsymbol{u}_t$ is approximated by first-order finite differences and the entries w_i of the diagonal matrix W from (10) are defined by

$$w_{ipt,ic} = \mathtt{TIMWGT}(ic)/(\mathtt{ABSTOL}(ic) + |U^{n+1}_{ipt,ic}| \cdot \mathtt{RELTOL}(ic)),$$
$$ipt \in \Omega, \quad ic = 1, \mathtt{NPDE}, \tag{12}$$

with

$$\mathtt{ABSTOL}(ic) = 0.01 \cdot \mathtt{TOLT} \cdot \mathtt{UMAX}(ic), \qquad \mathtt{RELTOL}(ic) = \mathtt{TOLT}. \tag{13}$$

Note that the time monitor is first-order although the used time-integration method is second-order. The reason is that an estimator based on interpolated solution values for one time level and computed solution values for another, leads readily to unnecessarily small time steps (cf. [16]). This is also the reason to exclude the boundary points in (11). The variables TIMWGT and TOLT are the user specified analogues of the variables in (5).

An integration step is rejected and redone at all grid levels if

$$\max_{level} \mathtt{TIMMON}(level) > 1.0. \tag{14}$$

A new step size is computed such that the prediction of the monitor at the next time point is 0.5, i.e.,

$$\Delta t_{\mathrm{new}} := \frac{0.5}{\max_{level} \mathtt{TIMMON}(level)} \cdot \Delta t. \tag{15}$$

If the step was accepted the increase in step size is restricted to a factor 2 and if the time step was rejected the decrease is restricted to a factor 4. Finally, the step size is restricted to a user specified minimum and maximum value and adjusted such that the rest of the integration interval is an integer number times Δt.

2.3. Solving the nonlinear system

Since we use an implicit time-integration method, at each time step a large system of nonlinear algebraic equations has to be solved. This is done using Newton's method in combination with a preconditioned iterative linear system solver. In our experiments this combination was more robust and most of the times less expensive than nonlinear Krylov solvers like, e.g., Nonlinear Orthomin [10] or so-called hybrid Krylov methods in which it is tried to speed up the convergence of the Newton process using information from the linear Krylov solver (see [9]). VLUGR3 offers two implementations of the Newton process: one where the Jacobian matrix

$$G = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}} \tag{16}$$

is computed and stored once for each grid level and each time step, and a matrix-free variant. In the latter the matrix–vector product Gv required in the linear system solver is approximated by a difference quotient. Note that the first option results in a *modified* Newton process, whereas the second option leads to a *true* Newton process. So the latter can have a better convergence behavior if the matrix–vector product is well enough approximated. Also note that in contrast to the common practice in MoL it is not possible to "freeze" the Jacobian and/or the preconditioner for a number of time steps, since the location and number of grid points of the refined grids can change every time step.

The linear system solvers are described in Section 2.4 and the approximation of the Jacobian and the matrix–vector product Gv in Section 2.5. Here we discuss some of the strategies implemented in the Newton procedure. Roughly spoken we have followed the approach as used in DASSL [7, pp. 123–124] and DASPK [8]. When solving the nonlinear system

$$\boldsymbol{F}(t, x, y, z, \boldsymbol{U}, \boldsymbol{U}_t, \boldsymbol{U}_x, \boldsymbol{U}_y, \boldsymbol{U}_z, \boldsymbol{U}_{xx}, \boldsymbol{U}_{yy}, \boldsymbol{U}_{zz}, \boldsymbol{U}_{xy}, \boldsymbol{U}_{xz}, \boldsymbol{U}_{yz}) = 0, \quad t > t_0, \tag{17}$$

which is the fully discretized form of (1) and (2), Newton generates linear systems of the form

$$G\boldsymbol{c}^{(k)} = -\boldsymbol{F}(\boldsymbol{U}^{(k)}). \tag{18}$$

These systems are solved with an arbitrary linear system solver and the solution is updated

$$\boldsymbol{U}^{(k+1)} = \boldsymbol{U}^{(k)} + \boldsymbol{c}^{(k)}. \tag{19}$$

The Newton process is continued until the iteration error $\|\boldsymbol{U} - \boldsymbol{U}^{(k)}\|_w$ is sufficiently small. For the Newton process and the underlying linear solvers the entries w_i of the diagonal matrix W in (10) are defined by

$$\begin{aligned} &w_{ipt,ic} = 1.0/(\mathtt{ATOL}(ic) + |U^{(0)}_{ipt,ic}| \cdot \mathtt{RTOL}(ic)), \\ &\quad ipt \in \Omega, \quad ic = 1, \mathtt{NPDE}, \end{aligned} \tag{20}$$

with

$$\mathtt{ATOL}(ic) = 0.01 \cdot \mathtt{TOL} \cdot \mathtt{UMAX}(ic), \qquad \mathtt{RTOL}(ic) = \mathtt{TOL}. \tag{21}$$

The user-set space and time tolerances TOLS and TOLT are buried in the tolerance

$$\mathtt{TOL} = 0.1 \min(\mathtt{TOLT}^2, \mathtt{TOLS}). \tag{22}$$

Assuming convergence of the Newton process the inequality

$$\|\boldsymbol{U} - \boldsymbol{U}^{(k)}\|_w \leqslant \frac{\rho}{1-\rho}\|\boldsymbol{U}^{(k)} - \boldsymbol{U}^{(k-1)}\|_w, \tag{23}$$

holds, where ρ is the convergence rate which is in actual computation approximated by

$$\rho \approx \rho^{(k)} = \left(\frac{\|\boldsymbol{U}^{(k)} - \boldsymbol{U}^{(k-1)}\|_w}{\|\boldsymbol{U}^{(k-1)} - \boldsymbol{U}^{(k-2)}\|_w}\right)^{1/2}. \tag{24}$$

This leads to the stopping criterion for the Newton iteration

$$\frac{\rho^{(k)}}{1-\rho^{(k)}}\|\boldsymbol{U}^{(k)} - \boldsymbol{U}^{(k-1)}\|_w < \mathtt{TOLNEW} = 1.0. \tag{25}$$

Note that we use the square of the user-defined time tolerance TOLT because the BDF formula (8) is second-order and the time monitor is first-order.

If during the Newton iteration $\rho^{(k)} > 0.9$ or if the maximum number of iterations is exceeded, a new Jacobian and/or preconditioner is computed, once, and the iteration is restarted. If the Newton process does not converge with the new Jacobian/preconditioner the time step is redone with $\Delta t = \Delta t/4$.

An extra difficulty, when solving nonlinear systems arising from the LUGR method, is to provide an initial solution. If one employs the Method of Lines on a single grid fixed in time the solution of the previous time level is in general a good initial estimate. In any case, if $\Delta t \to 0$ it is *the* solution. This is not the case with LUGR methods, however. The solution values injected from a finer grid are in general *not* a solution of the PDE system discretized on a coarser grid. So, even when $\Delta t \to 0$ it is still possible that the injected values on the previous time level are not a good initial estimate. This obviously is an exceptional situation, but for the nonlinear ground-water-flow problems we indeed experienced some serious problems with the convergence of the Newton process. So for the sake of robustness we also save the not-injected solution values, i.e., the solution values actually computed at the coarser grid and use these as initial estimate with interpolated values where a grid point at the previous time did not exist.

2.4. Solving the linear system

The Conjugate Gradient method is for symmetric systems an ideal iterative solver. It has the minimal residual property, i.e., the solution vector in the current Krylov subspace is chosen such that the residual vector is orthogonal to the subspace and it can be efficiently implemented using a three-term recurrency. Krylov solvers for nonsymmetric systems can roughly be divided in two groups. In the first the minimal residual property is maintained at the cost of a more expensive computational process (GCR, ORTHOMIN, GMRES). In the second the three-term recurrency is maintained and a nonoptimal descent algorithm results (BiCG, CGS, BiCGStab).

In VLUGR3 we implemented two Krylov solvers to solve the linear system (18). If the Jacobian matrix G and its ILU decomposition are available we use BiCGStab [20]. BiCGStab is a cheap solver (in memory and in CPU time) that appeared to be robust enough when combined with the ILU preconditioner. In the second case the matrix–vector multiplication Gv required in the linear system solver is approximated by a difference quotient (see (35)). For this matrix-free variant a comparably robust, standard preconditioner that does not make explicit use of the matrix G does not exist. Moreover, the matrix–vector multiplication can be rather expensive for some PDEs. Therefore we chose for this variant GMRES to have an optimal convergence behavior and so a minimal number of matrix–vector multiplications. As preconditioner we use (block-)diagonal scaling (see below in Section 2.6). However, in some cases the number of iterations and thus the number of storage required can be very large. To restrict the amount of storage one can of course restart the GMRES process after a certain number of iterations. However it is more profitable to use a recursive version of GCR in which the inner GMRES loop generates a better preconditioner and the outer GCR loop solves a well-preconditioned system. In VLUGR2 [3] we used GMRESR [21] and we reported some stagnations. In [12] a remedy is given against these stagnations. The idea is that the new Krylov basis vectors in the inner loop should not only be orthogonal to the previous ones but also to the corrections on the approximate solution already computed in the outer loop. This leads to the GCRO algorithm which is per iteration slightly more expensive than GMRESR (extra orthogonalizations)

but which in general uses less iterations and so less matrix–vector multiplications than GMRESR. A second reason not to use BiCGStab in the matrix-free variant was that in our experience BiCGStab was more sensitive to the inaccuracies induced by the approximation of Gv by a difference quotient than GMRES-like Krylov solvers, possibly because GMRES orthogonalizes the Krylov base after the matrix–vector multiplication.

It is not necessary to solve the linear system (18) up to machine precision (cf. [6,19]). The stopping criterion we use in our code is

$$\|P^{-1}r^{(l)}\|_w < \texttt{TOLLSS}, \qquad \texttt{TOLLSS} = \texttt{TOLNEW}/(10 \cdot 2^{\texttt{NIT}}), \tag{26}$$

where NIT is the current Newton-iteration index and P^{-1} is the preconditioner in use. In the case of GCRO we take the maximum of $\|D^{-1}r^{(l)}\|_w$ and $\|P^{-1}r^{(l)}\|_w$, where D is the (block-)diagonal preconditioner and P is obtained by the inner GMRES iterations. The inner GMRES iterations are stopped if the two-norm of the (weighted diagonally scaled) residual is less than TOLLSS and if a relative improvement of the residual of 0.01 was reached. Note that if P is a good approximation of G, then $P^{-1}r^{(l)}$ is a good approximation of the contribution to the Newton correction $c^{(k)}$ of the correction $x^{(l)} - x^{(l-1)}$ on the linear solution.

2.5. *The matrix–vector product Gv*

Approximation of the Jacobian

If we use modified Newton+BiCGStab to solve the nonlinear system the Jacobian is computed by numerical differencing. The matrix G, resulting from a second-order central discretization on the internal domain and second-order one-sided differences on the boundaries, consists of 19 diagonal blocks per rowblock. The storage and the matrix–vector multiplication are implemented analogously to the 2D case (cf. [3]). So G is stored in block 19-diagonal storage mode, i.e., in an array G(NPTS,NPDE,NPDE,-9:9). The placing of the 3 grid points coupled in x-direction is known. The actual placing of the grid points corresponding with the 8 lower diagonals is given in the array LLDG(NPTS,-9:-2) and for the 8 upper diagonals in LUDG(NPTS,2:9). If the matrix is stored in this way, the matrix–vector product can be vectorized very efficiently.

The Jacobian

$$G = \frac{\partial \boldsymbol{F}(t, x, y, z, \boldsymbol{U}, \boldsymbol{U}_t, \boldsymbol{U}_x, \boldsymbol{U}_y, \boldsymbol{U}_z, \boldsymbol{U}_{xx}, \boldsymbol{U}_{yy}, \boldsymbol{U}_{zz}, \boldsymbol{U}_{xy}, \boldsymbol{U}_{xz}, \boldsymbol{U}_{yz})}{\partial \boldsymbol{U}} \tag{27}$$

is approximated by numerical differencing. To save residual evaluations we have made use of the identity

$$\begin{aligned}
\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}} = {} & \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}, \boldsymbol{U}_t, \cdot)}{\partial \boldsymbol{U}} \\
& + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_x, \cdot)}{\partial \boldsymbol{U}_x} \cdot \frac{\partial \boldsymbol{U}_x}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_y, \cdot)}{\partial \boldsymbol{U}_y} \cdot \frac{\partial \boldsymbol{U}_y}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_z, \cdot)}{\partial \boldsymbol{U}_z} \cdot \frac{\partial \boldsymbol{U}_z}{\partial \boldsymbol{U}} \\
& + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{xx}, \cdot)}{\partial \boldsymbol{U}_{xx}} \cdot \frac{\partial \boldsymbol{U}_{xx}}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{yy}, \cdot)}{\partial \boldsymbol{U}_{yy}} \cdot \frac{\partial \boldsymbol{U}_{yy}}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{zz}, \cdot)}{\partial \boldsymbol{U}_{zz}} \cdot \frac{\partial \boldsymbol{U}_{zz}}{\partial \boldsymbol{U}} \\
& + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{xy}, \cdot)}{\partial \boldsymbol{U}_{xy}} \cdot \frac{\partial \boldsymbol{U}_{xy}}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{xz}, \cdot)}{\partial \boldsymbol{U}_{xz}} \cdot \frac{\partial \boldsymbol{U}_{xz}}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_{yz}, \cdot)}{\partial \boldsymbol{U}_{yz}} \cdot \frac{\partial \boldsymbol{U}_{yz}}{\partial \boldsymbol{U}}.
\end{aligned} \tag{28}$$

The implementation of (28) is very simple and vectorizes well. If we approximate the partial derivatives of $\boldsymbol{F}$ in the right-hand side of (28) by numerical differencing, the perturbation is only local to a grid point and therefore the Jacobian can be obtained by 10 residual evaluations. A further advantage of this implementation is that different space or time discretizations only lead to different multiplying factors in (28). The way to compute the partial derivative of $\boldsymbol{F}$ with respect to $\boldsymbol{U}$ is copied from DASSL [7, p. 124]

$$\frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}, \boldsymbol{U}_t, \cdot)}{\partial \boldsymbol{U}} \approx \frac{\boldsymbol{F}(\cdot, \boldsymbol{U} + \boldsymbol{\Delta}, \boldsymbol{U}_t + a_0 \boldsymbol{\Delta}, \cdot) - \boldsymbol{F}(\cdot, \boldsymbol{U}, \boldsymbol{U}_t, \cdot)}{\boldsymbol{\Delta}}, \tag{29}$$

where

$$\Delta_{ipt,ic} = \sqrt{\text{uround}} \cdot \text{sign}(\Delta t\, U_{tipt,ic}) \cdot \max(|U_{ipt,ic}|, |\Delta t\, U_{tipt,ic}|, \texttt{ATOL}(ic)). \tag{30}$$

For the other partial derivatives we use

$$\frac{\partial \boldsymbol{F}(\cdot, \boldsymbol{U}_p, \cdot)}{\partial \boldsymbol{U}_p} \approx \frac{\boldsymbol{F}(\cdot, \boldsymbol{U}_p + \boldsymbol{\Delta}_p, \cdot) - \boldsymbol{F}(\cdot, \boldsymbol{U}_p, \cdot)}{\boldsymbol{\Delta}_p} \quad \text{for } p = x, y, z, xx, yy, zz, xy, xz, yz. \tag{31}$$

The main difficulty in the numerical computation of the partial derivatives is the choice of the perturbation vector $\boldsymbol{\Delta}_p$, and especially to decide when the value to be perturbed should be considered zero. We chose the perturbation

$$\Delta_{p_{ipt,ic}} = \sqrt{\text{uround}} \cdot \max(|U_{p_{ipt,ic}}|, \texttt{ATOL}(ic) \cdot fac_p), \tag{32}$$

where

$$fac_x = \frac{1}{2\Delta x}, \qquad fac_{xx} = \frac{1}{\Delta x^2}, \qquad fac_{xy} = \frac{1}{4\Delta x \Delta y},$$

and other combinations analogously. So $\texttt{ATOL}/2\Delta x$ is considered to be the noise value for U_x if ATOL is the noise value for U. Finally, the "magic tricks"

$$\Delta_{p_{ipt,ic}} = \text{sign}(U_{p_{ipt,ic}}) \cdot \Delta_{p_{ipt,ic}} \tag{33}$$

and

$$\boldsymbol{\Delta} = (\boldsymbol{U} + \boldsymbol{\Delta}) - \boldsymbol{U} \tag{34}$$

have been applied to ensure that the perturbed value has the same sign as the original one and that the perturbed value is a true machine number. Note that in (30) the former is not necessarily true. If the PDE is, e.g., undefined for negative values of U this can occasionally be a source of difficulties.

This way of computing a Jacobian was sufficiently accurate for the problems we solved with the code. However, if for a specific problem Newton failures would occur more often than time step failures, it could be worthwhile to store the exact partial derivatives instead of the approximated ones.

Approximation of $G\boldsymbol{v}$

The matrix–vector multiplication $G\boldsymbol{v}$ required in the linear system solvers can be approximated by a difference quotient

$$G(\boldsymbol{U})\boldsymbol{v} = \frac{\boldsymbol{F}(\boldsymbol{U} + \sigma \boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{U})}{\sigma}. \tag{35}$$

The problem here of course is how to choose σ so that the vector U is appropriately perturbed. One approach is a generalization of the way the matrix elements in the Jacobian are approximated (cf. [9]). This results in

$$\sigma = \frac{\sqrt{\text{uround}}}{\langle v, v\rangle} \cdot \text{sign}(\Delta t\, \dot{U}^{\mathrm{T}} v) \cdot \max(|U^{\mathrm{T}} v|, |\Delta t\, \dot{U}^{\mathrm{T}} v|, \overline{\texttt{ATOL}}\,|v|), \tag{36}$$

where

$$\overline{\texttt{ATOL}} = (\texttt{ATOL(1)}, \ldots, \texttt{ATOL(NPDE)}, \ldots, \texttt{ATOL(1)}, \ldots, \texttt{ATOL(NPDE)})^{\mathrm{T}}$$

and $|v| = (|v_1|, \ldots, |v_N|)$. However this is not very robust if the components of the solution differ in orders of magnitude. Therefore we adopted the approach proposed in [8], i.e., we solve

$$(WD^{-1}GW^{-1}) \cdot (Wc) = -(WD^{-1}F). \tag{37}$$

This has the advantage that the matrix–vector multiplications needed in GMRES are of the form Ax with $A = WD^{-1}GW^{-1}$ and $\|x\|_2 = 1$. So in the $G(U)v$ approximation $\|v\|_w = \|Wv\|_2 = 1$. Since the Newton correction is assumed small with respect to the solution and the required tolerance if $\|c\|_w = 1$, this approach means that also the vector v is assumed small with respect to the solution U. So we can take $\sigma = 1.0$ in the difference quotient (35).

In the outer GCR loop we have to scale the multiplying vector with its two-norm before the matrix–vector multiplication is executed. However, the two-norm of this vector is needed for the error estimation, so the overhead is low.

2.6. *Preconditioning*

In our first linear solver BiCGStab is combined with a standard ILU type preconditioner, i.e., only those entries of the decomposition are computed that correspond with nonzero entries in the Jacobian. For the boundaries we use in the ILU preconditioner first-order discretization to maintain the 19-block diagonal structure. The ILU preconditioning is vectorized using a variant of the hyperplane method [1]. For the two-dimensional case this is described in [4]. The extension to 3D is straightforward. This linear solver is in our experience robust and rather efficient with respect to the number of iterations needed. Moreover, in contrast to the 2D case the vector performance of the ILU preconditioning is satisfactorily in 3D. A disadvantage of our first linear solver is the huge amount of memory needed, especially if the number of PDE components is large.

As preconditioner in our second linear solver we offer the choice between diagonal scaling and block-diagonal scaling. The latter should be used in case the coupling between the PDE components is such that the main diagonal is not a good enough approximation for the Jacobian. The (block-)diagonal preconditioner is computed analogously as the main (block-)diagonal entries of the Jacobian. Since the mixed derivatives have no influence on the main diagonal, 7 (or 7 · NPDE) residual evaluations are required to compute the preconditioner. Optionally, the contributions of the first-order derivatives at the boundaries can be neglected. We then can compute the (block-)diagonal of the Jacobian with only 4 (or 4 · NPDE) residual evaluations.

3. Numerical experiments

In this section we discuss the test results obtained with VLUGR3 for three example problems, viz., (I) Burgers' equation, a model for nonlinear convection–diffusion phenomena, (II) a system of coupled advection–reaction equations, and (III) a transport problem in ground-water flow. The main issue is to evaluate the robustness of the LUGR method. To that aim we consider for both nonlinear equation solvers the accuracy of the solution and the integration history for the example problems. The results are presented in tables and for the problem without reference solution (III) also in plots. Typical grid refinement behavior is also displayed in plots for all problems. In the description of the experiments the following notation is used:

- STEPS: number of successful time steps,
- JACS: number of Jacobian and/or preconditioner evaluations,
- NIT: number of Newton iterations,
- LSIT i: number of linear-solver iterations in the ith Newton iteration.

For all problems we set the space and time tolerance TOLS = TOLT = 0.1. The weight factors SPCWGT and TOLWGT are set to 1 for all components. For problems I and II UMAX is also set to 1 for all components. For our first linear solver BiCGStab+ILU the maximum number of linear iterations was set at 100, which was never reached. In our second solver, GCRO+(block-)diagonal preconditioning, we imposed a maximum of 20 inner GMRES iterations and 5 outer GCR iterations. We allowed also one restart. The maximum for the number of inner iterations was often reached but the linear solver as a whole only occasionally failed using diagonal scaling and never using the block-diagonal preconditioner.

To assess the local-refinement efficiency, i.e., to compare the LUGR code with a standard MoL approach on a (non)uniform fixed grid one should take into account two points, viz., use of memory and computational efficiency. When solving the linear systems with BiCGStab and ILU preconditioning the amount of memory is dominated by the two arrays needed to store the Jacobian and the preconditioner. Both take up 19 · NPDE · NPTS · NPDE floating-point words, where NPTS is the actual number of points in the grid. The matrix-free variant requires a number of vectors of length NPTS·NPDE. Since the memory requisite is in both cases determined by the number of grid points, the LUGR method will be readily more efficient in its memory use. It is more difficult to compare the efficiency with respect to the computational effort. Assuming that the overall accuracy is the same, a first approximation would be to compare the number of grid points at the finest uniform grid with the total number of grid points used over all grid levels. However one should keep in mind two things. First, it is in general cheaper to solve two systems of N equations than one of order $2N$. Second, the LUGR method requires extra computations for the local refinement: the determination of a new grid and the transfer of solution values between two grid levels. These overhead costs are especially significant if the number of PDE components and the number of (non)linear iterations is small. The LUGR method could be implemented computationally more efficient by adding pointers between the grid structures at different levels but that would be less efficient in memory. Another option is to "freeze" a specific series of nested grids for a few time steps. A disadvantage of this approach is that it obscures the conceptual simplicity and probably decreases the robustness of the method.

In this section we discuss the robustness and the local-refinement efficiency of the LUGR method. The vector performance and CPU time for all three example problems will be discussed in the next section. Our experiments were done on a Cray Y-MP with a memory limit of 52 Mword.

3.1. *Problem* I: *Burgers' equation*

This model, a simple 3D analogue of the Navier–Stokes equations, has a nonlinear convection term combined with a small diffusion term

$$\begin{aligned} &u_t + uu_x + vu_y + wu_z = \varepsilon \Delta u, \\ &v_t + uv_x + vv_y + wv_z = \varepsilon \Delta v, \\ &w_t + uw_x + vw_y + ww_z = \varepsilon \Delta w. \end{aligned} \tag{38}$$

We determined an exact solution for (38) representing a wave front moving skew through a unit cube. Using the Cole–Hopf transformation (see, e.g., [24, p. 97])

$$u = -2\varepsilon \frac{\varphi_x}{\varphi}, \qquad v = -2\varepsilon \frac{\varphi_y}{\varphi}, \qquad w = -2\varepsilon \frac{\varphi_z}{\varphi}, \tag{39}$$

system (38) is transformed into

$$\varphi_t = \varepsilon \Delta \varphi. \tag{40}$$

An exact solution representing a single shock is given by (cf. [24, p. 110])

$$\varphi = f_1 + f_2, \quad f_j = \exp\left(-\frac{c_{j1}x + c_{j2}y + c_{j3}z}{2\varepsilon} + \frac{(c_{j1}^2 + c_{j2}^2 + c_{j3}^2)t}{4\varepsilon} \right). \tag{41}$$

The center of the shock is where $f_1 = f_2$. We choose this center in the most *unfavorable* way for our grid refinement strategy, i.e., a plane skew through the cube

$$x - y - z = -\tfrac{3}{4}t. \tag{42}$$

This is obtained for $c_{jj} = 1$ and $c_{ij} = \frac{1}{2}$ giving as exact solution for (38)

$$u = \frac{f_1 + \frac{1}{2}f_2}{f_1 + f_2} = 1 - \frac{1}{2} \cdot \frac{1}{1 + \exp[(-x + y + z - \frac{3}{4}t)/(4\varepsilon)]}, \tag{43a}$$

$$v = w = \tfrac{3}{2} - u. \tag{43b}$$

Substituting v and w from (43b) into (38) we get the scalar PDE

$$u_t + uu_x + (\tfrac{3}{2} - u)(u_y + u_z) = \varepsilon \Delta u. \tag{44}$$

The domain is the unit cube. The initial solution and the Dirichlet boundary conditions are given by the exact solution (43). We discuss the results for both the scalar PDE (44) and the coupled system (38) on the time interval [0,1].

3.1.1. *Numerical results for Problem* I(a)

Our first test problem is the scalar Burgers' equation for a Reynolds number of 500 ($\varepsilon = 0.002$). Starting on a base grid with $\Delta x = \Delta y = \Delta z = \frac{1}{10}$ we allow a maximum of four grid levels resulting in a finest cell width of $\frac{1}{80}$. After 92 time steps the endpoint was reached using at each grid level 184 Newton iterations. Since Dirichlet boundary conditions are prescribed we can compute the diagonal-scaling preconditioner in our second linear solver with only four residual evaluations. The accuracy

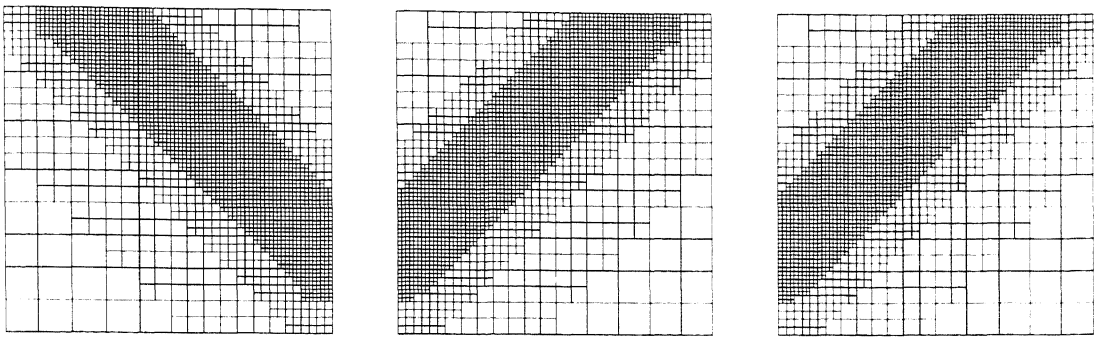

Fig. 1. Grid for Problem I(a) at $t = 1.0$. Slices at $x = 0.5$, $y = 0.5$, and $z = 0.5$.

Table 1
Integration history for Problem I(a)

	BiCGStab+ILU				GCRO+Diag			
Level	1	2	3	4	1	2	3	4
STEPS	92	92	92	92	92	92	92	92
JACS	92	92	92	92	92	92	92	92
NIT	184	184	184	184	184	184	184	184
LSIT 1	92	92	92	92	730	908	1206	1760
LSIT 2	21	84	90	91	453	885	1225	1738
# pts								
$t = 0.001$	1331	5481	22041	100637	1331	5481	22041	100637
$t = 0.6$	1331	7233	33677	164617	1331	7233	33677	164617
$t = 1.0$	1331	6789	31229	145065	1331	6789	31229	145129
Uniform	1331	9261	68921	531441	1331	9261	68921	531441

and grid refinements were alike for both linear solvers. The maximum norm of the error at $t = 1.0$ was 0.07. In Fig. 1 we picture the grid at $t = 1.0$. We show plots of orthogonal slices through the middle of the cube. As expected these slices are alike since the problem is symmetric in each coordinate direction. In Table 1 we list the number of grid points used at each grid level for three different times. The last row contains the number of points needed to reach the same level of refinement with a uniform grid. For the lower grid levels there is not much gain in using the adaptive-grid method. This is because the LUGR method creates a certain buffer in all directions around the wave resulting in refinement almost everywhere for the lower grid levels. However, at the finest grid level the difference in number of grid points is considerable. Predictably, the overhead of the LUGR method is for this problem rather large since it is a scalar PDE and a very small number of iterations is needed to solve the nonlinear system. Even in scalar mode for the BiCGStab solver about 25% of the CPU time is spent in routines dealing with the refinement. For the GCRO solver this is less than 10%, but this solver is less efficient for this problem especially in scalar mode. As can be seen from the results in Table 1 the ILU preconditioner is optimal for this problem.

Table 2
Integration history for Problems I(b) and I(c)

	Problem I(b)				Problem I(c)					
	GCRO+Diag				BiCGStab+ILU			GCRO+Diag		
Level	1	2	3	4	1	2	3	1	2	3
STEPS	78	78	78	78	36	36	36	36	36	36
JACS	78	78	78	78	36	36	36	36	36	36
NIT	156	156	157	217	72	72	88	72	72	78
LSIT 1	659	842	1147	1790	36	36	36	377	533	894
LSIT 2	403	826	1148	1733	31	33	34	361	527	851
LSIT 3				1125			16			84
# pts										
$t = 0.001$	1331	5481	22041	100637	1331	5481	25277	1331	5481	25277
$t = 0.6$	1331	7233	35317	164593	1331	7853	37933	1331	7853	37933
$t = 1.0$	1331	6789	31229	152665	1331	7393	35141	1331	7393	35197
Uniform	1331	9261	68921	531441	1331	9261	68921	1331	9261	68921

3.1.2. *Numerical results for Problem* I(b)

The second test problem is the coupled system (38). To solve a system of three PDEs using four refinement levels with a maximum number of points of approximately 170,000 the BiCGStab solver requires more memory than available on the Cray. The transition layer of the solution is $O(\sqrt{\varepsilon})$ so a cell width of $\frac{1}{80}$ is already rather coarse for a Reynolds number of 500 using central finite differencing especially since the transition layer is placed skew through the cube. Since three refinement levels is too few to solve (38) with $\varepsilon = 0.002$, we only used the matrix-free GCRO solver. The results are in agreement with the scalar example. After 78 time steps the endpoint was reached and the maximum norm of the error at $t = 1.0$ was again 0.07. The difference in the number of time steps is caused by the fact that if the estimated time errors are slightly different this can result in a different time step size for the rest of the interval since we adjust our step size such that the rest of the integration interval is an integer number times Δt. The refinements are equivalent with the picture shown in Fig. 1. Also the number of grid points is alike (cf. Table 2). It should be mentioned that for the finest grid the number of grid points changes a great deal from one time level to the next. E.g., for the scalar Burgers' equation this number was 153133 at $t = 0.99$. It is obvious from the integration history in Table 2 that the nonlinear system is harder to solve especially for the finer grid levels.

3.1.3. *Numerical results for Problem* I(c)

To compare the two nonlinear solvers we decreased the Reynolds number to 200 and solved (38) with a maximum of 3 grid levels resulting in a finest cell width of $\frac{1}{40}$. After 36 time steps the endpoint was reached using about two Newton iterations per time step. The maximum norm of the error at $t = 1.0$ was 0.06. Since the wave front is less sharp than in the previous example more grid points are required on the finer grid levels (cf. Table 2).

Predictably, the overhead is much smaller than for Problem I(a) approximately 1% in scalar mode and approximately 15% in vector mode for the BiCGStab solver. The linear solver BiCGStab+ILU

is again optimal with respect to the number of linear iterations needed but not in computational time. The computation of the $19 \cdot 3$ block-diagonal Jacobian and its decomposition takes more time than the extra linear iterations needed in the second solver using the simplest possible preconditioner, viz., diagonal-scaling that requires only four residual evaluations (see also Table 7 in Section 4.1). One can see from the results in Table 2 that this is still an acceptable preconditioner although more linear iterations per Newton iteration are required than in the scalar Problem I(a).

3.2. *Problem* II: *rotating sphere problem*

Our second example is the advection–reaction system

$$\boldsymbol{c}_t + u\boldsymbol{c}_x + v\boldsymbol{c}_y + w\boldsymbol{c}_z = \boldsymbol{f}(\boldsymbol{c}), \tag{45}$$

where u, v, and w are components of a velocity field in the x-, y-, and z-direction, respectively and the reaction term $\boldsymbol{f}$ consists of the components

$$\begin{aligned} f_1 &= -k_2 c_1 c_2 + k_1 c_2^2, \\ f_2 &= -k_1 c_2^2 + k_2 c_1 c_2. \end{aligned} \tag{46}$$

If we set the constant sum of c_1 and c_2 to d, the solution of (45) with zero velocity is given by

$$\begin{aligned} c_1(t) &= \frac{d}{k_1 + k_2} \frac{k_1(1-\alpha) + (k_1 + k_2)\alpha \, \mathrm{e}^{-dk_2 t}}{1 - \alpha + \alpha \, \mathrm{e}^{-dk_2 t}}, \\ c_2(t) &= d - c_1(t), \end{aligned} \tag{47}$$

with

$$\alpha = \frac{(k_1 + k_2)c_1(0) - dk_1}{dk_2}. \tag{48}$$

For the parameters we selected $k_1 = 1000$, $k_2 = 1$. As initial values we chose $c_1(0) = 0$ and $c_2(0) = d$, so that $\alpha = -k_1/k_2$.

For the advection part we constructed an exact solution which represents a rotating sphere with the highest solution value in its center along an ellipse on a skew plane $y = z$ through the unit cube. This rotation is obtained by choosing the velocities

$$\begin{aligned} u &= 2\pi\sqrt{2}\left(\frac{y+z}{2} - \frac{1}{2}\right), \\ v &= -\pi\sqrt{2}\left(x - \tfrac{1}{2}\right), \\ w &= v. \end{aligned} \tag{49}$$

The exact sphere solution for (45) with $\boldsymbol{f} = 0$ is given by

$$\exp(-80[(x - r(t))^2 + (y - s(t))^2 + (z - s(t))^2]) \tag{50}$$

with

$$r(t) = \tfrac{1}{4}(2 + \sin(2\pi t)), \qquad s(t) = \tfrac{1}{4}(2 + \tfrac{1}{2}\sqrt{2}\cos(2\pi t)). \tag{51}$$

Table 3
Integration history for Problem II

	BiCGStab+ILU				GCRO+Block-Diag				GCRO+Diag			
Level	1	2	3	4	1	2	3	4	1	2	3	4
STEPS	335	335	335	335	335	335	335	335	343	343	343	343
JACS	335	335	335	335	335	335	335	335	348	348	347	345
NIT	670	670	670	671	670	670	670	671	695	693	691	688
LSIT 1	335	335	335	335	2575	2807	3397	4148	3312	4029	6612	6231
LSIT 2	16	26	37	45	1322	1404	1693	2431	1577	2130	3647	3274
Avg. # pts	1331	4500	13700	35800	1331	4500	13700	35800	1331	4500	13700	35800
Uniform	1331	9261	68921	531441	1331	9261	68921	531441	1331	9261	68921	531441

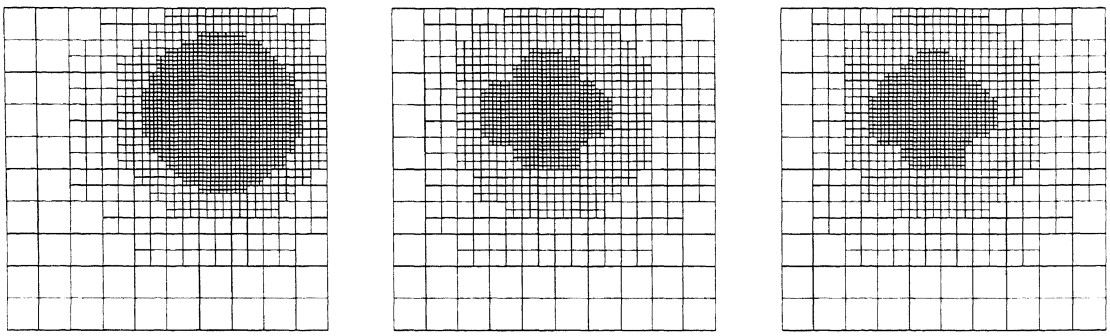

Fig. 2. Grid for Problem II at $t = 1.0$. Slices at $x = 0.5$, $y = 0.5$, and $z = 0.5$.

Summarizing, our second test problem is given by (45) with the velocity field from (49) and the reaction term f from (46). At the inflow boundaries we impose the exact solution. The solution is given by (47) with $d = d(x, y, z, t)$ defined by (50) and $c_1(0) = 0$, resulting in

$$c_1(t) = d(x, y, z, t)\frac{1 - e^{-d(x,y,z,t)\cdot t}}{\frac{1001}{1000} - e^{-d(x,y,z,t)\cdot t}},$$
$$c_2(t) = d(x, y, z, t)\frac{10^{-3}}{\frac{1001}{1000} - e^{-d(x,y,z,t)\cdot t}}. \tag{52}$$

In the short time interval $[0, 0.01]$ the solution c_2 decreases over 90%. For the rest of the time interval, (45) is rather an advection problem with the extra difficulty that for negative solution values (45) becomes unstable.

3.2.1. Numerical results for Problem II

For this test example we used a base grid with eleven points in each direction. A maximum of four grid levels was allowed. The initial time step size was 0.00001.

In the first run we encountered instabilities due to negative solution values causing a break-down of the run. We then adjusted the problem so that after every time step negative solution values were replaced by zero. With this simple adjustment, resulting in the same error values at $t = 0.5$, the endpoint was reached. The error at $t = 1.0$ was the same as obtained for the advection equation

($f = 0$), viz., 0.17 for the first component (the maximum value was 0.83 instead of 1.0). The only difference in integration history between the two runs with and without reaction term was the size of the time steps used in the very first part of the time interval [0, 0.03]. The advection run used over the whole interval step sizes of 0.003, whereas the run solving (45) used step sizes from 0.00001 increasing to 0.003 within [0,0.03]. The integration performance over the rest of the interval was the same. In this case the efficiency gain in memory of the LUGR method is much larger than in the previous case. This is as expected since the refinement area is now a sphere instead of a skew plane (cf. Table 3). The plot in Fig. 2 of the grid at $t = 1.0$ shows that the sphere is approximated well by the grid refinements.

As can be seen in Table 3 the ILU preconditioner again is optimal in terms of linear iterations required. For the second linear solver, GCRO, we first used the simple diagonal scaling neglecting the first-order derivatives at the outflow boundaries. For this solver the problem was a bit more difficult. Newton failures occurred five times resulting in a first step size of 0.4E–7, but after the initial problems there were no more Newton failures and the time step size again increased to 0.003 within [0,0.03]. We then run the problem with the block-diagonal preconditioner also neglecting the first-order derivatives. Although both the computation of this preconditioner and the backsolves are more expensive then the diagonal scaling it pays off. The integration history is as smooth as for BiCGStab+ILU and much less linear iterations are needed. For more detailed information we refer to Table 3.

3.3. *Problem* III: *a 3D fluid-flow/salt-transport problem*

We consider a model for an isothermal, single-phase, two-component saturated flow problem which consists of two PDEs basic to ground-water flow: the continuity equation and the transport equation. For the background of these equations we refer to [19]. We here present the model in non-conservative form. As independent variables we have the pressure p and the salt mass fraction ω. The continuity equation for the fluid and the salt transport equation are given by

$$n\rho\left(\beta\frac{\partial p}{\partial t} + \gamma\frac{\partial \omega}{\partial t}\right) + \nabla\cdot(\rho \boldsymbol{q}) = 0, \tag{53a}$$

$$n\rho\frac{\partial \omega}{\partial t} + \rho \boldsymbol{q}\cdot\nabla\omega + \nabla\cdot(\rho \boldsymbol{J}^{\omega}) = 0, \tag{53b}$$

where n is the porosity parameter of the porous medium, β a compressibility coefficient, and γ a salt coefficient. Darcy's law gives the equation for the fluid velocity $\boldsymbol{q} = (q_1, q_2, q_3)^{\mathrm{T}}$

$$\boldsymbol{q} = -\frac{k}{\mu}(\nabla p - \rho \boldsymbol{g}), \tag{54}$$

with $\boldsymbol{g}$ the acceleration of gravity vector and k the permeability coefficient of the porous medium. The density ρ and the viscosity μ obey the state equations

$$\rho = \rho_0 \exp[\beta p + \gamma\omega], \tag{55}$$

$$\mu = \mu_0 \cdot m(\omega), \quad m(\omega) = 1 + 1.85\omega - 4.1\omega^2 + 44.5\omega^3, \tag{56}$$

where ρ_0 is the reference density of fresh water and μ_0 a reference viscosity. The equation for the salt-dispersion flux vector is given by Fick's law

$$J^{\omega} = -nD\nabla\omega, \tag{57}$$

with the dispersion tensor D for the solute salt defined as

$$nD = (nD_{\mathrm{mol}} + \alpha_{\mathrm{T}}|q|)I + \frac{\alpha_{\mathrm{L}} - \alpha_{\mathrm{T}}}{|q|}qq^{\mathrm{T}}, \quad |q| = \sqrt{q^{\mathrm{T}}q}. \tag{58}$$

The coefficients D_{mol}, α_{T} and α_{L} correspond with the molecular diffusion, the transversal dispersion, and the longitudinal dispersion, respectively.

Our examples are connected with laboratory experiments that deal with the displacement of fresh water by a polluting fluid in a tank filled with a porous medium. Fresh water is flowing from left to right through the tank. The polluting fluid, which has a higher density than fresh water, is injected with constant velocity through a slit at the top of the tank. We simulate here a pollution with salt water of two different concentrations, one with a salt mass fraction of 0.0935, and the other a pollution with brine having a salt mass fraction of 0.25. The latter is more demanding. In both cases this gives rise to a fresh–salt water plume, but the steepness of the front is dependent of the salt mass fraction.

The tank is defined by a box $\Omega = \{(x, y, z) \mid 0 \leqslant x \leqslant 2.5, 0 \leqslant y \leqslant 0.5, 0 \leqslant z \leqslant 1.0\}$. The acceleration of gravity vector takes the form $g = (0, 0, g)^{\mathrm{T}}$, where g is the gravity constant. The initial values at $t = 0$ at $\Omega \cup \partial\Omega$ are taken as

$$p(x, y, z, 0) = (0.03 - 0.012x + 1 - z)\rho_0 g, \qquad \omega(x, y, z, 0) = 0. \tag{59}$$

For $0 < t \leqslant t_{\mathrm{end}}$ the following boundary conditions are imposed

$$\begin{array}{llll}
x = 0, & y \in [0, 0.5], & z \in [0, 1]: & p = p(x, y, z, 0), \quad \omega = 0, \\
x = 2.5, & y \in [0, 0.5], & z \in [0, 1]: & p = p(x, y, z, 0), \quad \omega_x = 0, \\
y = 0, 1, & x \in [0, 2.5], & z \in [0, 1]: & q_2 = 0, \quad \omega_y = 0, \\
z = 0, & x \in [0, 2.5], & y \in [0, 0.5]: & q_3 = 0, \quad \omega_z = 0, \\
z = 1, & x, y \notin B: & & q_3 = 0, \quad \omega_z = 0, \\
 & (x, y) \in B: & & \rho q_3 = -4.95_{10^{-2}}, \quad \omega = \omega_B,
\end{array} \tag{60}$$

where

$$B = \{(x, y) \mid 0.375 \leqslant x \leqslant 0.4,\ 0.2 \leqslant y \leqslant 0.3\}.$$

The last line is connected with the slit where the salt water is injected with a prescribed velocity and concentration. In the first experiment the salt mass fraction $\omega_B = 0.0935$, in the second $\omega_B = 0.25$. The other conditions are self-evident. All remaining problem data are contained in Table 4.

3.3.1. Numerical results for Problem III(a) ($\omega_B = 0.0935$)

As cell width of the base grid we used 0.1 in each direction. The maximum number of grid levels allowed was four. For this problem we set UMAX(1) = $\rho_0 g = 10000$ and UMAX(2) = 0.1. In Fig. 3 we give contour plots of the salt concentration in the plane $y = 0.2$ after 2 and 6 hours. The contour lines of the higher salt mass fractions (0.5–1.0 times ω_B) display a likely solution pattern and show no

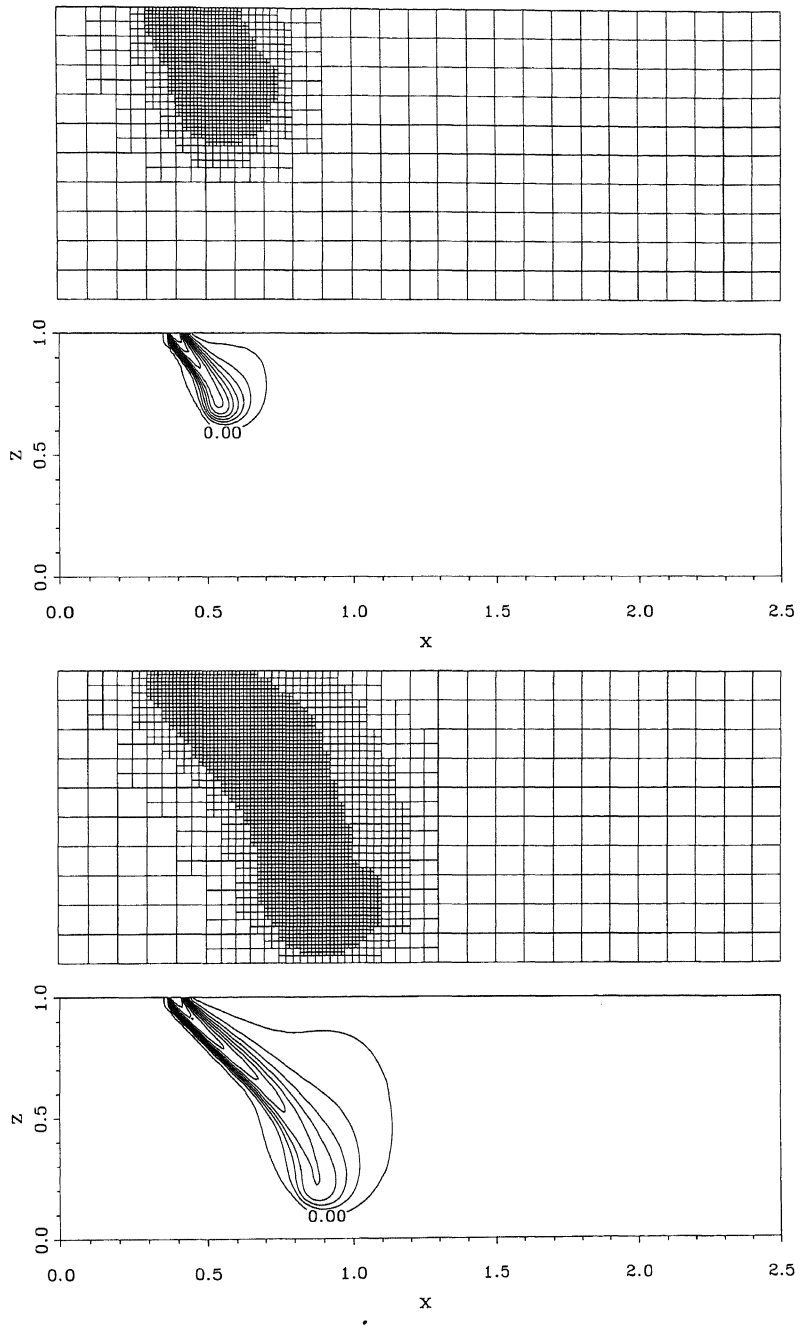

Fig. 3. Problem III(a) ($\omega_B = 0.0935$), slice at $y = 0.2$. Grid and 10% contour lines of the salt mass fraction after 2 hours (above) and after 6 hours (below).

Table 4
Data for Problem III

n	0.35	k	7.18_{10-11}	nD_{mol}	1_{10-9}
ρ_0	1000	μ_0	0.001	α_T	0.001
γ	ln(2)	g	10	α_L	0.01
β	0.0	t_{end}	2_{10^4}		

Table 5
Integration history for Problem III(a)

		BiCGStab+ILU				GCRO+Block-diag			
	Level	1	2	3	4	1	2	3	4
$t = 7200$	STEPS	101	101	101	101	115	115	115	115
	JACS	103	103	103	105	126	125	125	135
	NIT	203	205	206	211	250	250	253	254
	LSIT 1	323	268	359	441	4012	4944	6634	9313
	LSIT 2	148	109	186	302	3313	2946	3682	6072
	LSIT 3							106	466
	# pts	1716	2211	7817	25647	1716	2211	7817	25647
$t = 21600$	STEPS	168	168	168	168	182	182	182	182
	JACS	170	170	170	172	193	192	193	202
	NIT	337	339	340	345	384	384	399	392
	LSIT 1	577	494	631	759	7013	8442	11806	16016
	LSIT 2	267	235	360	632	5295	4798	6682	9969
	LSIT 3							556	571
	LSIT 4							187	
	# pts	1716	4675	20419	66527	1716	4675	20419	66527
	Uniform	1716	11781	86961	667521	1716	11781	86961	667521

wiggles despite the steepness of the solution near the inlet slit. However, for the lower concentrations there is an unexpected "drip" in the solution. Since the density decreases from top to bottom, one would expect that the pollutant would be more easier taken by the flow instead of sinking to the bottom. It is not likely that this is caused by the grid refinement procedure, since the refined grids are nicely placed around the fresh–salt water plume. Moreover, a second run on a uniform grid with a cell width of 0.025 gave a solution with the same characteristics.

The integration data are given in Table 5. Using BiCGStab+ILU as linear solver the integration in time is again performed smoothly, the step size steadily increases from approximately 1 at the start of the problem to approximately 200 at t_{end}. Both solvers give the same solution up to plotting accuracy. In this case we could not neglect the boundary conditions in the preconditioner for GCRO. This was to be expected since the boundary conditions at the inlet do include first-order derivatives. With block-diagonal preconditioning the second solver behaved reasonably smooth in time. Ten Newton failures occurred at about $t = 3300$, but in the remainder of the time interval the step size steadily increased to 260 at t_{end}. Restriction to diagonal scaling led to too many failures of the linear solver to be of practical use. For this problem BiCGStab+ILU is by far the more efficient in CPU time, but if

Table 6
Integration history for Problem III(b)

	Level	1	2	3	4	5
$t = 7200$	STEPS	178	178	178	178	178
	JACS	185	184	184	184	198
	NIT	369	368	367	379	376
	LSIT 1	457	431	556	698	970
	LSIT 2	172	195	275	440	896
	LSIT 3					92
	# pts	1716	3157	10237	32541	100799
$t = 14400$	STEPS	243	243	243	243	243
	JACS	261	260	266	257	282
	NIT	521	520	526	525	558
	LSIT 1	739	689	837	991	2121
	LSIT 2	286	309	455	674	1432
	LSIT 3					223
	LSIT 4					97
	LSIT 5					63
	# pts	1716	4059	16063	52715	145394
	Uniform	1716	11781	86961	667521	5229441

memory is the bottle-neck the problem can be solved with the matrix-free solver. It should be noted that this solver is less robust. Since the quality of the stopping criterion (26) is dependent on the correctness of the approximation of the Jacobian G by the preconditioner P it is important to have a good preconditioner. This means for the second linear solver that the number of iterations in the inner GMRES loop should not be too small so that the system is well enough solved.

Note that the solution of the linear systems is more demanding in this case. Therefore the LUGR approach will be from a computational point of view more profitable than in the previous examples.

3.3.2. Numerical results for Problem III(b) ($\omega_B = 0.25$)

We started with the same base grid as in the previous case. A fifth grid level was required to solve the brine transport problem since a higher salt mass fraction results in steeper fronts. UMAX was set to (10000,0.25). The contour plots in Fig. 4 show again a slice from the solution at $y = 0.2$. As expected, a pollutant with higher density will sink faster to the bottom, which is reached after four hours. For this problem we present only the results for the BiCGStab+ILU solver. The figures for the integration history are given in Table 6. They show even more strikingly than in the previous case, that the solution of the (non)linear systems at a fine grid appears to be more cumbersome than at the coarser grids. This could probably be explained by near-singularities caused by the hydrodynamic dispersion tensor in the vicinity of vortices [14].

It would not be feasible to solve Problems III(a) or III(b) on a uniform fine grid. However, knowing the mass fraction of a pollutant and the pressure gradient between $x = 0$ and $x = 2.5$, one could predict on forehand a region where the polluting fluid would spread and use a nonuniform grid with refinements in that specific region.

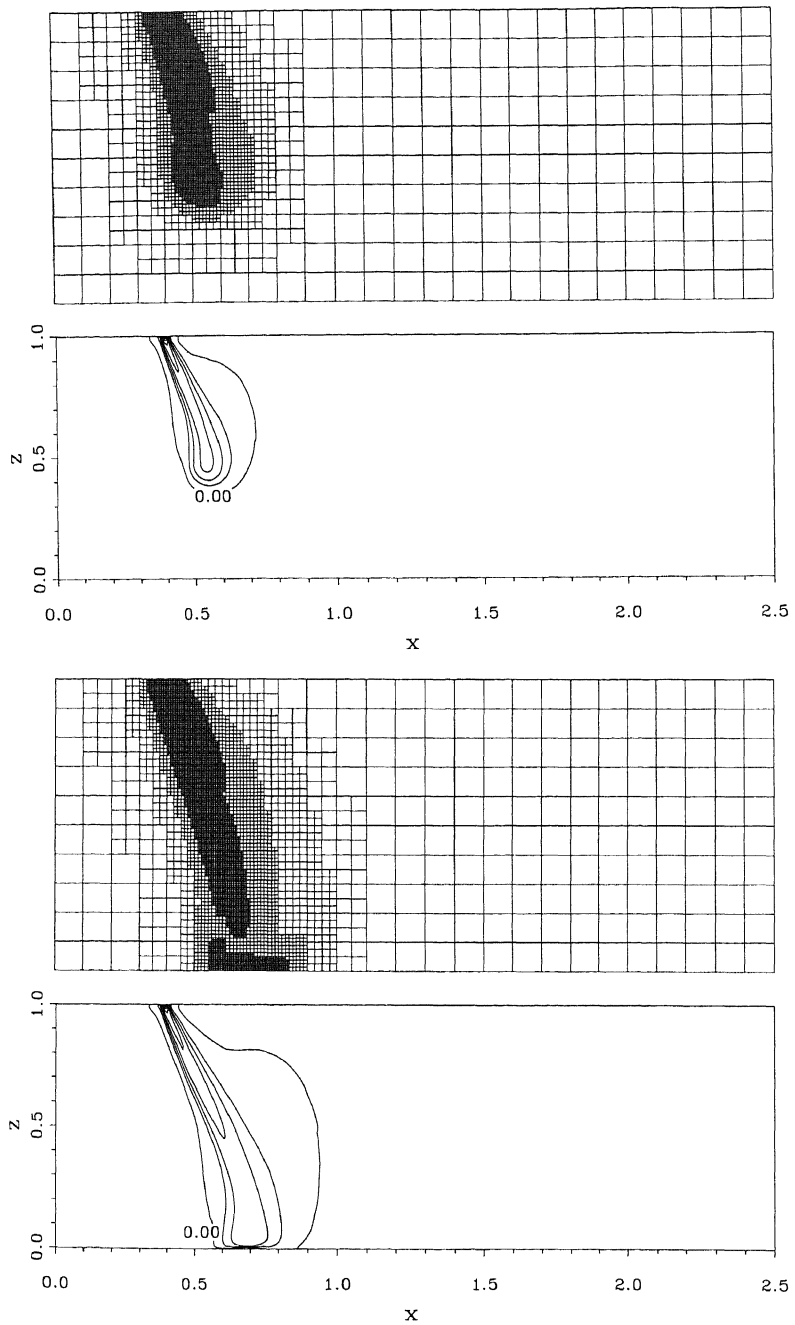

Fig. 4. Problem III(b) ($\omega_B = 0.25$), slice at $y = 0.2$. Grid and 10% contour lines of the salt mass fraction after 2 hours (above) and after 4 hours (below).

Table 7
Global performance

		BiCGStab		GCRO		
		CP sec	Mflop	CP sec	Mflop	Prec
Problem I(a)	scalar	1263	12	2446	22	Diag
	vector	430	36	509	104	Diag
Problem I(b)	vector			1260	160	Diag
Problem I(c)	scalar	1439	10	857	24	Diag
	vector	175	79	132	154	Diag
Problem II	vector	873	63	787	126	Diag
	vector			624	105	Block
Problem III(a)	vector	758	119	2854	195	Block
Problem III(b)	vector	4222	117			

4. Performance

Our performance evaluation was done on a Cray YMP with the CF77 compiling system. Scalar results were obtained using `cf77 -Wf" -o novector"` and vector results with `cf77 -Zv -Wf"-o aggress"`. To measure the Megaflop rate and the CPU time spent in a routine we used the Cray utility Perftrace [11] that gives the hardware performance by program unit (compiler flags `-F` and loader flags `-F -lperf`).

4.1. Global performance

We first give a global idea of the performance of the two different solvers, i.e., `VLUGR3`+BiCGStab and `VLUGR3`+GCRO. In Table 7 the CPU time and the Mflop rate is shown for the example problems.

BiCGStab+ILU is the more robust solver and uses the least number of linear iterations. Therefore, if memory is not a problem it is in most cases more efficient, especially on scalar processors. If the number of PDE components is large or if (block-)diagonal scaling is a good preconditioner, then GCRO can be competitive or even faster. This matrix-free solver is more memory efficient and vectorizes also much better, but since it is more sensitive to stopping criteria and restrictions on the number of iterations we feel that it should be used with care.

4.2. Vector results

In this section we discuss the vector performance of the LUGR code for the three example problems. The timings were done on one processor of a Cray Y-MP which has a clock cycle time of 6 ns. This gives a theoretical peak performance on one processor of 167 Mflops and 333 when chaining an add and a multiplication. Since during one cycle time one store and two loads can be performed, indirect addressing of one of the vector operands of a triad will reduce the performance at least with a factor of 2, bank conflicts left out of consideration. When more vectors are indirectly addressed

Table 8
Vector performance of top 5 routines for Problem I(a)

	BiCGStab+ILU					GCRO+Diag			
	# calls	Avg. time	ACM %	Mflop		# calls	Avg. time	ACM %	Mflop
INJON	816	7.8E-2	14.8	0	DERIVS	9641	1.4E-2	26.4	137
ILU backs	2044	2.2E-2	25.3	69	GMRESO	1393	6.0E-2	42.9	235
JACSLP	277	1.5E-1	35.0	0	INJON	816	7.8E-2	55.4	0
JACSUP	277	1.4E-1	44.3	0	MKBND	276	1.3E-1	62.5	0
MKBND	276	1.3E-1	52.6	0	SETBA	277	1.3E-1	69.5	0

Table 9
Vector performance of top 5 routines for Problem I(b)

	GCRO+Diag			
	# calls	Avg. time	ACM %	Mflop
DERIVS	10374	4.7E-02	38.5	145
GMRESO	1294	2.6E-01	64.9	251
PDEF	11622	1.0E-02	74.4	215
INJON	690	9.5E-02	79.6	0
MATVEC	9688	5.2E-03	83.6	183

Table 10
Vector performance of top 5 routines for Problem I(c)

	BiCGStab+ILU					GCRO+Diag			
	# calls	Avg. time	ACM %	Mflop		# calls	Avg. time	ACM %	Mflop
ILU dec	108	5.8E-1	36.0	65	DERIVS	3849	1.3E-2	37.7	137
ILU backs	676	6.0E-2	59.4	78	GMRESO	430	8.3E-2	64.8	245
MATVEC	444	3.8E-2	69.2	143	PDEF	4281	2.7E-3	73.7	210
PDEF	3472	2.0E-3	73.2	213	INJON	208	3.3E-2	78.9	0
INJON	208	3.2E-2	77.1	0	MATVEC	3627	1.4E-3	82.7	176

the (current) impossibility on the Y-MP to chain more than one gathered/scattered load/store would reduce the performance to a much larger degree.

On all problems the vector performance of the LUGR code using the matrix-free GCRO+(block-) diagonal preconditioning is satisfactory or even good. For more difficult problems a vector speed of approximately 200 Mflops is reached. For Problem I(a) and to a lesser extent also for Problem II the code with BiCGStab+ILU almost behaves as a scalar code. Closer inspection shows that all routines that deal with the solution of the PDE system on *one specific grid* have a satisfactory vector performance. The definition of the PDE achieves 150–250 Mflops, the matrix–vector multiplication about 150 and the preconditioner approximately 70 Mflops. The disappointing overall vector performance for the first two problems is caused by the grid refinement "overhead" routines that contain

Table 11
Vector performance of top 5 routines for Problem II for solvers BiCGStab+ILU and GCRO+Diag

	BiCGStab+ILU					GCRO+Diag			
	# calls	Avg. time	ACM %	Mflop		# calls	Avg. time	ACM %	Mflop
ILU dec	1340	1.6E–1	24.6	62	DERIVS	33583	7.0E–3	30.2	137
ILU backs	5611	2.4E–2	40.2	69	GMRESO	4397	3.5E–2	50.0	239
INJON	3003	2.5E–2	48.9	0	INJON	3084	2.6E–2	60.2	0
MATVEC	2930	1.5E–2	54.0	140	PDEF	39133	1.7E–3	68.6	240
JACSLP	1006	4.3E–2	59.0	0	MKBND	1038	3.9E–2	73.7	0

Table 12
Vector performance of top 5 routines for Problem II for solver GCRO+Block-diag

	GCRO+Block-diag			
	# calls	Avg. time	ACM %	Mflop
DERIVS	22460	7.0E–03	25.5	136
INJON	3003	2.6E–02	37.8	0
GMRESO	4148	1.7E–02	49.4	226
PDEF	33180	1.6E–03	58.1	240
MKBND	1005	3.9E–02	64.4	0

Table 13
Vector performance of top 5 routines for Problem III(a)

	BiCGStab+ILU					GCRO+Block-diag			
	# calls	Avg. time	ACM %	Mflop		# calls	Avg. time	ACM %	Mflop
PDEF	15006	1.6E–2	31.4	210	PDEF	83865	2.1E–2	62.9	209
ILU backs	9340	2.0E–2	56.5	72	GMRESO	4150	1.0E–1	77.7	254
MATVEC	7972	1.3E–2	70.6	142	DERIVS	72957	5.4E–3	91.7	134
ILU dec	682	1.0E–1	80.1	62	BCKBDI	72957	6.6E–4	93.4	113
INJON	1518	1.6E–2	83.3	0	MATVEC	71364	5.8E–4	94.8	175

Table 14
Vector performance of top 5 routines for Problem III(b)

	BiCGStab+ILU			
	# calls	Avg. time	ACM %	Mflop
ILU backs	21071	6.6E–2	33.1	73
PDEF	29277	3.6E–2	57.9	210
MATVEC	18282	4.4E–2	77.2	142
ILU dec	1326	2.3E–1	84.2	66
INJON	3064	3.3E–2	86.7	0

no floating-point operations and that take a considerable amount of the total CPU time. In the case of the scalar Problem I(a) this amounts to even 70% of the total CPU time. For the two-component system of Problem II the overhead still amounts to 35%. In Tables 8–14 we can see that the most time consuming "search" routine is INJON which transfers the solution values of previous time levels to the grid at a current time level. The time spent in this routine could be diminished by adding pointers between coinciding grid points from one grid to another during creation of the new grid. In the current implementation one has to search for coinciding grid points in both grids. It is also possible to use a one-step time-integration formula instead of the two-step BDF formula. This would approximately halve the time used to transfer solution values of previous time levels to the current grid. The less complicated way of defining the boundary in 3D compared to the structure in 2D in VLUGR2 [3] has its consequences when those structures are used (as in JACSLP and JACSUP which create the dependency lists for the vectorizable version of the preconditioner) or created (MKBND). However one should keep in mind that these routines are less dominant when the problem is more nonlinear or when the number of PDE components is larger so that more work has to be done to *solve* the PDE-system on a specific grid (cf. Tables 13 and 14). For Problem III(a) the overhead costs are approximately 10% and for Problem III(b) even less than 5%.

5. Summary

We have discussed the performance of a Method of Lines solver based on a Local Uniform Grid Refinement method for systems of time-dependent PDEs in three dimensions. The vectorizable implementation of this method is an extension of the 2D code VLUGR2 [3]. The LUGR method proved to be robust and efficient with respect to the location of the refined grids, especially when a very fine grid was required in part of the domain. The experiments with VLUGR2 [3] revealed that in two dimensions the overhead for the grid refinement is negligible when using an implicit time integrator even for simple scalar problems. In three space dimensions the local refinement overhead is larger and can even dominate the CPU time for simple problems. It is possible to decrease this overhead by adding extra pointers to the data structure or by using a one-step time-integration formula. However, if the number of components is large or the nonlinearity of the problem is high (as for instance in Problem III) the CPU time needed for the grid structure and refinement is again negligible.

Our LUGR code offers two different iterative nonlinear solvers. In the first the Jacobian of the Newton process is approximated and stored. The linear solver used is BiCGStab [20] combined with standard ILU preconditioning. The second solver is a matrix-free Newton process and uses GCRO [12] and (block-)diagonal scaling. If memory demands are no restriction we advocate the former since it is more robust and generally uses less CPU time. The matrix-free solver not only avoids the storage of Jacobian and ILU matrices, it also generates a true Newton process instead of a modified one. An additional advantage is that tailor-made space-discretization schemes, resulting in other couplings than the here used 19-point stencil, can be more easily implemented. The vector performance of this solver is good, from 100 Mflops for a simple scalar PDE to 200 Mflops for a more nonlinear system of PDEs.

The first linear solver is less efficiently vectorizable but uses also in general less CPUtime. For simple scalar problems the CPU time is dominated by the grid refinement overhead which is for this solver even larger than for the matrix-free one. It will of course still be faster than solving

on a uniform fine grid. The ILU preconditioning of the linear systems vectorizes better than in the two-dimensional case, since the number of computations that can be done independently from each other is much larger. However, the Mflop rate is still hampered by the fact that only one indirect load/store instruction can be issued at a time, versus two load and one store instructions without gather/scatter necessity (cf. [4]).

Acknowledgement

We wish to thank Mart Oostrom and Jan van Eijkeren (RIVM) for providing us the data for Problem III.

References

[1] C.C. Ashcraft and R.G. Grimes, On vectorizing incomplete factorization and SSOR preconditioners, *SIAM J. Sci. Statist. Comput.* **9** (1) (1988) 122–151.

[2] J.G. Blom and J.G. Verwer, A vectorizable adaptive grid solver for PDEs in 3D, Report NM-R9319, CWI, Amsterdam (1993).

[3] J.G. Blom and J.G. Verwer, VLUGR2: a vectorized local uniform grid refinement code for PDEs in 2D, Report NM-R9306, CWI, Amsterdam (1993).

[4] J.G. Blom and J.G. Verwer, Vectorizing matrix operations arising from PDE discretization on 9-point stencils, *J. Supercomput.* **8** (1994) 29–51.

[5] J.G. Blom and J.G. Verwer, VLUGR3: a vectorizable adaptive grid solver for PDEs in 3D. II. Code description, Report NM-R9405, CWI, Amsterdam (1994).

[6] J.G. Blom, J.G. Verwer and R.A. Trompert, A comparison between direct and iterative methods to solve the linear systems arising from a time-dependent 2D groundwater flow model, *Internat. J. Comput. Fluid Dynamics* **1** (1993) 95–113.

[7] K.E. Brenan, S.L. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* (North-Holland, New York, 1989).

[8] P.N. Brown, A.C. Hindmarsh and L.R. Petzold, Using Krylov methods in the solution of large-scale differential-algebraic systems, Technical Report TR 93-37, Computer Science Department, University of Minnesota, Minneapolis, MN (1993).

[9] P.N. Brown and Y. Saad, Hybrid Krylov methods for nonlinear systems of equations, *SIAM J. Sci. Statist. Comput.* **11** (1) (1990) 450–481.

[10] A.T. Chronopoulos, Nonlinear CG-like iterative methods, Report UMSI 91/99, University of Minnesota Supercomputer Institute, Minneapolis, MN (1991).

[11] Cray Research, Inc., *UNICOS Performance Utilities Reference Manual*, SR-2040 6.0 edition (1991).

[12] E. De Sturler and D.R. Fokkema, Nested Krylov methods and preserving the orthogonality, In N. Duane Melson, T.A. Manteuffel and S.F. McCormick, eds., *Sixth Copper Mountain Conference on Multigrid Methods*, NASA Conference Publication **3324**, Part 1 (1993) 111–126.

[13] R.A. Trompert, MOORKOP, an adaptive grid code for initial-boundary value problems in two space dimensions, Report NM-N9201, CWI, Amsterdam (1992).

[14] R.A. Trompert, A note on singularities caused by the hydrodynamic dispersion tensor, Report NM-R9302, CWI, Amsterdam (1993).

[15] R.A. Trompert, Local Uniform Grid Refinement for time-dependent partial differential equations, Ph.D. Thesis, University of Amsterdam (1994).

[16] R.A. Trompert and J.G. Verwer, A static-regridding method for two-dimensional parabolic partial differential equations, *Appl. Numer. Math.* **8** (1991) 65–90.

[17] R.A. Trompert and J.G. Verwer, Analysis of the implicit Euler local uniform grid refinement method, *SIAM J. Sci. Comput.* **14** (1993) 259–278.
[18] R.A. Trompert and J.G. Verwer, Runge-Kutta methods and local uniform grid refinement, *Math. Comp.* **60** (1993) 591–616.
[19] R.A. Trompert, J.G. Verwer and J.G. Blom, Computing brine transport in porous media with an adaptive-grid method, *Internat. J. Numer. Methods Fluids* **16** (1993) 43–63.
[20] H.A. van der Vorst, BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* **13** (2) (1992) 631–644.
[21] H.A. van der Vorst and C. Vuik, GMRESR: a family of nested GMRES methods, Report 91-80, Faculty of Technical Mathematics and Informatics, TU Delft, Netherlands (1991).
[22] J.G. Verwer and R.A. Trompert, An adaptive-grid finite-difference method for time-dependent partial differential equations, in: D.F. Griffiths and G.A. Watson, eds., *Proceedings 14th Biennial Dundee Conference on Numerical Analysis*, Pitman Research Notes in Mathematics Series **260** (Pitman, London, 1992) 267–284.
[23] J.G. Verwer and R.A. Trompert, Analysis of local uniform grid refinement, *Appl. Numer. Math.* **13** (1993) 251–270.
[24] G.B. Whitham, *Linear and Nonlinear Waves* (Wiley, New York, 1974).