Centrum voor Wiskunde en Informatica
**REPORT***RAPPORT*

# A vectorizable adaptive grid solver for PDEs in 3D

J.G. Blom and J.G. Verwer

Department of Numerical Mathematics

**NM-R9319 1993**

Report NM-R9319
ISSN 0169-0388

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

# A Vectorizable Adaptive Grid Solver for PDEs in 3D

J.G. Blom, J.G. Verwer

*CWI*

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

**Abstract**

This paper describes the application of an adaptive-grid finite-difference solver to some time-dependent three-dimensional systems of partial differential equations. The code is a 3D extension of the 2D code VLUGR2[3].

*AMS Subject Classification (1991):* Primary: 65M20. Secondary: 65M50, 65Y20.
*CR Subject Classification (1991):* G.1.8, G.4, J.2
*Keywords & Phrases:* software, application in physical sciences, partial differential equations, method of lines, adaptive-grid methods, nonsymmetric sparse linear systems, iterative solvers, vectorization.
*Note:* This work was supported by Cray Research, Inc., under grant CRG 93.03, via the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF).

## 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 ODEs is solved in time with a robust 'off-the-shelf' ODE solver. The spatial grid used can be either uniform or nonuniform but is invariable for the time interval. For problems with steep and moving fronts this can be an inefficient approach since too much grid points are needed at areas where the solution (no longer) has 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 statically (e.g., after each time step). This introduces of course always a certain amount of overhead but when the use of implicit ODE solvers is profitable or when the number of PDE components is very large such solvers can be more efficient already in two dimensions. In three dimensions the standard approach often will no longer be feasible, not only because of the inefficiency of the computations, but even more so because the physical memory of most computers will not be large enough to contain the solution values at all grid points and, e.g., the Jacobian needed for the solution of the nonlinear system.

In [8, 9, 10, 12, 11, 13] an adaptive-grid finite-difference method was developed to solve time-dependent two-dimensional systems of PDEs. 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. In [3] the vectorized implementation of this method in the code VLUGR2 is described (VLUGR2 is very similar to Trompert's code MOORKOP[6]). We have extended this method to the 3D case and at this moment the first version of our aimed vectorizable adaptive-grid finite-difference code for three-dimensional time-dependent PDEs with spatial differential operators of order 2 is available. The code has been designed to solve initial boundary value problems for systems of partial differential equations that fit in the following master equation

$$\mathcal{F}(t, x, y, z, \mathbf{u}, \mathbf{u}_t, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z, \mathbf{u}_{xx}, \mathbf{u}_{yy}, \mathbf{u}_{zz}, \mathbf{u}_{xy}, \mathbf{u}_{xz}, \mathbf{u}_{yz}) = 0, \quad (x, y, z) \in \Omega, \quad t > t_0, \tag{1.1}$$

where the solution $\mathbf{u}$ may be a vector and the domain $\Omega$ an arbitrary domain that can be described by a, possibly disjunct, 'brick' structure. The boundary conditions belonging to system (1.1) are formulated as

$$\mathcal{B}(t, x, y, z, \mathbf{u}, \mathbf{u}_t, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z) = 0, \quad (x, y, z) \in \partial\Omega, \quad t > t_0, \tag{1.2}$$

and the initial conditions satisfy

$$\mathbf{u}(t_0, x, y, z) = \mathbf{u}_0(x, y, z), \quad (x, y, z) \in \Omega \cup \partial\Omega. \tag{1.3}$$

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

- Burgers' equation with as solution a wave moving skew through a cube with a hole,
- a rotating sphere problem described by a system of two advection-reaction equations, and
- a model for the simulation of pollution in groundwater flow.

The tests showed that, as in the 2D case, the *location* of the grid refinements proved to be efficient and robust. However, the implementation, which is a straightforward extension of the 2D code, could be improved. In 3D the overhead of the grid refinement increases, due to our choice of the datastructure. This could be solved by adding extra pointers to the datastructure. However, more important is that in the current implementation, even using adaptive-grid methods, memory still will be the limiting factor for real-life applications. This is mainly due to the (non)linear system solver used. In the current implementation the nonlinear systems are solved with modified Newton and the linear systems with BiCGStab[14] with ILU preconditioning which is a robust solver but lavish of memory.

The paper is organized as follows. In Section 2 we give a short survey of the LUGR algorithm and the implementation. Section 3 contains the description of the three problems. In this section we also look into the robustness and the efficiency of the solver on the basis of some numerical experiments. In Section 4 we discuss the vector performance of the solver for these three examples. Finally, Section 5 contains a summary of our findings and some future plans.

## 2. Outline of the LUGR algorithm

### *2.1 Static regridding*

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 a MoL 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 IBVP on the current grid with the current time step
2. If the required resolution in space is not yet reached:
   (a) Determine at forward time level a new, embedded, grid of next finer grid level

(b) Get solution values at previous time level(s) on the new grid

(c) Interpolate internal boundary values from old grid at forward time

(d) Get initial values for the Newton process at forward time

(e) Goto 1.

3. Inject fine grid solution values in coinciding coarser grid nodes

4. Estimate error in time-integration. If time error is acceptable

   - Advance time level

5. Determine new step size, goto 1 with coarse base grid as current grid.

For the time integration we use the second-order two-step implicit BDF method with variable step sizes. The resulting system of nonlinear equations is solved with modified Newton and (preconditioned) iterative linear solvers. For the space discretization standard second-order finite differences are used, central on the internal domain and one-sided at the boundaries. Where grid refinement is required we divide a grid cell in 8 equal parts, so we do not apply semi-refinement, i.e., refinement in just one or two directions parallel to an axis. Where interpolation is needed to get solution values, linear interpolation is used.

Most strategic choices have been copied from the 2D code `VLUGR2`[3]. Here again the refinement strategy is based on the curvature monitor in each grid point $(i,j,k)$

$$|\Delta x^2.u^{ic}_{xx}(i,j,k)| + |\Delta y^2.u^{ic}_{yy}(i,j,k)| + |\Delta z^2.u^{ic}_{zz}(i,j,k)|, \quad ic = 1, \texttt{NPDE}, \tag{2.4}$$

where $\Delta x$, $\Delta y$, and $\Delta z$ are the grid width in the $x$-, respectively, the $y$-, respectively, the $z$-direction.

*2.2 Implementation*

The first version of our aimed vectorizable adaptive-grid finite-difference code for 3D time-dependent PDEs (1.1) is a more or less straightforward extension of `VLUGR2`[3].

The solution at a specific grid level is stored along planes and each plane is stored row-wise, one component vector after the other. Solutions from 3 different time levels have to be saved. For the oldest time level only the injected solution values for the computation of $\mathbf{U}_t$ with the BDF method is stored. For the previous time level also the original, not-injected, solution is needed to serve as an initial solution estimate in the Newton process.

The datastructure used to represent the nested grids and the corresponding solution data is to a large extent the same as in the 2D case. The boundary datastructure, however, is different because an extension of the 2D boundary structure to 3D would strongly complicate the task of a user to define a domain. Therefore a simpler, but computationally less efficient, implementation of the boundaries is used. To achieve a good vector performance, pointers to the boundary points in the grid structure are available so that, e.g., the computation of the PDE can be performed in two 'sweeps', one, using direct addressing, over the whole domain for the PDE system on the internal domain (1.1) and one, using indirect addressing, for the boundary conditions (1.2). For an efficient use of memory all information with respect to the solutions and the grids is stored consecutively in one real and one integer work array.

The linear solver used in this paper is BiCGStab[14], a variant of the well-known CGS method [5], in combination with ILU preconditioning.

The implementation of the vectorizable matrix vector multiply as described in [2] has been straightforwardly extended to 3D. To vectorize the ILU decomposition and the backsolve used in the preconditioning a variant of the hyperplane method [1] has been developed in analogue with the implementation described in [2]. In contrast with the 2D case the vector performance of the ILU preconditioning is now satisfactorily, since the resulting vector lengths will be much larger in three dimensions.

## 3. Numerical experiments

In this section we discuss the test results obtained with our 3D LUGR code 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 groundwater flow. The first issue is to evaluate the robustness of the LUGR method. To that aim we consider 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. In the description of the experiments the following notation has been used:

- STEPS: Number of successful time steps
- JACS: Number of Jacobian evaluations
- NIT: Number of Newton iterations
- LSIT $i$: Number of BiCGStab iterations in the $i$-th Newton iteration

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. Since the memory requisite is 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 of the solver 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

$$\begin{aligned} u_t + uu_x + vu_y + wu_z &= \varepsilon \Delta u, \\ v_t + uv_x + vv_y + wv_z &= \varepsilon \Delta v, \\ w_t + uw_x + vw_y + ww_z &= \varepsilon \Delta w. \end{aligned} \tag{3.5}$$

We determined an exact solution for (3.5) representing a wave front moving skew through a unit cube. Using the Cole-Hopf transformation (see, e.g., [15, pg. 97])

$$u = -2\varepsilon \frac{\varphi_x}{\varphi}, \quad v = -2\varepsilon \frac{\varphi_y}{\varphi}, \text{ and, } w = -2\varepsilon \frac{\varphi_z}{\varphi} \tag{3.6}$$

(3.5) is transformed into

$$\varphi_t = \varepsilon \Delta \varphi. \tag{3.7}$$

An exact solution representing a single shock is given by (cf. [15, pg. 110])

| | Level | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $t = 0.001$ | 342 | 1854 | 7649 | 31409 | 144221 |
| $t = 0.602$ | 342 | 2074 | 10290 | 45084 | 203990 |
| $t = 1.0$ | 342 | 2050 | 9681 | 42746 | 177528 |
| Uniform | 342 | 2170 | 15282 | 114274 | 882882 |

Table 1: Number of grid points used for Problem I

$$\varphi = f_1 + f_2, \quad f_j = \exp\left(-\frac{c_{j1}x + c_{j2}y + c_{j3}z}{2\varepsilon} + \frac{(c_{j1}^2 + c_{j2}^2 + c_{j3}^2)t}{4\varepsilon}\right). \tag{3.8}$$

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

$$x - y - z = -\frac{3}{4}t. \tag{3.9}$$

This is obtained for $c_{jj} = 1$ and $c_{ij} = 1/2$ giving as exact solution for (3.5)

$$u = \frac{f_1 + \frac{1}{2}f_2}{f_1 + f_2} = 1 - \frac{1}{2}\frac{1}{1 + \exp[(-x + y + z - \frac{3}{4}t)/(4\varepsilon)]}, \tag{3.10a}$$

$$v = w = \frac{3}{2} - u. \tag{3.10b}$$

Substituting $v$ and $w$ from (3.10b) into (3.5) we get, as our first test problem, the scalar PDE

$$u_t + uu_x + (\frac{3}{2} - u)(u_y + u_z) = \varepsilon\Delta u \tag{3.11}$$

on a domain that is a cube $[0,1] \times [0,1] \times [0,1]$ with a hole inside at $([\frac{1}{3}, \frac{2}{3}]^3)$. The initial solution and the Dirichlet boundary conditions are given by the exact solution (3.10a). We discuss the results on the time interval [0,1] for a Reynolds number of 500 ($\varepsilon = 0.002$).

*3.1.1 Numerical results for Problem I* Starting on a base grid with $\Delta x = \Delta y = \Delta z = \frac{1}{6}$ we allow a maximum of 5 grid levels resulting in a finest cell width of $\frac{1}{96}$. After 266 time steps the endpoint was reached, using at each grid level 532 Newton iterations and for each Newton iteration just 1 iteration of the linear system solver (sometimes even 0), which indicates that the ILU preconditioner is almost perfect for this problem. The maximum norm of the error at $t = 1.0$ was 0.04. In Table 1 we list the number of grid points used at each grid level for 3 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. As a matter of fact, for this number of points it would not be possible to store both the Jacobian and its ILU decomposition in the memory of the Cray Y-MP. 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 about 25% of the CPUtime is spent in routines dealing with the refinement.

Since the transition layer of the solution is $O(\sqrt{\varepsilon})$ a cell width of 1/96 is still rather coarse for a Reynolds number of 500 using central finite differencing, especially since the transition layer is placed skew through the cube. It was however not possible to add an extra level due to insufficient memory.

*3.2 Problem II: Rotating sphere problem*
Our second example is the advection-reaction system

$$\mathbf{c}_t + u\mathbf{c}_x + v\mathbf{c}_y + w\mathbf{c}_z = \mathbf{f}(\mathbf{c}), \tag{3.12}$$

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

$$\begin{aligned} f_1 &= -k_2 c_1 c_2 + k_1 c_2^2 \\ f_2 &= -k_1 c_2^2 + k_2 c_1 c_2. \end{aligned} \tag{3.13}$$

If we set the, constant, sum of $c_1$ and $c_2$ to $d$, the solution of (3.12) with zero velocity is given by

$$\begin{aligned} c_1(t) &= \frac{d}{k_1+k_2}\frac{k_1(1-\alpha)+(k_1+k_2)\alpha e^{-dk_2 t}}{1-\alpha+\alpha e^{-dk_2 t}} \\ c_2(t) &= d - c_1(t) \end{aligned} \tag{3.14}$$

with

$$\alpha = \frac{(k_1+k_2)c_1(0) - dk_1}{dk_2}. \tag{3.15}$$

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

$$\begin{aligned} u &= 2\pi\sqrt{2}(\frac{y+z}{2} - \frac{1}{2}) \\ v &= -\pi\sqrt{2}(x - \frac{1}{2}) \\ w &= v \end{aligned} \tag{3.16}$$

The exact sphere solution for (3.12) with $\mathbf{f} = 0$ is given by

$$\exp(-80[(x - r(t))^2 + (y - s(t))^2 + (z - s(t))^2]) \tag{3.17}$$

with

$$r(t) = (2 + \sin(2\pi t))/4, \quad s(t) = (2 + \frac{1}{2}\sqrt{2}\cos(2\pi t))/4. \tag{3.18}$$

Summarizing, our second test problem is given by (3.12) with the velocity field from (3.16) and the reaction term $\mathbf{f}$ from (3.13). At the inflow boundaries we impose the exact solution. The solution is given by (3.14) with $d = d(x, y, z, t)$ defined by (3.17) and $c_1(0) = 0$, resulting in

$$\begin{aligned} c_1(t) &= d(x,y,z,t)\frac{1 - e^{-d(x,y,z,t).t}}{\frac{1001}{1000} - e^{-d(x,y,z,t).t}} \\ c_2(t) &= d(x,y,z,t)\frac{10^{-3}}{\frac{1001}{1000} - e^{-d(x,y,z,t).t}} \end{aligned} \tag{3.19}$$

In the short time interval $[0, 0.01]$ the solution $c_2$ decreases over 90%. For the rest of the time interval, (3.12) is rather an advection problem with the extra difficulty that for negative solution values (3.12) becomes unstable.

| | Level | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| STEPS | 278 | 278 | 278 | 278 |
| JACS | 280 | 280 | 280 | 280 |
| NIT | 560 | 562 | 562 | 562 |
| LSIT 1 | 530 | 417 | 430 | 511 |
| LSIT 2 | 96 | 101 | 108 | 112 |
| Av. # pts | 1331 | 4400 | 13800 | 36000 |
| Uniform | 1331 | 9261 | 68921 | 531441 |

Table 2: Integration history for Problem II

*3.2.1 Numerical results for Problem II* For this test example we used a base grid with 11 points in each direction. A maximum of 4 grid levels was allowed.

Instabilities due to negative solution values caused a break-down of the run at $t = 0.83$. At $t = 0.5$ the maximum norm of the error had been 0.09 for the first component and 0.0004 for the second. We then did a second run where after every timestep 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 ($\mathbf{f} = 0$), viz., 0.18 for the first component (the maximum value was 0.82 instead of 1.0). The only difference in integration history between the two runs with and without reaction term was the time stepsizes used in the very first part of the time interval $[0, 0.03]$. The advection run used over the whole interval stepsizes of 0.004, whereas the run solving (3.12) used stepsizes from 0.00005 increasing to 0.004 within [0,0.03]. The integration performance over the rest of the interval was the same. For more complete information we refer to Table 2. Note that in this case the efficiency gain in memory of the LUGR method is much larger than in the previous case because the refinement area is now a sphere instead of a skew plane. Again, due to lack of memory, it would not be possible to solve this problem with the same accuracy on a uniform grid in core on the Cray Y-MP. The computational time spent in the refinement routines is much less than for problem I, ca. 7% for the scalar run. This overhead is small considering the low number of iterations needed to solve the nonlinear and linear systems.

*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 2 PDEs basic to groundwater flow: the continuity equation and the transport equation. For the background of these equations we refer to [11]. We here present the model in non-conservative form. As independent variables we have the pressure $p$ and the salt mass fraction $\omega$. The continuity equation for the fluid and the salt transport equation are given by

$$n\rho(\beta\frac{\partial p}{\partial t} + \gamma\frac{\partial \omega}{\partial t}) + \nabla \cdot (\rho \mathbf{q}) = 0, \tag{3.20a}$$

$$n\rho\frac{\partial \omega}{\partial t} + \rho \mathbf{q} \cdot \nabla\omega + \nabla \cdot (\rho \mathbf{J}^{\omega}) = 0, \tag{3.20b}$$

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

$$\mathbf{q} = -\frac{k}{\mu}(\nabla p - \rho \mathbf{g}), \tag{3.21}$$

with $\mathbf{g}$ the acceleration-of-gravity vector and $k$ the permeability coefficient of the porous medium. The density $\rho$ and the viscosity $\mu$ obey the state equations

$$\rho = \rho_0 \exp[\beta p + \gamma\omega], \tag{3.22}$$

| | | | | | |
|---|---|---|---|---|---|
| $n$ | 0.35 | $k$ | $7.18_{10^{-11}}$ | $nD_{mol}$ | $1_{10^{-9}}$ |
| $\rho_0$ | 1000 | $\mu_0$ | 0.001 | $\alpha_T$ | 0.001 |
| $\gamma$ | ln(2) | $g$ | 10 | $\alpha_L$ | 0.01 |
| $\beta$ | 0.0 | $t_{end}$ | $2_{10^4}$ | | |

Table 3: Data for Problem III.

$$\mu = \mu_0.m(\omega),\ m(\omega) = 1 + 1.85\omega - 4.1\omega^2 + 44.5\omega^3, \tag{3.23}$$

where $\rho_0$ is the reference density of fresh water and $\mu_0$ a reference viscosity. The equation for the salt-dispersion flux vector is given by Fick's law

$$\mathbf{J}^\omega = -n\mathbf{D}\nabla\omega, \tag{3.24}$$

with the dispersion tensor $\mathbf{D}$ for the solute salt defined as

$$n\mathbf{D} = (nD_{mol} + \alpha_T|\mathbf{q}|)I + \frac{\alpha_L - \alpha_T}{|\mathbf{q}|}\mathbf{q}\mathbf{q}^T,\ |\mathbf{q}| = \sqrt{\mathbf{q}^T\mathbf{q}}. \tag{3.25}$$

The coefficients $D_{mol}$, $\alpha_T$ and $\alpha_L$ correspond, respectively, with the molecular diffusion and the transversal and longitudinal dispersion.

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, that has a higher density than the 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, more demanding, a pollution with brine having a salt mass fraction of 0.25. 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 $\Omega = \{(x, y, z)|0 \le x \le 2.5, 0 \le y \le 0.5, 0 \le z \le 1.0\}$. The acceleration of gravity vector takes the form $\mathbf{g} = (0, 0, g)^T$, where $g$ is the gravity constant. The initial values at $t = 0$ at $\Omega \cup \partial\Omega$ are taken as

$$p(x, y, z, 0) = (0.03 - 0.012x + 1 - z)\rho_0 g, \quad \omega(x, y, z, 0) = 0. \tag{3.26}$$

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

$$\begin{array}{lll}
x = 0, & y \in [0, 0.5], z \in [0, 1]: & p = p(x, y, z, 0),\ \omega = 0, \\
x = 2.5, & y \in [0, 0.5], z \in [0, 1]: & p = p(x, y, z, 0),\ \omega_x = 0, \\
y = 0, 1 & x \in [0, 2.5], z \in [0, 1]: & q_2 = 0, \omega_y = 0, \\
z = 0 & x \in [0, 2.5], y \in [0, 0.5]: & q_3 = 0, \omega_z = 0, \\
z = 1 & x, y \notin B: & q_3 = 0, \omega_z = 0, \\
 & B = \{(x, y)|\ 0.375 \le x \le 0.4,\ 0.2 \le y \le 0.3\} & \\
 & (x, y) \in B: & \rho q_3 = -4.95_{10^{-2}}, \omega = \omega_B.
\end{array} \tag{3.27}$$

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 3.

*3.3.1 Numerical results for Problem IIIa ($\omega_B = 0.0935$)* As cell width of the base grid we used 0.1 in each direction. The maximum number of grid levels allowed was 4. In Fig. 1 we give contourplots of the salt concentration in the plane $y = 0.25$ after 2 and 6 hours. The contour lines of the higher salt mass fractions (0.5 - 1.0 times $\omega_B$) display a likely solution pattern and show no wiggles despite the steepness of the solution near the inlet slit. For the lower concentrations, however, there is an unexpected 'drip' in the solution. Since the density decreases from top to bottom, one would expect

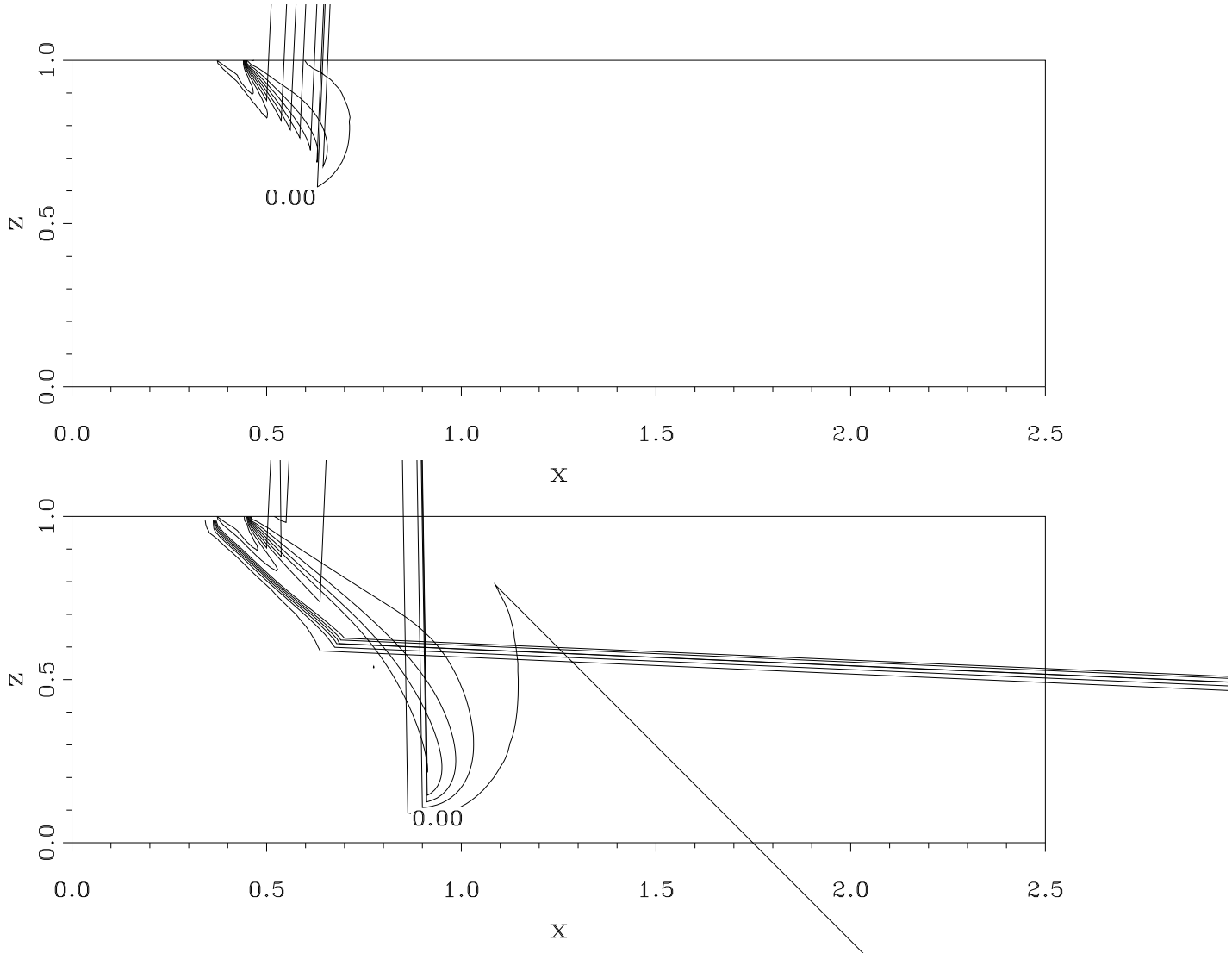


Figure 1: Problem IIIa ($\omega_B = 0.0935$), slice at $y = 0.25$
10% contour lines of the salt mass fraction after 2 hours (above) and after 6 hours (below)

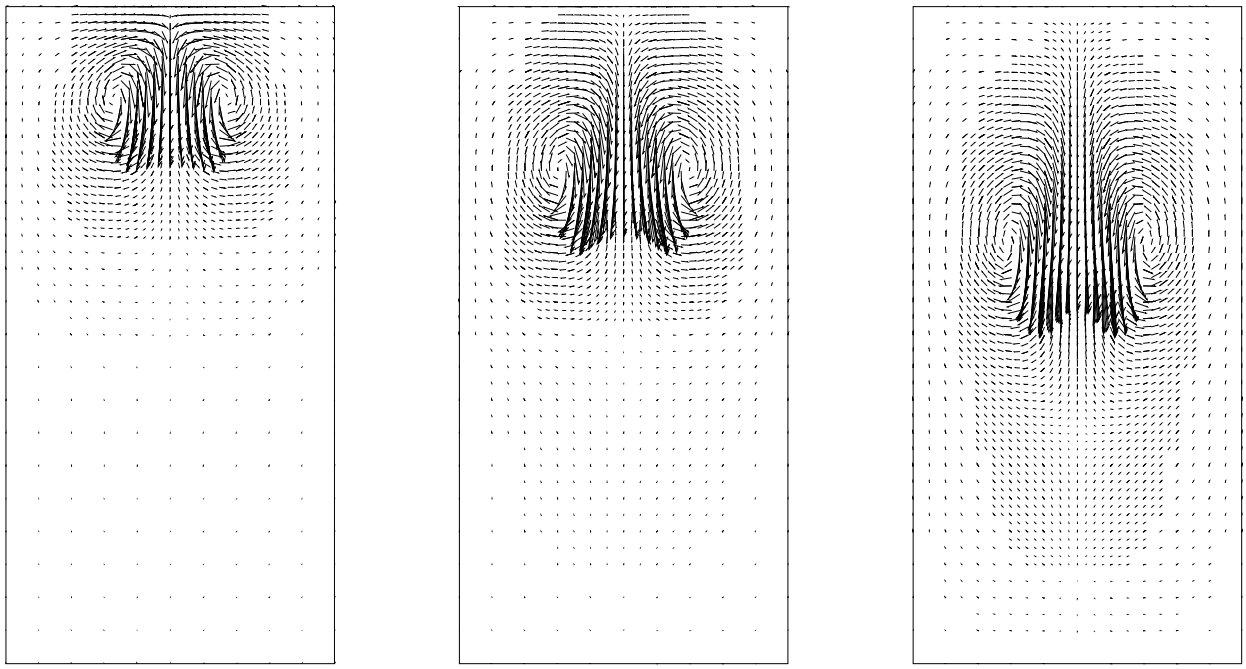

Figure 2: Problem IIIa ($\omega_B = 0.0935$): Flow field in the planes $x = 0.5,\ 0.6,\ 0.7$ after 6 hours
Length unit arrow: 0.263E-4, resp. 0.211E-4, resp. 0.194E-4

| | | Level | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| $t = 7200$ | STEPS | 142 | 142 | 142 | 142 |
| | JACS | 142 | 142 | 142 | 142 |
| | NIT | 284 | 284 | 287 | 302 |
| | LSIT 1 | 1148 | 732 | 843 | 1121 |
| | LSIT 2 | 331 | 247 | 343 | 566 |
| | # pts | 1716 | 2211 | 7817 | 25679 |
| $t = 21600$ | STEPS | 238 | 238 | 238 | 238 |
| | JACS | 238 | 238 | 238 | 238 |
| | NIT | 476 | 476 | 479 | 494 |
| | LSIT 1 | 1875 | 1459 | 1662 | 2196 |
| | LSIT 2 | 579 | 492 | 639 | 1043 |
| | # pts | 1716 | 4675 | 20419 | 66459 |
| | Uniform | 1716 | 11781 | 86961 | 667521 |

Table 4: Integration history for Problem IIIa

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 a second run on a uniform grid with a cell width of 0.025 gave a solution with the same characteristics. In Fig. 2 we give the flowfields in the planes $x = 0.5$, 0.6, and 0.7 after 6 hours.

The integration data are given in Table 4. The integration in time is again performed smoothly, the stepsize steadily increases from ca. 1 at the start of the problem to ca. 200 at $t_{end}$. 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 IIIb ($\omega_B = 0.25$)* 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. The contourplots in Fig. 3, again a slice from the solution at $y = 0.25$, show that, as expected, a pollutant with higher density will sink faster to the bottom, which is reached after 4 hours. The figures for the integration history are given in Table 5. Note that, even more striking than in the previous case, 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 [7].

It would not be feasible to solve problem IIIa or IIIb 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.

## 4. Vector results

In this section we will discuss the vector performance of the LUGR code for the three example problems. The timings were done on 1 processor of a Cray Y-MP, which has a clock cycle time of 6ns. This gives a theoretical peak performance on 1 processor of 167 Mflops and 333 when chaining an add and a multiply. Since during one cycle time 1 store and 2 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 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.

To measure the Megaflop rate and the CPU time of a routine we used the Cray utility Perftrace[4],

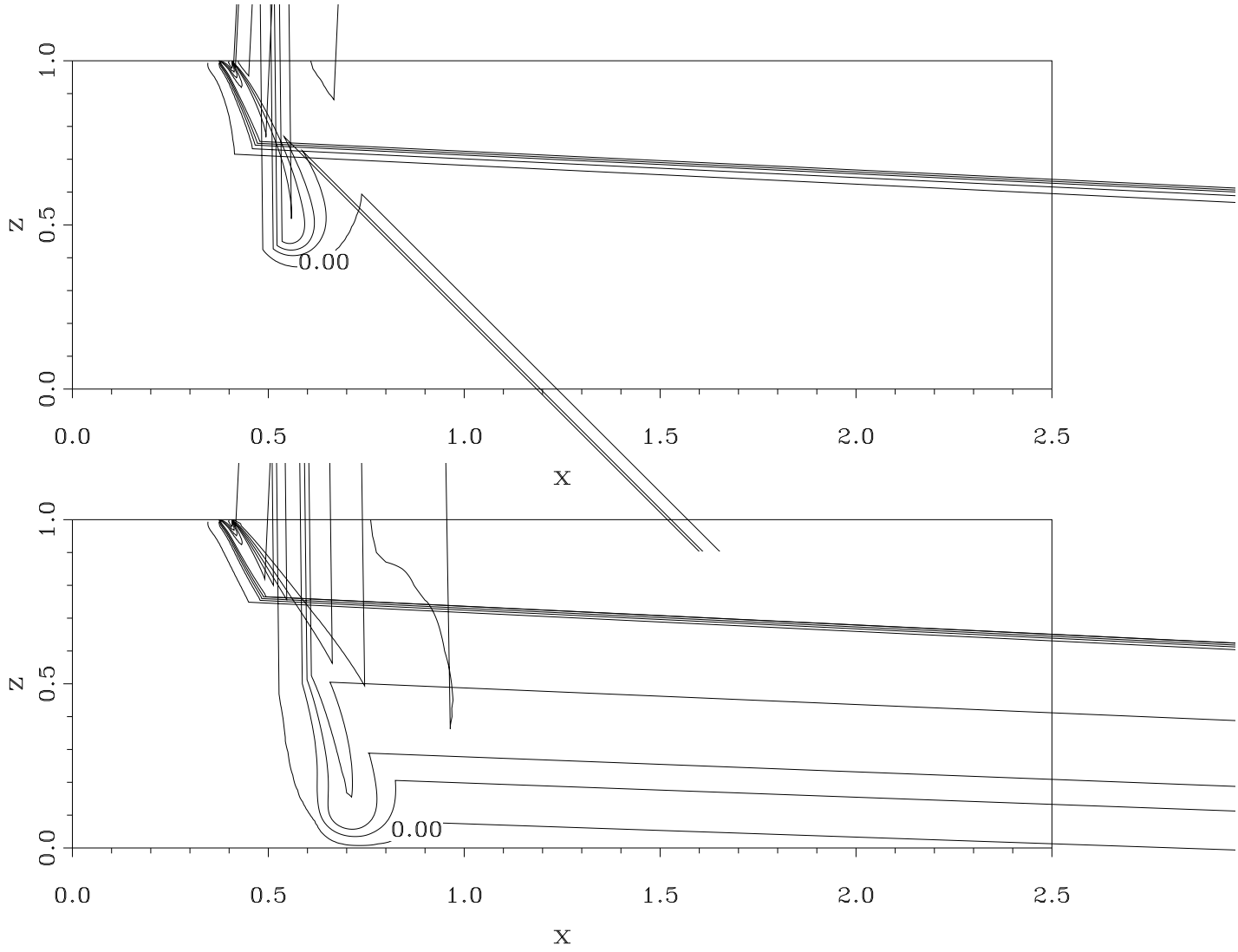


Figure 3: Problem IIIb ($\omega_B = 0.25$), slice at $y = 0.25$
10% contour lines of the salt mass fraction after 2 hours (above) and after 4 hours (below)

| | | Level | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $t = 7200$ | STEPS | 173 | 173 | 173 | 173 | 173 |
| | JACS | 177 | 176 | 176 | 175 | 198 |
| | NIT | 354 | 352 | 374 | 429 | 602 |
| | LSIT 1 | 1223 | 915 | 964 | 1286 | 2006 |
| | LSIT 2 | 372 | 321 | 444 | 708 | 1258 |
| | LSIT 3 | | | | 159 | 532 |
| | LSIT 4 | | | | 51 | 276 |
| | LSIT 5 | | | | | 197 |
| | LSIT 6 | | | | | 171 |
| | LSIT 7 | | | | | 124 |
| | LSIT 8 | | | | | 97 |
| | LSIT 9 | | | | | 79 |
| | LSIT 10 | | | | | 63 |
| | # pts | 1716 | 2783 | 9681 | 29949 | 92149 |
| $t = 14400$ | STEPS | 223 | 223 | 223 | 223 | 223 |
| | JACS | 233 | 232 | 232 | 238 | 297 |
| | NIT | 466 | 464 | 501 | 661 | 746 |
| | LSIT 1 | 1739 | 1355 | 1419 | 1958 | 3328 |
| | LSIT 2 | 563 | 501 | 700 | 1097 | 2047 |
| | LSIT 3 | | | 58 | 386 | 920 |
| | LSIT 4 | | | | 121 | 573 |
| | LSIT 5 | | | | 66 | 444 |
| | LSIT 6 | | | | | 398 |
| | LSIT 7 | | | | | 328 |
| | LSIT 8 | | | | | 292 |
| | LSIT 9 | | | | | 269 |
| | LSIT 10 | | | | | 236 |
| | # pts | 1716 | 3971 | 16119 | 50247 | 132478 |
| | Uniform | 1716 | 11781 | 86961 | 667521 | 5229441 |

Table 5: Integration history for Problem IIIb

that gives the hardware performance by program unit.

First we give for all problems the overall vector floating-point performance:

Problem I 36 Megaflop
Problem II 69 Megaflop
Problem IIIa 113 Megaflop
Problem IIIb 115 Megaflop

For Problem I and to a lesser extent also for Problem II the program 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 multiply about 150 and the preconditioner circa 75 Mflops. The disappointing overall vector performance of the first two problems is caused by the grid refinement 'overhead' routines that contain no floating-point operations and that take a considerable amount of the total CPU time. In the case of the, scalar, Problem I this amounts to even 70% of the total CPU time. For the two-component system of Problem II the overhead still amounts to 30%. In Tables 6 and 7 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

| | # calls | Avg time | ACM % | Mflop |
|---|---|---|---|---|
| INJON | 3176 | 7.8E-2 | 15.2 | 0 |
| ILU backs | 6854 | 2.3E-2 | 24.9 | 74 |
| JACSLP | 1065 | 1.4E-1 | 34.4 | 0 |
| JACSUP | 1065 | 1.4E-1 | 43.8 | 0 |
| MKBND | 1064 | 1.4E-1 | 52.8 | 0 |
| Total CPU time in seconds | | | 1630 | |

Table 6: Vector performance of top 5 routines for Problem I.

| | # calls | Avg time | ACM % | Mflop |
|---|---|---|---|---|
| ILU dec | 1120 | 1.5E-1 | 21.3 | 66 |
| ILU backs | 6868 | 2.2E-2 | 40.8 | 75 |
| MATVEC | 4622 | 1.5E-2 | 49.7 | 143 |
| INJON | 2490 | 2.4E-2 | 57.3 | 0 |
| JACSLP | 841 | 4.4E-2 | 62.1 | 0 |
| Total CPU time in seconds | | | 788 | |

Table 7: Vector performance of top 5 routines for Problem II.

| | # calls | Avg time | ACM % | Mflop |
|---|---|---|---|---|
| ILU backs | 21933 | 1.8E-2 | 29.6 | 78 |
| PDEF | 20965 | 1.9E-2 | 58.2 | 178 |
| MATVEC | 20008 | 1.2E-2 | 76.5 | 145 |
| ILU dec | 952 | 1.0E-1 | 83.5 | 66 |
| INJON | 2130 | 1.6E-2 | 86.0 | 0 |
| Total CPU time in seconds | | | 1350 | |

Table 8: Vector performance of top 5 routines for Problem IIIa.

| | # calls | Avg time | ACM % | Mflop |
|---|---|---|---|---|
| ILU backs | 42332 | 6.5E-2 | 40.8 | 79 |
| MATVEC | 38804 | 4.5E-2 | 66.7 | 143 |
| PDEF | 28470 | 4.6E-2 | 85.9 | 177 |
| ILU dec | 1248 | 2.1E-1 | 89.9 | 70 |
| BiCGStab | 3528 | 3.4E-2 | 91.7 | 267 |
| Total CPU time in seconds | | | 6740 | |

Table 9: Vector performance of top 5 routines for Problem IIIb.

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 8 and 9). For Problem IIIa the overhead costs are ca. 10% and for Problem IIIb 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 datastructure or by using a one-step time-integration formula. However, if the number of components is large and/or the nonlinearity of the problem high (as for instance in Problem IIIb) the CPU time needed for the grid structure and refinement is again negligible.

The preconditioning of the linear systems vectorizes better than in the 2-dimensional case, since the number of computations that can be done independently from each other is much larger. The Mflop rate however is still hampered by the fact that only 1 indirect load/store instruction can be issued at a time, versus 2 load and 1 store instructions without gather/scatter necessity (cf. [2]).

Our test results indicate that memory demands still are the bottleneck (NB. the Cray Y-MP we used has a memory limit of 52Mw). Since we have used for the space discretization a central finite-difference scheme, this will readily be the case for convection dominated transport problems for which a relatively large number of grid levels may be required to avoid wiggles at places where the solution has large spatial gradients. This results in a large number of grid points and, especially for PDE systems with a large number of components, in Jacobians that require a considerable amount of datastorage. On the other hand, our grid stencil has 19 points to allow mixed derivatives. Omitting these would result in a 9 point stencil and hence in a lower memory demand.

Therefore, one of our next objectives is to implement and/or develop a matrix-free non-linear system solver to avoid the storage of Jacobian and ILU matrices. An additional advantage is that tailor-made space-discretization schemes, resulting in other couplings than the here used 19-point stencil, can then be more easily implemented.

## Acknowledgements

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. Stat. Comput.*, 9(1):122–151, 1988.

2. J.G. Blom and J.G. Verwer. Vectorizing matrix operations arising from PDE discretization on 9-point stencils. Report NM-R9221, CWI, Amsterdam, 1992. (to appear in J. Supercomputing).

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. Cray Research, Inc. *UNICOS Performance Utilities Reference Manual*, SR-2040 6.0 edition.

5. P. Sonneveld. CGS: A fast Lanczos-type solver for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989.

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

7. R.A. Trompert. A note on singularities caused by the hydrodynamic dispersion tensor. Report NM-R9302, CWI, Amsterdam, 1993.

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

9. R.A. Trompert and J.G. Verwer. Analysis of the implicit Euler local uniform grid refinement method. *SIAM J. Sci. Comput.*, 14:259–278, 1993.

10. R.A. Trompert and J.G. Verwer. Runge-Kutta methods and local uniform grid refinement. *Math. Comp.*, 60:591–616, 1993.

11. R.A. Trompert, J.G. Verwer, and J.G. Blom. Computing brine transport in porous media with an adaptive-grid method. *Int. J. Numer. Meth. in Fluids*, 16:43–63, 1993.

12. 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, editors, *Proc. 14-th Biennial Dundee Conference on Numerical Analysis*, pages 267–284. Pitman Research Notes in Mathematics Series 260, 1992.

13. J.G. Verwer and R.A. Trompert. Analysis of local uniform grid refinement. *Appl. Numer. Math.*, 13:251–270, 1993.

14. 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. Stat. Comput.*, 13(2), 1992.

15. G.B. Whitham. *Linear and Nonlinear Waves*. John Wiley & Sons, Inc., New York, London, Sydney, Toronto, 1974.