VLUGR2:
A Vectorizable Adaptive Grid Solver for PDEs in 2D

J.G. Blom, R.A. Trompert, and J.G. Verwer

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

Abstract
This paper deals with an adaptive-grid finite-difference solver for time-dependent two-dimensional systems of partial differential equations. It describes the ANSI FORTRAN 77 code, VLUGR2, auto-vectorizable on the Cray Y-MP, that is based on this method. The robustness and the efficiency of the solver, both for vector and scalar processors, is illustrated by the application of the code to two example problems arising from a groundwater-flow model.

Keywords & Phrases: software, partial differential equations, method of lines, adaptive grid methods, non-symmetric sparse linear systems, iterative solvers, vectorization.
Note: This work was supported by Cray Research, Inc., under grant CRG 92.05, via the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF).

1. Introduction
In previous work [11, 12, 13, 15, 14, 16, 10] an adaptive-grid finite-difference method to solve time-dependent two-dimensional systems of partial differential equations (PDEs) was discussed. Among others, a research code, MOORKOP[9], was developed that uses an implicit time-stepping method. In [1] we described a FORTRAN 77 code, VLUGR2, that is based on MOORKOP but with the intention to use it on a vector processor. To that aim the datastructure was adapted to facilitate the vector processing of the most time consuming parts of the code, and to make the use of memory as compact and as small as possible.

In this paper we describe the final version of VLUGR2. The main difference with the version in [1] is the solution of the systems of nonlinear equations. The nonlinear systems in VLUGR2 are solved by a choice from three solvers. The first two use modified Newton with as linear solver either BiCGStab[17] + ILU preconditioning or GCRO[8] with a simple (block) diagonal scaling. The third option is a matrix-free implementation of the second solver. Also the implementation of the time integration part slightly differs from [1] and is now identical to the implementation used in VLUGR3, which is the 3D extension of VLUGR2. The 3D code is described in detail in [3, 4].

The paper is organized as follows. In Section 2 we define the problem class of VLUGR2. Section 3 is devoted to an outline of the Local Uniform Grid Refinement method as implemented in VLUGR2 and contains a survey of the choices made in the code with respect to the refinement strategy and the integration strategy including the solution of the arising (non)linear systems. In Section 4 we discuss the for a user relevant part of the implementation and the amount of data storage required. There we also describe how to use VLUGR2, enlightened by an elaborated example problem. In Section 5 we discuss the performance of VLUGR2 on the same groundwater-flow problems as were used in [14, 1]. The performance evaluations were done on a Cray Y-MP in scalar and vector mode. Finally, Section 6 contains a summary of our findings.
2. PDE Definition
VLUGR2 has been designed to solve initial boundary value problems for systems of partial differential
equations that fit in the following master equation
\[ F(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) = 0, \quad (x, y) \in \Omega, \quad t > t_0, \] (2.1)
where the solution \( u \) may be a vector and the domain \( \Omega \) an arbitrary domain that can be described
by right-angled polygons (see, e.g., Fig. 1). The boundary conditions belonging to system (2.1) are
formulated as
\[ B(t, x, y, u, u_t, u_x, u_y) = 0, \quad (x, y) \in \partial \Omega, \quad t > t_0, \] (2.2)
and the initial conditions satisfy
\[ u(t_0, x, y) = u_0(x, y), \quad (x, y) \in \Omega \cup \partial \Omega. \] (2.3)

3. The Algorithm
In this section we give a short survey of the algorithm. We discuss the method parameters of VLUGR2
that can be specified by the user. For further information about algorithmic aspects we refer to [3].

3.1 Outline of the LUGR algorithm
The virtue of a Local Uniform Grid Refinement method lies in the fact that one reaches the accuracy of a
fine mesh width with considerably less computational effort and memory requirements, since the fine
subgrids cover only part of the domain. The concept of such a method is simple. Starting from a, uniform, coarse base grid covering
the whole domain, ever finer uniform subgrids are recursively created in a nested way in regions with high spatial activity. So all grid levels consist of one or
more disjoint sets of interconnected grid cells, all having the same size. Each time step a new initial
boundary value problem is solved at all grid levels ordered from coarse to fine. The fine grid solution
values are injected in the coinciding coarser grid nodes. The location and size of the subgrids are
adjusted each time step to follow, e.g., the movement of the steep parts of the solution.

For the space discretization standard second-order finite differences are used, central on the internal
domain and one-sided at the boundaries. At 'internal' corners, i.e., with an angle of 270°, we use also
one-sided differencing to prevent difficulties with the ILU decomposition of the Jacobian in the Newton
process. Where grid refinement is required we divide a grid cell in 4 equal parts. Where interpolation
is needed to get solution values, linear interpolation is used.

The refinement is governed by a curvature monitor. For each grid point \((i, j)\) this monitor is
determined by
\[ SPCMON(i, j) := \max_{i' = 1, NPE} SPCTOL(ic) \cdot (|Ax^2 \cdot u_{xx}(i, j)| + |Ay^2 \cdot u_{yy}(i, j)|), \] (3.1)
where \( Ax \) and \( Ay \) are the grid width in the x- and the y-direction, respectively, and
\[ SPCTOL(ic) := \frac{SPCGT(ic)}{UMAX(ic) \cdot TOLS}. \] (3.2)
The variables on the right-hand side of (3.2) are user specified quantities, \( 0 \leq SPCGT \leq 1 \) 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.

VLUGR2 also offers the possibility to enforce grid refinement in a priori selected regions by adapting
the \( SPCMON \) values (see Section 4.4).
3.2 Integration in time
We solve the system of PDEs using the Method of Lines approach. The PDEs are discretized in space and the resulting system of ODEs or DAEs is solved in time using the second-order two-step implicit BDF method with variable step sizes. In the first time step we apply as usual Backward Euler. The time integration is controlled by a second solution monitor, viz., the maximum over all existing grid levels of

$$\| \Delta t \mathbf{u} \|_w,$$

where $\Delta t$ is the current time step size and $\mathbf{u}_t$ is approximated by first-order finite differences. The norm used in (3.3) is a weighted root-mean-square norm

$$\| \mathbf{v} \|_w = \| W \mathbf{v} \|_2,$$

with $W$ a diagonal matrix given by

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

The entries $w_i$ of the diagonal matrix $W$ are defined by

$$w_{ipt, ic} = \text{TIMWGT}(ic) / (\text{ABSTOL}(ic) + [U_{ipt, ic}^{n+1} \cdot \text{RELTOL}(ic)]), \quad ipt \in \Omega \ \text{and} \ ic = 1, \text{NPDE},$$

with

$$\text{ABSTOL}(ic) = 0.01 \cdot \text{TOL} \cdot \text{UMAX}(ic) \ \text{and} \ \text{RELTOL}(ic) = \text{TOL}.$$  

(3.7)

The variables TIMWGT and TOLT are the, user specified, analogues of the variables in (3.2).

To solve the resulting system of nonlinear equations a Newton process is used combined with a preconditioned iterative linear solver. The stopping criterion for the Newton process is

$$\frac{\rho}{1 - \rho} \| \mathbf{U}^{(k)} - \mathbf{U}^{(k-1)} \|_w < \text{TOLNEW},$$

(3.8)

where $\rho$ is an approximation of the convergence rate. The entries $w_i$ of the diagonal matrix $W$ in (3.4)-(3.5) are here defined by

$$w_{ipt, ic} = 1.0 / (\text{ATOL}(ic) + [U_{ipt, ic}^0 \cdot \text{RTOL}(ic)]), \quad ipt \in \Omega \ \text{and} \ ic = 1, \text{NPDE},$$

(3.9)

with

$$\text{ATOL}(ic) = 0.01 \cdot \text{TOL} \cdot \text{UMAX}(ic) \ \text{and} \ \text{RTOL}(ic) = \text{TOL}; \quad \text{TOL} = 0.1 \ \text{min(TOL}^2, \text{TOLS}).$$

(3.10)

For the linear solver the stopping criterion reads

$$\| P^{-1} r^{(k)} \|_w < \text{TOLSS} / 2^k,$$

(3.11)

with $k$ the current Newton-iteration index and $P^{-1}$ the preconditioner. Both stopping criteria can be influenced easily by the user by changing the value of TOLNEW (default 1.0) and TOLSS (TOLNEW/10) in parameter statements. The maximum number of iterations in both solvers can also be set via parameter statements.

In VLIB2R2 the nonlinear systems can be solved with modified Newton and the linear systems either with BICGSTAB[17] with ILU preconditioning or with GCRO8 and (block) diagonal scaling as preconditioner. GCRO is a recursive variant of GCR in which the inner (GMRES) loop dynamically generates a preconditioner. A third possibility is to use matrix-free Newton combined with the latter linear solver. In the first two solvers the Jacobian matrix is computed by numerical differencing, once per time step, and stored. How to approximate the entries of the Jacobian is discussed in [1, 3]. For the problems we solved with the code the approximation of the Jacobian was sufficiently accurate.
However, if for a specific problem Newton failures would often be the cause of time step failures, it could be worthwhile to store the exact partial derivatives instead of the approximated ones (see Section 4.4).

For the solvers that make explicit use of the Jacobian we developed in [2] a vectorizable implementation of the matrix-vector multiply and of the ILU preconditioning routines. This implementation is designed especially for systems of PDEs in 2D discretized on a 9-point stencil and on a grid that is bounded by arbitrary right-angled polygons.

In the third solver the product of the Jacobian times a vector, which is the only need for the Jacobian when using GCRO (or BiCGStab), is approximated by a difference quotient on the residual. In this case the vector performance of the matrix-vector multiply is of course determined by the user implementation of the residual evaluation. An advantage of this solver is that it allows an easy change of space-discretization schemes resulting in other couplings than the 9-point stencil.

4. The code

4.1 Datastructure

To achieve a good vector performance, pointers are stored to the boundary points in the grid structure so that, e.g., the computation of the PDE can be performed in two ‘sweeps’. In the first, using direct addressing, the PDE residual on the internal domain (2.1) is computed over the whole domain, including the boundary. In the second sweep, using indirect addressing, the values on the boundary are reset by incorporating the boundary conditions (2.2).

The solution at a specific grid level is stored row-wise, one component vector after the other. Solutions from 3 different time levels are saved. For the oldest time level only the injected solution values for the computation of $U_t$ with the BDF method. For the previous time level also the original, not-injected, solution is needed to serve as an initial solution estimate in the Newton process.

A grid at a specific grid level is stored in the following datastructure

- **LR0W**: the actual number of rows in the grid, the pointers to the start of a row in the grid, and the number of grid points
- **IR0W**: the row number of a row in the (virtual) rectangle
- **ICOL**: the column number of a grid point in the (virtual) rectangle
- **ILBND**: the total number of physical boundaries and corners in the actual domain, the pointers to the start of a specific boundary or corner in LBND, and the number of boundary points
- **ILBND**: the type of the boundaries

```
1: lower boundary
2: left boundary
3: upper boundary
4: right boundary
12: lower left corner (90°)
23: left upper corner (90°)
34: upper right corner (90°)
41: right lower corner (90°)
21: left lower corner (270°)
32: upper left corner (270°)
43: right upper corner (270°)
```
14: lower right corner ($270^\circ$)

- **LBWD**: the pointers to the boundary points in the actual grid
- **LE Below**: pointer to node below in the actual grid or 0, if index node is lower boundary point
- **Lab Above**: pointer to node above in the actual grid or 0, if index node is upper boundary point.

All grids from the three different time levels are saved. For the base grid all information is saved, i.e., the arrays containing the grid information, the pointer arrays needed for the Jacobian and the pointer arrays for the hyperplane ordering. The latter is done even if the matrix-free option is chosen, to facilitate a continuation of the integration using another solver. For the higher level grids only the first 3 arrays (LRow, IRow, ICO) are saved.

The above mentioned arrays and workspace for the nonlinear system solver are stored consecutively in the real and integer work arrays. Pointers for each time and grid level indicate the start in a work array of a solution or grid at a specific time and at a specific grid level. For a more complete description of the contents of the work arrays we refer to the documentation of the enveloping routine VLUGR2.

### 4.2 Data storage requirements

Many architectures have a relatively small main memory and make use of so-called virtual memory. On those systems the real-time performance is often not determined by the floating-point operations but by memory operations (data swap). Therefore our goal was to keep the amount of workspace small, and to store all data in a compact way.

**VLUGR2** requires roughly the following amount of memory

**INTEGER**: $5 \cdot \text{MAXLEV} \cdot \text{NPTS}_{av} + 5 \cdot \text{NPTS}_{max} + \text{LSSIWK}$

**REAL**: $5 \cdot \text{MAXLEV} \cdot \text{NPDE} \cdot \text{NPTS}_{av} + (2 + 9 \cdot \text{NPDE}) \cdot \text{NPTS}_{max} + \text{LSSRWK}$

where \( \text{NPTS}_{av} \) is the average number of grid points over all grid levels and \( \text{NPTS}_{max} \) is the maximum number of grid points on any level. The workstorage needed for the linear system solver (including Jacobian and/or preconditioner) is when using BiCGStab + ILU

- \( \text{LSSIWK} : 9 \cdot \text{NPTS}_{max} \)
- \( \text{LSSRWK} : 18 \cdot \text{NPDE} \cdot \text{NPTS}_{max} \cdot \text{NPDE} \)

When using GCRO with the Jacobian approximated and stored

- \( \text{LSSIWK} : 6 \cdot \text{NPTS}_{max} \)

and the real workspace is dependent on the choice of the preconditioner:

- GCRO + block-diagonal scaling

  - \( \text{LSSRWK} : \text{JACRWK} + \text{NPTS}_{max} \cdot \text{NPDE} \cdot (\max(\text{NPDE} \cdot 5 + 3, 2 \cdot \text{MAXLR} + \text{MAXL} + 5) + \text{NPDE}) \),

- GCRO + diagonal scaling

  - \( \text{LSSRWK} : \text{JACRWK} + \text{NPTS}_{max} \cdot \text{NPDE} \cdot (2 \cdot \text{MAXLR} + \text{MAXL} + 6) \),

with \( \text{JACRWK} : 9 \cdot \text{NPDE} \cdot \text{NPTS}_{max} \cdot \text{NPDE} \). When using matrix-free GCRO, \( \text{LSSIWK} = 0 \) and the real workspace is as before with \( \text{JACRWK} = 0 \). In the above \( \text{MAXLR} \) is the maximum number of outer GCR iterations and \( \text{MAXL} \) is the maximum number of inner GMRES iterations. These values can be adapted via parameter statements.

The workstorage required will be checked against the user defined workspace. If the workspace is too small, a message is printed with the needed amount given.
4.3 How to use VLUGR2
Our aim was to keep the use of VLUGR2 as simple as possible. Most method parameters are set in parameter statements in the code itself. The user has to specify only the problem parameters and routines. If one wishes to change one of the method parameters the corresponding parameter statements can be changed in the code.

In the simplest case, a well-scaled problem on a rectangle, one has to specify

- the number of PDEs,
- the initial and final time, and the initial step size,
- the lower-left and the upper-right corner of the domain, and the initial grid width in the \( x \)- and \( y \)-direction,
- the space and time tolerance TOLS and TOLT, and
- the subroutines PDEIV, PDEF, and PDEBC, to specify respectively, the initial solution (2.3), the PDE system at the internal domain (2.1) and the boundary conditions (2.2).

In this case the parameters SPCWGT, TIMWGT and UMAX (cf. (3.2) and (3.6)) are set to their default value, which is 1. If the initial domain is not a true rectangle, a virtual rectangle is to be placed around the irregular domain. In this case the user should also provide a routine that defines the initial grid. Furthermore, one can supply a routine to enforce grid refinement and a monitor routine for user purposes that will be called after each successful time step (cf. Section 4.4).

4.4 General description
The calling sequence of VLUGR2 is

```fortran
CALL VLUGR2 (NPDE, T, TOUT, DT, XL, YL, XR, YU, DX, DY,
  + TOLS, TOLT, INFO, RINFO, RWK, LENRWK, IWK, LENIWK, LWK, LENLWK,
  + MNTR)
```

where in the simplest case, using all default values the meaning of the parameters is

NPDE: the number of PDE components

T: the initial time

TOUT: the final time

DT: the initial time step size

\((XL, YL):\) coordinate of lower-left corner of domain (a rectangle)

\((XR, YU):\) coordinate of upper-right corner of domain

DX: cell width in \( x \)-direction of base grid

DY: cell width in \( y \)-direction of base grid

TOLS: space tolerance

TOLT: time tolerance

INFO: \( INFO(1) = 0 \), which implies that default parameters are used
RINFO: dummy array

RWK(LENRWK): real workspace, LENRWK \approx (19+18\cdot\text{NPDE})\cdot\text{NPTS}\cdot\text{NPDE}, with NPTS the expected maximum number of points on a grid

IWK(LENIWK): integer workspace, LEINIWK \approx 24 \cdot \text{NPTS}

LWK(LENLWK): logical workspace, LENLWK \approx 2 \cdot \text{NPTS}

MNTR: MNTR = 0, first call of VLUGR2 for this problem.

Furthermore one should supply the routines

PDEIV to specify the initial solution (2.3),
PDEF to specify the PDE system at the internal domain (2.1), and
PDEBC to specify the boundary conditions (2.2).

If one wants to continue the integration after returning from VLUGR2, one should set MNTR to 1. All parameters can be changed although T, DT, XL, YL, XR, YU, DX, and DY will be overwritten with their old values. The contents of the work arrays RWK and IWK should not be altered.

A more sophisticated use can be made with the aid of the INFO and RINFO arrays and by overloading some subroutines. If INFO(1) \neq 0 a number of parameters can be specified in INFO and RINFO. The value between brackets is the default value used when INFO(1) = 0.

INFO(2): MAXLEV (3), the maximum number of grid levels allowed

INFO(3): RCTDOM (0), if RCTDOM = 0 the initial domain is a rectangle, otherwise the user should specify the subroutine INIDOM to define the initial grid (see below)

INFO(4): LINSYS (0), the linear system solver and preconditioner

  0: BiCGStab + ILU
  10: GCRO + block-diagonal scaling
  11: GCRO + block-diagonal scaling (neglecting first-order derivatives at the boundaries)
  12: GCRO + diagonal scaling
  13: GCRO + diagonal scaling (neglecting first-order derivatives at the boundaries)
  20: matrix-free GCRO + block-diagonal scaling
  21: matrix-free GCRO + block-diagonal scaling (neglecting first-order derivatives at the boundaries)
  22: matrix-free GCRO + diagonal scaling
  23: matrix-free GCRO + diagonal scaling (neglecting first-order derivatives at the boundaries)

INFO(5): LUNPDS (0), the logical unit number of the file where information on the integration history will be written. If LUNPDS = 0 only global information will be written on standard output

INFO(6): LUNLS (0), the logical unit number of the file where information on the Newton process will be written. If LUNLS = 0 no information will be written

INFO(7): LUNLSS (0), the logical unit number of the file where information on the linear system solver will be written. If LUNLSS = 0 no information will be written

RINFO(1): DTMIN (0.0), the minimum time step size allowed
RINFO(2): DTMAX (TOUT-T), the maximum time step size allowed

RINFO(3): UMAX ((1.0)), the approximate maximum values of the PDE solution components. These values are used for scaling purposes

RINFO(3+NPDE): SPGW (1.0), the weighting factors used in the space monitor to indicate the relative importance of a PDE component on the space monitor

RINFO(3+2-NPDE): TIGW (1.0), the weighting factors used in the time monitor to indicate the relative importance of a PDE component on the time monitor.

The subroutines that are candidates for replacement by the user are

MONITR to monitor the solution or the grids; will be called after each successful time step

CHSPCM to enforce grid refinement at a specific point in space and time and on a specific level

INIDOM to specify the initial grid for a domain that is not rectangular

DERIV to store the exact partial derivatives of the residual F with respect to (the derivatives of) U.

Finally, the package also contains a number of routines that facilitate the use of the datastructure

PRDOM to print the domain one has defined with INIDOM

SETXY to get the x- and y-coordinates corresponding with the grid points

PRSOL to print the solution and the corresponding coordinate values at all grid levels

DUMP to dump all necessary information for a restart on file

RDDUMP to read all necessary information for a restart from the dump file.

For the details of these routines we refer to the documentation in VUGR2.

4.5 Example problem
Our example problem is the two-dimensional Burgers’ system

\[ u_t = -uu_x - vv_y + \varepsilon (u_{xx} + u_{yy}) \]  \hspace{1cm} (4.1a)
\[ v_t = -sv_x - vv_y + \varepsilon (v_{xx} + v_{yy}) \]  \hspace{1cm} (4.1b)

on the domain \( \Omega \) as given in Fig. 1. On the boundaries \( \partial \Omega \) we prescribe Dirichlet conditions. An exact solution is given by (cf. [5])

\[ u = \frac{3}{4} - \frac{1}{4} \frac{1}{1 + \exp((-4x + 4y - t)/(32 \cdot \varepsilon))} \]  \hspace{1cm} (4.2a)
\[ v = \frac{3}{4} + \frac{1}{4} \frac{1}{1 + \exp((-4x + 4y - t)/(32 \cdot \varepsilon))} \]  \hspace{1cm} (4.2b)

The solution represents a wave front at \( y = x + 0.25t \). The speed of propagation is \( \sqrt{2}/8 \) and is perpendicular to the wave front. We solve this problem at the time interval \([0,0,3,0]\) and with \( \varepsilon = 10^{-3} \).

For this example we will describe shortly how to write the user routines. The complete text of the user program is enclosed with the package.
Figure 1: Example domain
Left the physical domain, right the base grid on the virtual rectangle.
Numbers at the left and the right are the node numbers of the grid points
Numbers inside the domain give the boundary or corner order as stored in the boundary structure

In the first call of VLUGR2 the initial grid is formulated in the INIDOM routine, a refinement is enforced
up to level 3 in the lower right corner of the physical domain, and the monitor routine is used to print
the error after each time step and at each grid level. After \( t = 1.0 \) we will stop program 1, write all
information to file and restart the computation with a different program reading the data from file. In
that run we overload DERIVF with our own version which stores the exact partial derivatives \( \partial F / \partial U_p \).

We enclose the domain by a virtual rectangle \(((0,0,0),(1,0,1))\) and make a virtual base grid of
11 \times 11 grid points (cf. Fig. 1). For the first part of the time interval \([0,1.0]\) the user program will contain

```
MNTR = 0
NPDE = 2
T    = 0.0
TOUT = 1.0
DT   = 0.001
```

C Since domain is not a rectangle the grid parameters need not to be
C specified

```
TOLV = 0.1
TOLT = 0.05
INFO(1) = 1
```

C MAXLEV

```
INFO(2) = 5
```

C Domain not a rectangle

```
INFO(3) = 1
```

C Linear system solver: GCR0 + Diagonal scaling (no first order
C derivatives at the boundaries)

```
INFO(4) = 13
```

C

```
OPEN (UNIT=61,FILE='RunInfo')
```
C Write integration history to unit # 61
    INFO(5) = 61
C Write Newton info to unit # 61
    INFO(6) = 61
C Write GCR0 info to unit # 61
    INFO(7) = 61
C DTMIN = 1E-7
    RINFO(1) = 1.0E-7
C DTMAX = 1.0
    RINFO(2) = 1.0
C UMAX = 1.0
    RINFO(3) = 1.0
    RINFO(4) = 1.0
C SPCWGT = 1.0
    RINFO(5) = 1.0
    RINFO(6) = 1.0
C TIMWGT = 1.0
    RINFO(7) = 1.0
    RINFO(8) = 1.0
C
C Call main routine
    CALL VLUGR2 (NPDE, T, TOUT, DT, XL, YL, XR, YU, DX, DY,
            + TOI, TOIT, INFO, RINFO, RWK, LENRWK, IWK, LENIWK, LWK, LENLWK,
            + MNTR)

Since we want to restart the computation we write an unformatted file with all the necessary information

OPEN (UNIT=LUNDMP, FILE='DUMP', FORM='UNFORMATTED')
CALL DUMP (LUNDMP, RWK, IWK)
CLOSE(LUNDMP)

Next the PDE defining routines

SUBROUTINE PDEIV
    . . .
    DO 10 I = 1, NPTS
        U(I,1) = 0.75 - 0.25/(1+EXP((-4*X(I)+4*Y(I)-T)/(32*EPS)))
        U(I,2) = 0.75 + 0.25/(1+EXP((-4*X(I)+4*Y(I)-T)/(32*EPS)))
    10 CONTINUE

SUBROUTINE PDEF
    . . .
    DO 10 I = 2, NPTS-1
        RES(I,1) = UT(I,1) -
        + (-U(I,1)*UX(I,1) - U(I,2)*UY(I,1) + EPS*(UXX(I,1)+UYY(I,1))))
        RES(I,2) = UT(I,2) -
        + (-U(I,1)*UX(I,2) - U(I,2)*UY(I,2) + EPS*(UXX(I,2)+UYY(I,2))))
    10 CONTINUE

SUBROUTINE PDEBC
    . . .
    NBNDS = LLBND(0)
DO 10 K = LLBND(1), LLBND(NBNDS+1)-1
I = LBND(K)
RES(I,1) = U(I,1) - 
+ (0.75 - 0.25/((1+EXP((-4*X(I)+4*Y(I)-T)/(32*EPS)))))
RES(I,2) = U(I,2) - 
+ (0.75 + 0.25/((1+EXP((-4*X(I)+4*Y(I)-T)/(32*EPS)))))
10 CONTINUE

To define the initial grid in INIDOM one should store

XL = 0.0
YL = 0.0
XR = 1.0
YU = 1.0
DX = 0.1
DY = 0.1

LROW(0:12) = (11, 1, 4, 15, 26, 37, 46, 57, 68, 79, 88, 97, 106)
IROW(1:11) = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
ICOL(1:105) = (0, 1, 2,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
0, 1, 2, 3, 4, 5, 6, 7, 8

For the boundaries, when numbered as in Fig. 1 the following should be stored in LLBND, ILBND, and LBND

LLBND(0:29) = ( 28,
1, 2, 11, 18, 19, 24, 31, 37, 42, 48, 53, 55, 56, 58,
59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73)
ILBND(1:28) = ( 1, 2, 3, 4, 1, 4,
1, 2, 3, 4,
1, 2, 3, 4,
12, 23, 34, 41, 14, 41,
12, 23, 34, 41,
14, 43, 32, 21)
LBND(1:72) = ( 2,
4, 15, 26, 37, 46, 57, 68, 79, 88,
98, 99, 100, 101, 102, 103, 104,
96,
86, 85, 84, 83, 82,
70, 59, 48, 39, 28, 17, 6,
8, 9, 10, 11, 12, 13,
18, 29, 40, 49, 60,
72, 73, 74, 75, 76, 77,
67, 56, 45, 36, 25,
23 3 3 3 3 3 3 3 34 XX XX
2 ... ... ... ... ... 4 XX XX
2 ... 14 1 1 1 1 1 1 41 XX XX
2 ... 4 23 3 3 3 3 3 3 34
2 ... 4 2 ... ... ... ... ... 4
2 ... 4 2 ... 14 1 1 21 ... 4
2 ... 4 2 ... 4 XX XX 2 ... 4
2 ... 4 2 ... 43 3 3 32 ... 4
2 ... 4 2 ... ... ... ... ... 4
2 ... 4 12 1 1 1 1 1 1 41
12 1 41 XX XX XX XX XX XX XX

Figure 2: Print out by the subroutine PRDOM of the example domain

52, 53,
43,
33, 32
42,
1, 97, 105, 87, 81, 3, 7, 14, 78, 71, 51, 31, 34, 54)

To check if we defined the domain correct we print it out with

```
INTEGER IDOM((NX+1)*(NY+1))
```

```
LLBND(30) = LLBND(29)
CALL PRDOM (IROW, IROW, ICOL, LLBND, ILBND, LBND,
+     IDOM, NX, NY)
```

which prints internal points of the domain as ..., external points as XX and for physical boundary points their ILBND value. The results are shown in Fig. 2.

To restart the computation we have to read the information from file and call VLUGR2 with MNTR = 1. In this run we use the default parameters. Note that the old values for T, DT, XL, YL, XR, YU, DX, and DY will be taken.

C Continuation call of VLUGR2
```
TOUT = 3.0
TOLS = 0.1
TOLT = 0.05
INFO(1) = 0
MNTR = 1
```

```
OPEN(UNIT=LUNDMP,FILE='DUMP',FORM='UNFORMATTED')
CALL RDDUMP (LUNDMP, RWK, LENRWK, IWK, LENIWK)
CLOSE(LUNDMP)
```

C C call main routine
```
CALL VLUGR2 (NPDE, T, TOUT, DT, XL, YL, XR, YU, DX, DY,
+     TOLS, TOLT, INFO, RINFO, RWK, LENRWK, IWK, LENIWK, LWK, LENLWK,
For this run we want to use the exact partial derivatives \( \partial F / \partial U_p \) and therefore we have overloaded DERIVF with our own subroutine of which contents we give a survey below.

**SUBROUTINE DERIVF**

... 

C 

C A0: coefficient of \( U_{n+1} \) in time derivative 

C

C Loop over the components of the (derivatives of) \( U \)

C IC = 1

C

dF(U,Ut)/dU_ic

DO 20 IPT = 1, NPTS

FU(IPT,1,IC) = A0 - (-UX(IPT,1))

FU(IPT,2,IC) = - (-UX(IPT,2))

20 CONTINUE

C

C dF(Ux)/dUx_ic

DO 40 IPT = 1, NPTS

FUX(IPT,1,IC) = - (-U(IPT,1))

FUX(IPT,2,IC) = 0.0

40 CONTINUE

...

C IC = 2

C

C dF(U,Ut)/dU_ic

DO 120 IPT = 1, NPTS

FU(IPT,1,IC) = - (-UY(IPT,1))

FU(IPT,2,IC) = A0 - (-UY(IPT,2))

120 CONTINUE

...

C

C dF(Uyy)/dUyy_ic

DO 180 IPT = 1, NPTS

FUY(IPT,1,IC) = 0.0

FUY(IPT,2,IC) = - (EPS)

180 CONTINUE

C

C Correct boundaries (incl. the internal); all Dirichlet so \( F_U = 0.0 \)

NBND = LLBND(0)

DO 100 LB = LLBND(1), LLBND(NBND+2)-1

IPT = LBND(LB)

FU(IPT,1,1) = 1.0

FU(IPT,1,2) = 0.0

FU(IPT,2,1) = 0.0

FU(IPT,2,2) = 1.0

FUX(IPT,1,1) = 0.0
Figure 3: Grids for the example problem at $t = 1.0$ and at $t = 3.0$.

\[
\text{FYU} (\text{IPT},2,2) = 0.0
\]

100 CONTINUE

In Fig. 3 the generated grids after the final times of the first and the second program are shown. Recall that in the first run refinement up to order 3 is forced at the lower-right corner of the physical domain. The corresponding accuracy is, measured in the maximum norm, 0.01 at $t = 1.0$ and 0.08 at $t = 3.0$. Note that in the second run the number of grid levels is restricted to 3 which affects of course the accuracy.

5. Numerical experiments
In this section we discuss the test results obtained with VLUGR2 for the groundwater-flow problems that were also used as examples in [14, 1].

5.1 Problem description: the 2D fluid-flow / salt-transport problem
In [14, 1] we consider a model for a non-isothermal, single-phase, two-component saturated flow problem which consists of 3 PDEs basic to groundwater flow: the continuity equation, the transport equation and the temperature equation. For the background of these equations we refer to [14]. We here present the model in non-conservative form. As independent variables we have the pressure $p$, the salt mass fraction $\omega$, and the temperature $T$. The continuity equation for the fluid, the salt transport equation, and the temperature equation are given by

\[
n\rho \left( \frac{\partial \rho}{\partial t} + \frac{\partial \omega}{\partial t} + \alpha \frac{\partial T}{\partial t} \right) + \nabla \cdot (\rho \mathbf{q}) = 0,
\]

\[
n\rho \frac{\partial \omega}{\partial t} + \rho \mathbf{q} \cdot \nabla \omega + \nabla \cdot (\rho \mathbf{J}^\omega) = 0,
\]

\[
n\rho \frac{\partial T}{\partial t} + \rho \mathbf{q} \cdot \nabla T + \nabla \cdot (\rho \mathbf{J}^T) = 0,
\]
\[ c^m \rho^m \frac{\partial T}{\partial t} + \rho c_\text{q} \cdot \nabla T + \nabla \cdot (J^T) = 0, \]  
(5.1c)

where \( n \) is the porosity parameter of the porous medium, \( \beta \) a compressibility coefficient, \( \gamma \) a salt coefficient, \( \alpha \) a temperature coefficient, \( c^m \rho^m \) a density expression satisfying \( c^m \rho^m = n \rho_p + (1-n) \rho_s \), and \( c \) the specific heat capacity of the porous medium. Darcy’s law gives the equation for the fluid velocity \( \textbf{q} = (q_1, q_2)^T \)

\[ \textbf{q} = -\frac{k}{\mu} (\nabla p - \rho g), \]  
(5.2)

with \( g \) 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

\[ \rho = \rho_0 \exp[\alpha(T - T_0) + \beta(p - p_0) + \gamma \omega], \]  
(5.3)

\[ \mu = \mu_0(T) \cdot m(\omega), \quad m(\omega) = 1 + 1.85 \omega - 4.0 \omega^2, \]  
(5.4)

where \( \rho_0 \) is the reference density of fresh water, \( p_0 \) a reference pressure, \( T_0 \) a reference temperature, and \( \mu_0(T) \) a possibly temperature-dependent reference viscosity. The equation for the salt-dispersion flux vector is given by Fick’s law

\[ \textbf{J}^s = -n \textbf{D} \nabla \omega, \]  
(5.5)

with the dispersion tensor \( \textbf{D} \) for the solute salt defined as

\[ n \textbf{D} = (n D_{\text{mol}} + \alpha T |\textbf{q}|) \textbf{I} + \frac{\alpha_L - \alpha_T}{|\textbf{q}|} \textbf{qq}^T, \quad |\textbf{q}| = \sqrt{\textbf{q}^T \textbf{q}}. \]  
(5.6)

The coefficients \( D_{\text{mol}}, \alpha_T \) and \( \alpha_L \) correspond, respectively, with the molecular diffusion and the transversal and longitudinal dispersion. Finally, the heat-flux vector \( \textbf{J}^T \) is given by

\[ \textbf{J}^T = -\textbf{H} \nabla T, \]  
(5.7)

where \( \textbf{H} \), the heat conductivity tensor of the porous medium, is defined as

\[ \textbf{H} = (\kappa + \lambda_T |\textbf{q}|) \textbf{I} + \frac{\lambda_L - \lambda_T}{|\textbf{q}|} \textbf{qq}^T. \]  
(5.8)

The parameter \( \kappa \) is the coefficient of heat conductivity and \( \lambda_T \) and \( \lambda_L \) are, respectively, the transversal and longitudinal heat conductivity coefficients.

Our examples are connected with Intraval test case 13 [7]. This laboratory experiment deals with the displacement of fresh water by brine in a thin vertical column filled with a porous medium. Salt water of a high concentration is injected through gates at the bottom of the column giving rise to a fresh-salt water front moving in all directions into the column.

### 5.1.1 Data for Problem I.

In [7] the experiment was carried out under isothermal conditions. We assume non-isothermal conditions and suppose that warm brine is injected. Because the column is very thin, the flow can be considered as being two-dimensional. In our numerical experiment it is supposed that two gates are used for salt water injection so that initially two disjunct salt / fresh water fronts exist which later interact and merge into one front. Due to dispersion the fronts smooth out with time in all directions. For \( t \) sufficiently large the fronts disappear completely which means that the whole medium is filled with the high-salt-concentration fluid.

In Problem I the model is considered on the time-space domain \([0, t_{\text{end}}] \times \Omega \) where the flow domain \( \Omega \) representing the vertical column is the unit square \( \Omega = \{(x, y) | 0 < x, y < 1 \} \). Since \( \Omega \) represents a vertical cross section, the independent variable \( y \) stands for a vertical variable with unit vector pointing upward. Hence the acceleration of gravity vector takes the form \( \textbf{g} = (0, -g)^T \), where \( g \) is the gravity constant. The initial values at \( t = 0 \) at \( \Omega \cup \partial \Omega \) are taken as...
Figure 4: Distribution of salt concentration at $t = 500, 5000, 10000$ and $20000$ with the corresponding grid configuration.
\[ p(x, y, 0) = p_0 + (1 - y) \rho_0 g, \quad \omega(x, y, 0) = 0, \quad \text{and} \quad T(x, y, 0) = T_0, \quad (5.9) \]

which correspond, respectively, to hydrostatic pressure, fresh water and a non-heated medium. For \( 0 < t \leq t_{end} \) the following boundary conditions are imposed

\[
\begin{align*}
x &= 0, 1, \quad 0 \leq y \leq 1: & q_1 &= 0, \quad \omega_x = 0, \quad T_x = 0, \\
y &= 1, \quad 0 < x < 1: & p &= p_0, \quad \omega_y = 0, \quad T_y = 0, \\
y &= 0, \quad \frac{9}{10} \leq x \leq \frac{10}{10}: & q_2 &= q_c, \quad \omega = \omega_0, \quad T = T_c, \\
y &= 0, \quad 0 < x < \frac{9}{10}, \quad \frac{10}{10} < x < 1: & q_2 &= 0, \quad \omega_y = 0, \quad T_y = 0.
\end{align*}
\]  

The third line is connected with the two gates where the warm brine is injected with a prescribed velocity and concentration. The other conditions are self-evident. All remaining problem data are contained in Table 1.

We have run this example using the following set of numerical parameters

\[ \Delta t_0 = 0.1 \quad \text{and} \quad \Delta x = 0.05, \quad \Delta y = 0.05, \]

\[ \text{TOLS} = 0.1 \quad \text{and} \quad \text{TOLT} = 0.1, \]

\[ \text{UMAX} = (1.1E + 5, 0.25, 292). \]

For all other parameters we used the default choice. Since \( \Delta x \) and \( \Delta y \) are the cell widths of the coarse base grid and MAXLEV = 3 this results in a finest grid size allowed of 1/80. Although this is small enough to avoid wiggles, at most time intervals the refinement is restricted by MAXLEV and not by TOLS.

In Fig. 4 we show the distribution of salt concentration at various time levels with the corresponding grid configuration. The temperature distribution shows a comparable behavior, but less steep and slower in time. The results for all solvers are alike.

### 5.1.2 Data for Problem II.

In the second example we consider the case of total impermeability for part of the flow domain, viz., the region \( \{(x, y) | 0 \leq x \leq 0.5, \quad 0.4 \leq y \leq 0.6 \} \). We also consider the flow now to be isothermal and incompressible (\( \alpha = \beta = 0 \)).

The boundary conditions for the impermeability region are

\[
\begin{align*}
0 \leq x &\leq 0.5, \quad y = 0.4, 0.6: & p_y &= -\rho g, \quad \omega_y = 0, \\
x &= 0.5 & 0.4 < y < 0.6: & p_x &= 0, \quad \omega_x = 0.
\end{align*}
\]  

(5.11)

We have closed the right gate so that salt water is injected only through the left gate. Hence we deal with a single salt / fresh water front which first collides with the region of impermeability and then must flow around it. For this problem we run until steady state is reached (\( \omega \equiv \omega_0 \), i.e., \( t_{end} = 10^5 \). All other parameters, if required, are chosen as in Problem I.

The distribution of the salt concentration at various time levels with the corresponding grid configuration is shown in Fig. 5.
Figure 5: Distribution of salt concentration at $t = 10000, 20000, 30000$ and $100000$ with the corresponding grid configuration.
<table>
<thead>
<tr>
<th>Solver preconditioner</th>
<th>BiCGStab ILU</th>
<th>GCRO diagonal</th>
<th>GCRO block-diag</th>
<th>matrix-free GCRO block-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td># time steps</td>
<td>197</td>
<td>197</td>
<td>197</td>
<td>207</td>
</tr>
<tr>
<td># Newton it./level</td>
<td>396</td>
<td>396</td>
<td>396</td>
<td>430</td>
</tr>
<tr>
<td>2</td>
<td>396</td>
<td>396</td>
<td>396</td>
<td>440</td>
</tr>
<tr>
<td>3</td>
<td>396</td>
<td>396</td>
<td>396</td>
<td>472</td>
</tr>
<tr>
<td># Lin. sys. it. per</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton it., level</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 1</td>
<td>1045</td>
<td>8962</td>
<td>8894</td>
<td>9960</td>
</tr>
<tr>
<td>1, 2</td>
<td>763</td>
<td>14691</td>
<td>14558</td>
<td>15811</td>
</tr>
<tr>
<td>1, 3</td>
<td>746</td>
<td>15978</td>
<td>15372</td>
<td>20239</td>
</tr>
<tr>
<td>2, 1</td>
<td>278</td>
<td>7005</td>
<td>6792</td>
<td>8304</td>
</tr>
<tr>
<td>2, 2</td>
<td>429</td>
<td>7135</td>
<td>7066</td>
<td>12857</td>
</tr>
<tr>
<td>2, 3</td>
<td>467</td>
<td>6978</td>
<td>7025</td>
<td>17117</td>
</tr>
<tr>
<td>3, 1</td>
<td>66</td>
<td>66</td>
<td>273</td>
<td></td>
</tr>
<tr>
<td>3, 2</td>
<td></td>
<td></td>
<td>596</td>
<td></td>
</tr>
<tr>
<td>3, 3</td>
<td></td>
<td></td>
<td>3721</td>
<td></td>
</tr>
<tr>
<td>4, 1</td>
<td></td>
<td></td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>4, 2</td>
<td></td>
<td></td>
<td>389</td>
<td></td>
</tr>
<tr>
<td>4, 3</td>
<td></td>
<td></td>
<td>736</td>
<td></td>
</tr>
<tr>
<td>5, 2</td>
<td></td>
<td></td>
<td>82</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Integration history for Problem I.

<table>
<thead>
<tr>
<th>Solver preconditioner</th>
<th>BiCGStab ILU</th>
<th>GCRO diagonal</th>
<th>GCRO block-diag</th>
<th>matrix-free GCRO block-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td># time steps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Newton it./level</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>306</td>
<td>305</td>
<td>305</td>
<td>334</td>
</tr>
<tr>
<td>2</td>
<td>614</td>
<td>612</td>
<td>612</td>
<td>718</td>
</tr>
<tr>
<td>3</td>
<td>604</td>
<td>602</td>
<td>602</td>
<td>709</td>
</tr>
<tr>
<td># Lin. sys. it. per</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton it., level</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 1</td>
<td>1831</td>
<td>15984</td>
<td>15759</td>
<td>18058</td>
</tr>
<tr>
<td>1, 2</td>
<td>1239</td>
<td>24433</td>
<td>24042</td>
<td>27330</td>
</tr>
<tr>
<td>1, 3</td>
<td>1215</td>
<td>24208</td>
<td>24243</td>
<td>29995</td>
</tr>
<tr>
<td>2, 1</td>
<td>782</td>
<td>10914</td>
<td>10198</td>
<td>14686</td>
</tr>
<tr>
<td>2, 2</td>
<td>899</td>
<td>10553</td>
<td>10083</td>
<td>21131</td>
</tr>
<tr>
<td>2, 3</td>
<td>823</td>
<td>11027</td>
<td>10218</td>
<td>24439</td>
</tr>
<tr>
<td>3, 1</td>
<td></td>
<td></td>
<td>1027</td>
<td></td>
</tr>
<tr>
<td>3, 2</td>
<td></td>
<td></td>
<td>1702</td>
<td></td>
</tr>
<tr>
<td>3, 3</td>
<td></td>
<td></td>
<td>4808</td>
<td></td>
</tr>
<tr>
<td>4, 1</td>
<td></td>
<td></td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>4, 2</td>
<td></td>
<td></td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>4, 3</td>
<td></td>
<td></td>
<td>1032</td>
<td></td>
</tr>
<tr>
<td>5, 1</td>
<td></td>
<td></td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>5, 2</td>
<td></td>
<td></td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>5, 3</td>
<td></td>
<td></td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>6, 3</td>
<td></td>
<td></td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Integration history for Problem II.
5.2 Integration history
In Tables 2 and 3 we give the integration history for Problem I and II, respectively. The number of linear system iterations given under GCRO is the sum of the outer-loop iterations (the GCR process) and the inner-loop iterations (GMRES 'preconditioner'). For both problems the solvers using the Jacobian explicitly are very robust. Note that without preconditioning GCRO, with the default values \(\text{MAXL} = 5\) and \(\text{MAXL} = 20\), most probably would outperform BiCGStab, so the ILU preconditioner does a good job. The diagonal and the block-diagonal preconditioner both take into account the first-order derivatives at the boundaries. The difference in performance between the two is minor, the number of iterations decreases only slightly. For the matrix-free version the difference between these two preconditioners is much clearer. Using block-diagonal preconditioning the matrix-free solver only occasionally suffers from time step rejections due to Newton failures. With diagonal scaling this number is much higher.

5.3 Global performance

<table>
<thead>
<tr>
<th>Problem</th>
<th>BiCGStab ILU</th>
<th>GCRO diagonal</th>
<th>GCRO block-diag</th>
<th>matrix-free GCRO block-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CP sec</td>
<td>Mflop</td>
<td>CP sec</td>
<td>Mflop</td>
</tr>
<tr>
<td>Problem I</td>
<td>scalar</td>
<td>368</td>
<td>16</td>
<td>820</td>
</tr>
<tr>
<td></td>
<td>vector</td>
<td>74</td>
<td>78</td>
<td>138</td>
</tr>
<tr>
<td>Problem II</td>
<td>scalar</td>
<td>220</td>
<td>16</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>vector</td>
<td>50</td>
<td>71</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4: Global performance.

Our performance evaluation was done on a Cray Y-MP with the CF77 compiling system. Scalar results were obtained using `cf77 -Wf"-o nvector"`, and vector results with `cf77 -Zv -Wf"-o aggress"`. The timings were done on 1 processor with 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[6], that gives the hardware performance by program unit.

We first give a global idea of the performance of the different solvers. In Table 4 the CPU time and the Mflop rate is shown for both example problems. It is clear that although the (matrix-free) GCRO solver reaches an almost optimal vector speed, considering that we use indirect addressing, the CPU time is (much) larger than when we use BiCGStab + ILU. This is due to the fact that the number of iterations needed is so much higher.

5.4 Vector results
In this section we will compare the vector performance of the solvers.

Tables 5 and 6 show the vector performance of the five routines that use the most CPU time (accumulated). One can see that the PDE definition on the internal domain (PDEF) vectorizes nicely, as expected, since no exceptions have to be made. The definition of the boundary conditions (PDEBC) run at a lower Mflop rate of about 65, but this routine takes only 1% of the total CPU time. All in all it appears that, with the exception of the preconditioning, VLUG2 + BiCGStab is an efficiently vectorized code. The routines needed for the irregular grid structure and refinement take together less than 5% of the CPU time (in vectorized mode). So the overhead for the grid refinement is negligible.
<table>
<thead>
<tr>
<th>BiCGStab+ILU</th>
<th>GCRO+Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td># calls</td>
<td>Avg time</td>
</tr>
<tr>
<td>ILU backs</td>
<td>8688</td>
</tr>
<tr>
<td>ILU dec</td>
<td>391</td>
</tr>
<tr>
<td>PDEF</td>
<td>11826</td>
</tr>
<tr>
<td>MATVEC</td>
<td>7500</td>
</tr>
<tr>
<td>INJON</td>
<td>1174</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BiCGStab+ILU</th>
<th>GCRO+Block-diag</th>
<th>matrix-free GCRO+Block-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td># calls</td>
<td>Avg time</td>
<td>ACM %</td>
</tr>
<tr>
<td>MATVEC</td>
<td>59931</td>
<td>1.2E-3</td>
</tr>
<tr>
<td>GMRES</td>
<td>3485</td>
<td>1.3E-2</td>
</tr>
<tr>
<td>PDEF</td>
<td>11826</td>
<td>1.1E-3</td>
</tr>
<tr>
<td>Bl. backs</td>
<td>61119</td>
<td>1.3E-4</td>
</tr>
<tr>
<td>INJON</td>
<td>1174</td>
<td>1.5E-3</td>
</tr>
</tbody>
</table>

Table 5: Vector performance of top 5 routines for Problem I.
ACM %: Accumulated percentage of CPU-time spent

<table>
<thead>
<tr>
<th>BiCGStab+ILU</th>
<th>GCRO+Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td># calls</td>
<td>Avg time</td>
</tr>
<tr>
<td>ILU backs</td>
<td>15385</td>
</tr>
<tr>
<td>PDEF</td>
<td>12326</td>
</tr>
<tr>
<td>ILU dec</td>
<td>880</td>
</tr>
<tr>
<td>MATVEC</td>
<td>13632</td>
</tr>
<tr>
<td>INJON</td>
<td>1710</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BiCGStab+ILU</th>
<th>GCRO+Block-diag</th>
<th>matrix-free GCRO+Block-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td># calls</td>
<td>Avg time</td>
<td>ACM %</td>
</tr>
<tr>
<td>GMRES</td>
<td>5380</td>
<td>8.1E-3</td>
</tr>
<tr>
<td>MATVEC</td>
<td>94684</td>
<td>3.7E-4</td>
</tr>
<tr>
<td>PDEF</td>
<td>12270</td>
<td>6.2E-4</td>
</tr>
<tr>
<td>Bl. backs</td>
<td>96442</td>
<td>4.1E-5</td>
</tr>
<tr>
<td>GCR</td>
<td>1758</td>
<td>1.1E-3</td>
</tr>
</tbody>
</table>

Table 6: Vector performance of top 5 routines for Problem II.
ACM %: Accumulated percentage of CPU-time spent
as long as we need to solve our problems with an implicit time integrator.

6. SUMMARY
In this paper we described VLUGR2, a vectorizable Method of Lines solver based on a Local Uniform Grid Refinement method for systems of time-dependent PDEs in two space dimensions. Our code offers three different solvers for the nonlinear systems. In the first two the Jacobian of the Newton process is approximated and stored. The linear solvers are BiCGStab[17] combined with standard ILU preconditioning and GCRO[8] with (block) diagonal scaling. The third nonlinear solver is a matrix-free Newton process with as linear solver GCRO. An advantage of a matrix-free solver is that tailor-made space-discretization schemes, resulting in other couplings than the here used 9-point stencil, can be more easily implemented. For a large number of PDE components its memory use can also be profitable. However it is less robust and generally uses more CPU time even though the vector performance of this solver is very good.

We discussed the performance of VLUGR2 for two groundwater-flow problems. The LUGR method proves to be robust and efficient with respect to the location of the refined grids. The nonlinear solvers that make explicit use of the Jacobian both are reliable. BiCGStab with as preconditioner the vectorized implementation of the ILU decomposition of the Jacobian is for these problems computationally the most efficient. Since in two dimensions memory requirements are often not the bottleneck, we strongly advocate the use of this solver.

REFERENCES
8. E. de Sturler and D.R. Fokkema. Nested Krylov methods and preserving the orthogonality. Preprint NR. 796, Department of Mathematics, University of Utrecht, the Netherlands, 1993.

