Algorithm 759: VLUGR3: A Vectorizable Adaptive-Grid Solver for PDEs in 3D—Part II. Code Description

J. G. BLOM and J. G. VERWER
CWI, Amsterdam

This article describes an ANSI Fortran 77 code, VLUGR3, autovectorizable on the Cray Y-MP, that is based on an adaptive-grid finite-difference method to solve time-dependent three-dimensional systems of partial differential equations.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Fortran;* G.1.8 [**Numerical Analysis**]: Partial Differential Equations; G.4 [**Mathematical Software**]

General Terms: Algorithms

Additional Key Words and Phrases: Adaptive-grid methods, iterative solvers, method of lines, nonsymmetric sparse linear systems, partial differential equations, software, vectorization

1. INTRODUCTION

In Trompert [1994], Trompert and Verwer [1991; 1993a; 1993b], Trompert et al. [1993], and Verwer and Trompert [1992; 1993] an adaptive-grid finite-difference method is studied to solve time-dependent two-dimensional systems of partial differential equations (PDEs). Trompert [1992] developed a research code, MOORKOP, that uses the Method of Lines (MoL) approach combining an implicit time-stepping method with the adaptive-grid algorithm. Based on MOORKOP an ANSI Fortran 77 code, VLUGR2 [Blom and Verwer 1993b] was developed with the intention to use it on a vector processor. We extended this method to the 3D case, and in this article we describe the resulting ANSI Fortran 77 code, VLUGR3, that implements a vectorizable adaptive-grid solver for time-dependent three-dimensional systems of PDEs on an arbitrary domain bounded by right-angled polyhedrons.

This work was supported by CRAY Research, Inc., under grant CRG 93.03, via the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF).
Authors' address: Department of Numerical Mathematics, M267, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands; email: gollum@cwi.nl.

In Section 2 we define the problem class of VLUGR3. Section 3 is devoted to a short survey of the algorithm. For a more comprehensive report on the algorithmic aspects we refer to the companion paper [Blom and Verwer 1994b]. In Section 4 we discuss the implementation, viz., the user-relevant part of the data structure and the storage requirements needed. Finally, in Section 5 we describe how to use VLUGR3, enlightened by an elaborated example problem.

2. PDE DEFINITION

VLUGR3 has been designed to solve initial boundary value problems that fit in the following master equation

$$\mathcal{F}(t, x, y, z, \mathbf{u}, \mathbf{u}_t, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z, \mathbf{u}_{xx}, \mathbf{u}_{yy}, \mathbf{u}_{zz}, \mathbf{u}_{xy}, \mathbf{u}_{xz}, \mathbf{u}_{yz}) = 0,$$
$$(x, y, z) \in \Omega, \quad t > t_0, \quad (2.1)$$

where the solution $\mathbf{u}$ may be a vector and the domain Ω an arbitrary "brick-structured" domain, i.e., a domain that can be described by right-angled polyhedrons. The boundary conditions belonging to system (2.1) are formulated as

$$\mathcal{B}(t, x, y, z, \mathbf{u}, \mathbf{u}_t, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z) = 0, \quad (x, y, z) \in \partial\Omega, \quad t > t_0, \quad (2.2)$$

and the initial conditions satisfy

$$\mathbf{u}(t_0, x, y, z) = \mathbf{u}_0(x, y, z), \quad (x, y, z) \in \Omega \cup \partial\Omega. \quad (2.3)$$

Although the problem class (2.1) is quite large, it should be mentioned that VLUGR3 is tuned for time-dependent parabolic PDEs. The time integration is done implicitly, and for the spatial discretization, central finite differences are employed.

3. THE ALGORITHM

In this section we give a short survey of the algorithmic aspects of the code. The implementation of the time integration, and in particular the solution of the nonlinear systems, has changed in comparison with the implementation of Blom and Verwer [1993a]. Here, we discuss the parameters of VLUGR3, that can be set by the user. For further information about the strategic choices we refer to the companion paper [Blom and Verwer 1994b]. Note that the time integration in the final version of VLUGR2 [Blom et al. 1994] is also implemented along the lines described in Blom and Verwer [1994b].

3.1 Outline of the LUGR Algorithm

The concept of Local Uniform Grid Refinement is 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. If necessary, a next level of refinement is created. If the required accuracy in space is reached, the fine-grid solution values are injected in the coinciding coarser-grid nodes. Then the time integration error is estimated.

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 into eight equal parts, so we do not apply semirefinement, 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.

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

$$\max_{ic=1,\mathrm{NPDE}} \mathrm{SPCTOL}(ic) \cdot (|\Delta x^2 \cdot u_{xx}^{ic}(i, j, k)| + |\Delta y^2 \cdot u_{yy}^{ic}(i, j, k)| + |\Delta z^2 \cdot u_{zz}^{ic}(i, j, k)|) \quad (3.1)$$

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

$$\mathrm{SPCTOL}(ic) := \frac{\mathrm{SPCWGT}(ic)}{\mathrm{UMAX}(ic) \cdot \mathrm{TOLS}}. \quad (3.2)$$

The variables on the right-hand side of (3.2) are user-specified quantities. Specifically, $0 \leq \mathrm{SPCWGT} \leq 1$ 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.

Finally, VLUGR3 offers the user the possibility to enforce grid refinement in a priori selected regions (see the description of the CHSPCM subroutine in Section 5.1).

3.2 Integration in Time

To solve the system of PDEs we use the Method of Lines (MoL) approach. The PDEs are discretized in space, and the resulting system of ODEs or DAEs is solved in time with the second-order two-step implicit BDF method with variable step sizes. The time integration is controlled by a solution monitor value, the maximum over all existing grid levels of

$$\|\Delta t \mathbf{u}_t\|_w, \quad (3.3)$$

where Δt is the current time step size, and $\mathbf{u}_t$ is approximated by first-order finite differences. The monitor is measured in a weighted root-mean-square

norm

$$\|\mathbf{v}\|_w = \|W\mathbf{v}\|_2, \tag{3.4}$$

where W is a diagonal matrix defined by

$$W = 1/\sqrt{N}\ \mathrm{diag}(w_1, \ldots, w_N). \tag{3.5}$$

The entries w_i of the diagonal matrix W are defined by

$$w_{ipt,ic} = \mathsf{TIMWGT}(ic)/(\mathsf{ABSTOL}(ic) + |U^{n+1}_{ipt,ic}| \cdot \mathsf{RELTOL}(ic)),$$
$$ipt \in \Omega \quad \text{and} \quad ic = 1, \mathsf{NPDE}, \tag{3.6}$$

with

$$\mathsf{ABSTOL}(ic) = 0.01 \cdot \mathsf{TOLT} \cdot \mathsf{UMAX}(ic) \quad \text{and} \quad \mathsf{RELTOL}(ic) = \mathsf{TOLT}. \tag{3.7}$$

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

The resulting system of nonlinear equations is solved with a Newton process combined with a preconditioned iterative linear solver. The stopping criterion for the Newton process reads as

$$\frac{\rho}{1-\rho}\,\|\mathbf{U}^{(k)} - \mathbf{U}^{(k-1)}\|_w < \mathsf{TOLNEW}, \tag{3.8}$$

where ρ is an approximation of the convergence rate. The entries w_i of the diagonal matrix W in (3.5) are defined by

$$w_{ipt,ic} = 1.0/(\mathsf{ATOL}(ic) + |U^{(0)}_{ipt,ic}| \cdot \mathsf{RTOL}(ic)),$$
$$ipt \in \Omega \quad \text{and} \quad ic = 1, \mathsf{NPDE}, \tag{3.9}$$

with

$$\mathsf{ATOL}(ic) = 0.01 \cdot \mathsf{TOL} \cdot \mathsf{UMAX}(ic) \quad \text{and} \quad \mathsf{RTOL}(ic) = \mathsf{TOL};$$
$$\mathsf{TOL} = 0.1\ \min(\mathsf{TOLT}^2, \mathsf{TOLS}). \tag{3.10}$$

The stopping criterion for the linear solver is

$$\|P^{-1}r^{(l)}\|_w < \mathsf{TOLLSS}/2^k, \tag{3.11}$$

where k is the current Newton-iteration index, and P^{-1} is the preconditioner in use. Both stopping criteria can be influenced easily by the user by changing the value of TOLNEW (default 1.0) and TOLLSS (TOLNEW/10) in

parameter statements. The maximum number of iterations in the nonlinear and linear solver can also be adapted in this way.

VLUGR3 offers two different iterative nonlinear solvers. The nonlinear systems can be solved with modified Newton and the linear systems with BiCGStab [van der Vorst 1992] with ILU preconditioning or with matrix-free Newton with GCRO [De Sturler and Fokkema 1993] as linear solver and (block) diagonal scaling as preconditioner. In the first the Jacobian matrix is computed by numerical differencing, once per time step, and stored. For this solver, which makes explicit use of the Jacobian, we developed in Blom and Verwer [1994a] a vectorizable implementation of the matrix-vector multiply and the ILU preconditioning routines written especially for systems of PDEs in 2D discretized on a nine-point stencil and on a grid that is bounded by arbitrary right-angled polygons. In VLUGR3 these vectorizable modules have been implemented in an analogous way. In the second option 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. In this case the matrix-vector multiply vectorizes equally well as the residual evaluation. This solver also allows an easy change of the space discretization scheme resulting in other couplings than the 19-point stencil used here.

In the documentation of the enveloping routine VLUGR3 some guidelines are given on how to replace the space discretization module or how to implement a different linear solver.

4. THE CODE

4.1 Data Structure

To achieve a good vector performance, pointers to the boundary points in the grid structure are available so that the computation of the PDE can be performed in two "sweeps." In the first sweep, using direct addressing, the PDE residual is computed over the whole space 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 data structure 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 data structure, 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. 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 solution at a specific grid level is stored along horizontal planes, and each plane is stored rowwise, one component vector after the other. Solutions from three 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.

A grid at a specific grid level is stored in the following data structure:

—LPLN: the actual number of horizontal planes, the pointers to the start of a plane in LROW, and the number of rows in the grid.

—IPLN: the plane number of a plane in the virtual box enclosing the physical domain.

—LROW: the pointers to the start of a row in the grid, and the number of grid points.

—IROW: the row number of a row in the virtual box enclosing the physical domain.

—ICOL: the column number of a grid point in the virtual box enclosing the physical domain.

—LLBND: the total number of physical boundary planes in the actual domain, the pointers to the start of a specific boundary plane in LBND, and the number of boundary points in LBND. Note that edges and corners are stored for each plane they belong to.

—ILBND: the type of the boundaries
1: left plane,
2: down plane,
3: right plane,
4: up plane,
5: front plane, and
6: back plane.

—LBND: the pointers to the boundary points in the actual grid structure.

—LBLWY: a pointer to the node below in Y-direction in the actual grid or 0, if the index node is a front-plane boundary point.

—LABVY: a pointer to the node above in Y-direction in the actual grid or 0, if the index node is a back-plane boundary point.

—LBLWZ: a pointer to the node below in Z-direction in the actual grid or 0, if the index node is a down-plane boundary point.

—LABVZ: a pointer to the node above in Z-direction in the actual grid or 0, if the index node is an up-plane boundary point.

All grids from three different time levels have to be saved. For the base grid all information is saved, i.e., the arrays containing the grid information and the pointer arrays needed for the Jacobian and for the hyperplane ordering. The latter is done even if the matrix-free option is chosen, to facilitate a restart using the other solver. For the higher-level grids only the first five arrays (LPLN, IPLN, LROW, IROW, ICOL) are saved.

The above-mentioned arrays and work space for the linear 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 VLUGR3.

4.2 Data Storage Requirements

The Cray Y-MP system does not have much memory hierarchy, viz., registers, main memory, and disks. This means that for normal use one does not have to think much about memory requirements as long as the program and data fit into the main memory. On most other architectures the hierarchy is extended, e.g., with virtual memory, with the consequence that the main memory is much smaller. On those systems the real-time performance is often not determined by the floating-point operations but by memory operations. Therefore we consider it also of importance to keep the data storage small in amount and compact.

VLUGR3 requires roughly the following amount of memory

$$\text{INTEGER: } 7 \cdot \text{MAXLEV} \cdot \text{NPTS}_{av} + 7 \cdot \text{NPTS}_{max} + \text{LSSIWK}$$

$$\text{REAL: } 5 \cdot \text{MAXLEV} \cdot \text{NPDE} \cdot \text{NPTS}_{av} + (3 + 13 \cdot \text{NPDE}) \cdot \text{NPTS}_{max} + \text{LSSRWK}$$

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

$$\text{LSSIWK: } 19 \cdot \text{NPTS}_{max}$$

$$\text{LSSRWK: } 38 \cdot \text{NPDE} \cdot \text{NPTS}_{max} \cdot \text{NPDE}.$$

When using GCRO, LSSIWK = 0, and the real work space is dependent on the choice of the preconditioner:

GCRO+ block-diagonal scaling

$$\text{LSSRWK: } \text{NPTS}_{max} \cdot \text{NPDE} \cdot (\max(\text{NPDE} \cdot 7 + 3,\ 2 \cdot \text{MAXLR} + \text{MAXL} + 6) + \text{NPDE}),$$

GCRO+ diagonal scaling

$$\text{LSSRWK: } \text{NPTS}_{max} \cdot \text{NPDE} \cdot (2 \cdot \text{MAXLR} + \text{MAXL} + 7).$$

In the above, MAXLR is the maximum number of outer GCR iterations, and MAXL is the maximum number of inner GMRES iterations. These values can be adapted via parameter statements. For the default values of MAXLR(5) and MAXL(20) the real work storage needed in both solvers is for a scalar PDE alike, and for a system of PDEs the gain is approximately $38 \cdot \text{NPDE} \cdot \text{NPTS}_{max} \cdot (\text{NPDE} - 1)$.

The work storage required will be checked against the user-defined work space. If the work space is too small a message is printed with the needed amount given.

5. HOW TO USE VLUGR3

We have deliberately chosen to keep the use of VLUGR3 as simple as possible, i.e., most method parameters are set in parameter statements in the code itself rather than letting the user specify them. 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 should be changed in the code.

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

—the number of PDEs,

—the initial and final time and the initial step size,

—the left/front/down and the right/back/up corner of the domain and the initial grid width in the x-, y-, and z-directions, respectively,

—the space and time tolerance TOLS and TOLT as used in the monitors (3.1) and (3.3), and

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

In this case the parameters SPCWGT, TIMWGT, and UMAX (see (3.2) and (3.6)) are set to their default value 1.

If the initial domain is not a true rectangular prism, a virtual box 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 (see Section 5.1).

5.1 General Description

The calling sequence of VLUGR3 is

```
      SUBROUTINE VLUGR3 (NPDE, T, TOUT, DT,
     +  XL, YF, ZD, XR, YB, ZU, DX, DY, DZ,
     +  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, YF, ZD): coordinate of left/front/down corner of domain (a rectangular prism)

(XR, YB, ZU): coordinate of right/back/upper corner of domain

DX: cell width in x-direction of base grid

DY: cell width in y-direction of base grid

DZ: cell width in z-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 $\approx (26 + 38 \cdot \text{NPDE}) \cdot \text{NPTS} \cdot \text{NPDE}$, with NPTS the expected maximum number of points on a grid

IWK(LENIWK): integer workspace, LENIWK $\approx 40 \cdot$ NPTS

LWK(LENLWK): logical workspace, LENLWK $\approx 2 \cdot$ NPTS

MNTR: MNTR = 0, first call of VLUGR3 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). Note, that PDEBC will be called after PDEF, which implies that in PDEF all grid points can be handled as if they were internal grid points.

If one wants to continue the integration after returning from VLUGR3, one should set MNTR to 1. All parameters can be changed although T, DT, XL, YF, ZD, XR, YB, ZU, DX, DY, and DZ will be overwritten with their old values. The contents of the work arrays RWK and IWK should not be altered.

A more sophisticated use of VLUGR3 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 parentheses 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 rectangular prism; 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)

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): LUNNLS (0), the logical unit number of the file where information on the Newton process will be written. If LUNNLS = 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): SPCWGT ((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): TIMWGT ((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 overloading 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 a rectangular prism.
—DERIVF, 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 data structure:

—PRDOM, to print the domain one has defined with INIDOM.
—SETXYZ, to get the x-, y- and, z-coordinates corresponding with the grid points.
—PRSOL, to print the solution and the corresponding coordinate values at all grid levels.

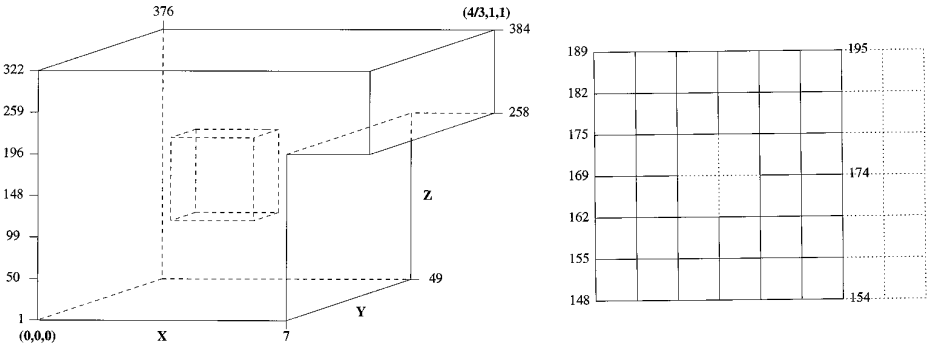

Fig. 1. Domain (left) and slice at $z = 0.5$ (right) with node numbering.

—WRUNI, to write to file the (interpolated) solution values on a uniform grid of a specified grid level and the maximum grid level used in each point.

—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 VLUGR3.

5.2 Example Problem

Our example problem is the three-dimensional Burgers' system

$$u_t + uu_x + vu_y + wu_z = \varepsilon \Delta u,$$

$$v_t + uv_x + vv_y + wv_z = \varepsilon \Delta v,$$

$$w_t + uw_x + vw_y + ww_z = \varepsilon \Delta w. \tag{5.1}$$

on a domain that is the unit cube with a projection at $[1, \frac{4}{3}] \times [0, 1] \times [\frac{2}{3}, 1]$ and a hole inside at $([\frac{1}{3}, \frac{2}{3}]^3)$ (Figure 1). On the boundaries $\partial\Omega$ we prescribe Dirichlet conditions. An exact solution is given by (see Blom and Verwer [1994b])

$$u = \frac{f_1 + (1/2) f_2}{f_1 + f_2} = 1 - \frac{1}{2} \frac{1}{1 + \exp[(-x + y + z - (3/4)t)/(4\varepsilon)]}, \tag{5.2a}$$

$$v = w = \frac{3}{2} - u. \tag{5.2b}$$

The solution represents a wave front at $x - y - z = -\frac{3}{4}t$. The speed of propagation is perpendicular to the wave front. We solve this problem at the time interval [0.0, 2.0] and with $\varepsilon = 5.10^{-3}$.

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.

In the first call of VLUGR3 the initial grid is formulated in the INIDOM routine; a refinement is enforced up to level 3 at the point (1.0, 0.5, 0.0) 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 the file. In that run we use the default parameter settings, and we overload DERIVF with our own version which stores the exact partial derivatives $\partial \mathbf{F}/\partial \mathbf{U}_p$.

We cover the domain by a virtual box ((0.0, 0.0, 0.0), (⅔, 1.0, 1.0)) and make a virtual base grid of $8 \times 6 \times 6$ grid cells (Figure 1). For the first part of the time interval [0, 1.0] the user program will contain the following:

```
MNTR = 0
NPDE = 3
T    = 0.0
TOUT = 1.0
DT   = 0.001

C Since domain is not a rectangular prism the grid parameters need not to be

C specified here (cf. INIDOM)

TOLS = 0.1
TOLT = 0.1
INFO(1) = 1

C MAXLEV
INFO(2) = 4

C Domain not a rectangular prism
INFO(3) = 1

C Linear system solver: GCRO + 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 GCRO 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
                RINFO(5) = 1.0
C SPCWGT = 1.0
                RINFO(6) = 1.0
                RINFO(7) = 1.0
                RINFO(8) = 1.0
C TIMWGT = 1.0
                RINFO(9) = 1.0
                RINFO(10) = 1.0
                RINFO(11) = 1.0
C
C Call main routine
                CALL VLUGR3 (NPDE, T, TOUT, DT, XL, YF, ZD, XR, YB, ZU, DX, DY,
          +     DZ, TOLS, TOLT, 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 is the PDE defining the routines:

```
      SUBROUTINE PDEIV
        . . .
      DO 10 I = 1, NPTS
        U(I,1) = 1-0.5/(1+EXP((-X(I)+Y(I)+Z(I)-0.75*T)/(4*5.0E-3)))
        U(I,2) = 1.5-U(I,1)
        U(I,3) = 1.5-U(I,1)
   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) + U(I,3)*UZ(I,1) -
     +             5.0E-3*(UXX(I,1)+UYY(I,1)+UZZ(I,1))
        RES(I,2) = UT(I,2) + U(I,1)*UX(I,2) +
```

```
     +          U(I,2)*UY(I,2) + U(I,3)*UZ(I,2) -
     +          5.0E-3*(UXX(I,2)+UYY(I,2)+UZZ(I,2))
       RES(I,3) = UT(I,3) + U(I,1)*UX(I,3) +
     +          U(I,2)*UY(I,3) + U(I,3)*UZ(I,3) -
     +          5.0E-3*(UXX(I,3)+UYY(I,3)+UZZ(I,3))
  10 CONTINUE
     SUBROUTINE PDEBC
      . . .
     NBNDS = LLBND(0)
     DO 10 K = LLBND(1), LLBND(NBNDS+1)-1
       I = LBND(K)
       UI = 1-0.5/(1+EXP((-X(I)+Y(I)+Z(I)-0.75*T)/4*5.0E-3)))
       RES(I,1) = U(I,1) - UI
       RES(I,2) = U(I,2) - (1.5-UI)
       RES(I,3) = U(I,3) - (1.5-UI)
  10 CONTINUE
```

To define the initial grid in INIDOM one should store the following:

```
XL = 0.0
YF = 0.0
ZD = 0.0
XR = 4.0/3.0
YB = 1.0
ZU = 1.0
DX = (XR-XL)/8
DY = (YB-YF)/6
DZ = (ZU-ZD)/6

LPLN(0:8)   =  (   7   1   8  15  22  29  36  43  50)
IPLN(1:7)   =  (   0   1   2   3   4   5   6)
LROW(1:50)  =  (   1   8  15  22  29  36  43,
                  50  57  64  71  78  85  92,
                  99 106 113 120 127 134 141,
                 148 155 162 169 175 182 189,
                 196 205 214 223 232 241 250,
                 259 268 277 286 295 304 313,
                 322 331 340 349 358 367 376,
                 385)
IROW(1:49)  =  (   0   1   2   3   4   5   6,
                                  (7 ×)
                   0   1   2   3   4   5   6)
ICOL(1:384) =  (   0   1   2   3   4   5   6,
                                  (24 ×)
                   0   1   2   3   4   5   6,
                   0   1   2   4   5   6,
                   0   1   2   3   4   5   6,
                   0   1   2   3   4   5   6,
                   0   1   2   3   4   5   6,
```

```
0    1    2    3    4    5    6    7    8,
                (21 ×)
0    1    2    3    4    5    6    7    8)
```

Let the order of the boundary planes be the following, first the outer planes: $x = 0.0$, $z = 0.0$, $x = 1.0$, $z = 2/3$, $x = 4/3$, $z = 1.0$, $y = 0.0$, and $y = 1.0$, and then the boundaries of the hole inside: $x = 2/3$, $z = 2/3$, $x = 1/3$, $z = 1/3$, $y = 2/3$, and $y = 1/3$. Then we store the following in LLBND, ILBND, and LBND:

```
LLBND(0:15) = (14 1 50 99 134 155 176 239 294 349 358 367 376 385 394 )
ILBND(1:14) = (1 2 3 2 3 4 5 6 1 2 3 4 5 6 )
LBND(1:393) = (1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113
  120 127 134 141 148 155 162 169 175 182 189 196 205 214 223 232 241 250 259 268 277 286
  295 304 313 322 331 340 349 358 367 376,
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
  35 36 37 38 39 40 41 42 43 44 45 46 47 48 49,
  49 42 35 28 21 14 7 98 91 84 77 70 63 56 147 140 133 126 119 112 105 195 188 181 174 168
  161 154 256 247 238 229 220 211 202,
  202 203 204 211 212 213 220 221 222 229 230 231 238 239 240 247 248 249 256 257 258,
  258 249 240 231 222 213 204 321 312 303 294 285 276 267 384 375 366 357 348 339 330,
  384 383 382 381 380 379 378 377 376 375 374 373 372 371 370 369 368 367 366 365 364 363
  362 361 360 359 358 357 356 355 354 353 352 351 350 349 348 347 346 345 344 343 342 341
  340 339 338 337 336 335 334 333 332 331 330 329 328 327 326 325 324 323 322,
  1 2 3 4 5 6 7 50 51 52 53 54 55 56 99 100 101 102 103 104 105 148 149 150 151 152 153 154
  196 197 198 199 200 201 202 203 204 259 260 261 262 263 264 265 266 267 322 323 324 325
  326 327 328 329 330,
  49 48 47 46 45 44 43 98 97 96 95 94 93 92 147 146 145 144 143 142 141 195 194 193 192 191
  190 189 258 257 256 255 254 253 252 251 250 321 320 319 318 317 316 315 314 313 384 383
  382 381 380 379 378 377 376,
  117 124 131 166 172 179 218 227 236,
  236 235 234 227 226 225 218 217 216,
  115 122 129 164 171 177 216 225 234,
  115 116 117 122 123 124 129 130 131,
  129 130 131 177 178 179 234 235 236,
  115 116 117 164 165 166 216 217 218 )
```

To check if we defined the domain correctly we print it out with the following:

```
      INTEGER IDOM(0:(NX+1)*(NY+1)*(NZ+1))
C
      LLBND(16) = LLBND(15)
      CALL PRDOM (LPLN, IPLN, LROW, IROW, ICOL, LLBND, ILBND,
     + LBND, IDOM, NX, NY, NZ)
```

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

```
Plane:  0
 ca 26 26 26 26 26 da XX XX
 12  2  2  2  2  2 23 XX XX
 12  2  2  2  2  2 23 XX XX
 12  2  2  2  2  2 23 XX XX
 12  2  2  2  2  2 23 XX XX
 12  2  2  2  2  2 23 XX XX
 aa 25 25 25 25 25 ba XX XX

Plane:  1
 16  6  6  6  6  6 36 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
 15  5  5  5  5  5 35 XX XX

Plane:  2
 16  6  6  6  6  6 36 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. ga 45 ha ..  3 XX XX
  1 .. 34  4 14 ..  3 XX XX
  1 .. ea 46 fa ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
 15  5  5  5  5  5 35 XX XX

Plane:  3
 16  6  6  6  6  6 36 XX XX
  1 .. .. .. .. ..  3 XX XX
  1 .. 35  5 15 ..  3 XX XX
  1 ..  3 XX  1 ..  3 XX XX
  1 .. 36  6 16 ..  3 XX XX
  1 .. .. .. .. ..  3 XX XX
 15  5  5  5  5  5 35 XX XX

Plane:  4
 16  6  6  6  6  6 oa 26 pa
  1 .. .. .. .. .. 32  2 23
  1 .. ma 25 na .. 32  2 23
  1 .. 23  2 12 .. 32  2 23
  1 .. ka 26 la .. 32  2 23
  1 .. .. .. .. .. 32  2 23
 15  5  5  5  5  5 ia 25 ja

Plane:  5
 16  6  6  6  6  6  6  6 36
  1 .. .. .. .. .. .. .. 3
  1 .. .. .. .. .. .. .. 3
  1 .. .. .. .. .. .. .. 3
  1 .. .. .. .. .. .. .. 3
  1 .. .. .. .. .. .. .. 3
 15  5  5  5  5  5  5  5 35

Plane:  6
 sa 46 46 46 46 46 46 46 ta
 14  4  4  4  4  4  4  4 34
 14  4  4  4  4  4  4  4 34
 14  4  4  4  4  4  4  4 34
 14  4  4  4  4  4  4  4 34
 14  4  4  4  4  4  4  4 34
 qa 45 45 45 45 45 45 45 ra

Legenda corners:
 aa: 5 2 1        na: 5 2 1
 ba: 5 3 2        oa: 6 2 3
 ca: 6 2 1        pa: 6 3 2
 da: 6 3 2        qa: 5 4 1
 ea: 6 4 3        ra: 5 4 3
 fa: 6 4 1        sa: 6 4 1
 ga: 5 4 3        ta: 6 4 3
 ha: 5 4 1
 ia: 5 2 3
 ja: 5 3 2
 ka: 6 3 2
 la: 6 2 1
 ma: 5 3 2
```

Fig. 2. Print out by the subroutine PRDOM of the example domain.

To restart the computation we have to read the information from file and call VLUGR3 with MNTR = 1. In this run we use the default parameters. Note that the old values for T, DT, XL, YF, ZD, XR, YB, ZU, DX, DY, and DZ will be taken.

```
C Continuation call of VLUGR3
      MNTR = 1
      TOUT = 2.0
      TOLS = 0.1
      TOLT = 0.1
C Default choices
      INFO(1) = 0
C
```

```
C Read info from file
      OPEN(UNIT=LUNDMP,FILE='DUMP',FORM='UNFORMATTED')
      CALL RDDUMP (LUNDMP, RWK, LENRWK, IWK, LENIWK)
      CLOSE(LUNDMP)
C
C call main routine
      CALL VLUGR3 (NPDE, T, TOUT, DT, XL, YF, ZD, XR, YB, ZU, DX, DY,
     +  DZ, TOLS, TOLT, INFO, RINFO, RWK, LENRWK, IWK, LENIWK,
     +  LWK, LENLWK, MNTR)
```

For this run we want to use the exact partial derivatives $\partial \mathbf{F}/\partial \mathbf{U}_p$, and therefore we have overloaded subroutine DERIVF with our own:

```
      SUBROUTINE DERIVF
      . . .
C A0: coefficient of U_n+1 in time derivative
C
Ccc We first zero all 'FU' arrays and then store the needed values for the
C internal domain
      DO 10 IPT = 1, NPTS
C dF_1(U,Ut)/dU_ic
      FU(IPT,1,1) = UX(IPT,1) + A0
      FU(IPT,1,2) = UY(IPT,1)
      FU(IPT,1,3) = UZ(IPT,1)
C dF_1(Up)/dUp_ic
      FUX(IPT,1,1) = U(IPT,1)
      FUY(IPT,1,1) = U(IPT,2)
      FUZ(IPT,1,1) = U(IPT,3)
C dF_1(Upp)/dUpp_ic
      FUXX(IPT,1,1) = -5.0E-3
      FUYY(IPT,1,1) = -5.0E-3
      FUZZ(IPT,1,1) = -5.0E-3
C dF_2(U,Ut)/dU_ic
      FU(IPT,2,1) = UX(IPT,2)
      FU(IPT,2,2) = UY(IPT,2) + A0
      FU(IPT,2,3) = UZ(IPT,2)
C dF_2(Up)/dUp_ic
      FUX(IPT,2,2) = U(IPT,1)
      FUY(IPT,2,2) = U(IPT,2)
      FUZ(IPT,2,2) = U(IPT,3)
```

```
C dF_2(Upp)/dUpp_ic
            FUXX(IPT,2,2) = −5.0E−3
            FUYY(IPT,2,2) = −5.0E−3
            FUZZ(IPT,2,2) = −5.0E−3

C dF_3(U,Ut)/dU_ic
            FU(IPT,3,1) = UX(IPT,3)
            FU(IPT,3,2) = UY(IPT,3)
            FU(IPT,3,3) = UZ(IPT,3) + A0

C dF_3(Up)/dUp_ic
            FUX(IPT,3,3) = U(IPT,1)
            FUY(IPT,3,3) = U(IPT,2)
            FUZ(IPT,3,3) = U(IPT,3)

C dF_3(Upp)/dUpp_ic
            FUXX(IPT,3,3) = −5.0E−3
            FUYY(IPT,3,3) = −5.0E−3
            FUZZ(IPT,3,3) = −5.0E−3

         10 CONTINUE

C

C Correct boundaries (incl. the internal); all Dirichlet so FUp = 0.0
          NBNDS = LLBND(0)
          DO 100 LB = LLBND(1), LLBND(NBNDS+2)−1
            IPT = LBND(LB)
            DO 110 IC = 1, NPDE
               DO 120 JC = 1, NPDE
                  FU (IPT,IC,JC) = 0.0
                  FUX (IPT,IC,JC) = 0.0
                  FUY (IPT,IC,JC) = 0.0
                  FUZ (IPT,IC,JC) = 0.0
                  FUXX(IPT,IC,JC) = 0.0
                  FUYY(IPT,IC,JC) = 0.0
                  FUZZ(IPT,IC,JC) = 0.0

     120 CONTINUE
               FU(IPT,IC,IC) = 1.0

     110 CONTINUE

     100 CONTINUE
```

In Figure 3 a slice at $y = 0.5$ of the generated grids after the final times of the first and the second program is shown. In the first run four grid levels are used, and refinement up to level three is forced at (1, 0.5, 0). In the second run the number of grid levels is restricted to three. The corresponding accuracy is, measured in the maximum norm over all time and grid levels, 0.06 for the first run and 0.13 for the second.

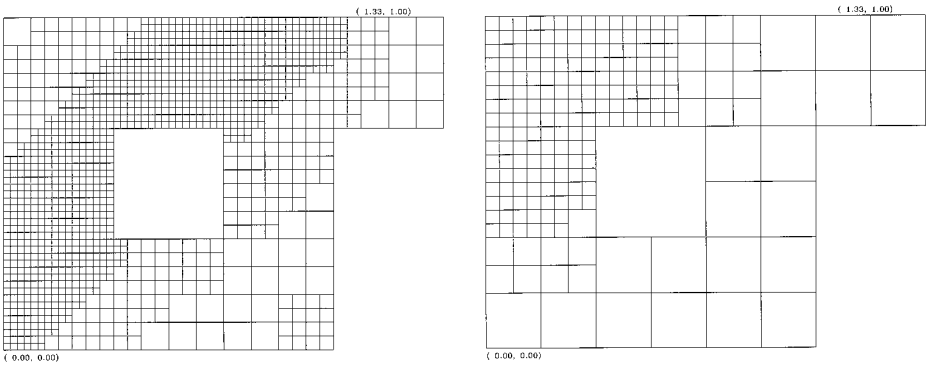

Fig. 3. Slice at $y = 0.5$ of grid at $t = 1.0$ (left) and $t = 2.0$ (right).

REFERENCES

BLOM, J. G. AND VERWER, J. G. 1993a. A vectorizable adaptive grid solver for PDEs in 3D. Rep. NM-R9319, CWI, Amsterdam.

BLOM, J. G. AND VERWER, J. G. 1993b. VLUGR2: A vectorized local uniform grid refinement code for PDEs in 2D. Rep. NM-R9306, CWI, Amsterdam.

BLOM, J. G. AND VERWER, J. G. 1994a. Vectorizing matrix operations arising from PDE discretization on 9-point stencils. *J. Supercomput. 8,* 29–51.

BLOM, J. G. AND VERWER, J. G. 1994b. VLUGR3: A vectorizable adaptive grid solver for PDEs in 3D. Part I. Algorithmic aspects and applications. *Appl. Numer. Math. 16,* 129–156.

BLOM, J. G., TROMPERT, R. A., AND VERWER, J. G. 1994. VLUGR2: A vectorizable adaptive grid solver for PDEs in 2D. Rep. NM-R9403, CWI, Amsterdam. See also this issue.

DE STURLER, E. AND FOKKEMA, D. R. 1993. Nested Krylov methods and preserving the orthogonality. In *NASA Conference Publication 3324.* Vol. 1, the *6th Copper Mountain Conference on Multigrid Methods,* N. D. Melson, T. A. Manteuffel, and S. F. McCormick, Eds. NASA, 111–126.

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

TROMPERT, R. A. 1994. Local uniform grid refinement for time-dependent partial differential equations. Ph.D. thesis, Univ. of Amsterdam, The Netherlands.

TROMPERT, R. A. AND VERWER, J. G. 1991. A static-regridding method for two-dimensional parabolic partial differential equations. *Appl. Numer. Math. 8,* 65–90.

TROMPERT, R. A. AND VERWER, J. G. 1993a. Analysis of the implicit Euler local uniform grid refinement method. *SIAM J. Sci. Comput. 14,* 259–278.

TROMPERT, R. A. AND VERWER, J. G. 1993b. Runge-Kutta methods and local uniform grid refinement. *Math. Comput. 60,* 591–616.

TROMPERT, R. A., VERWER, J. G., AND BLOM, J. G. 1993. Computing brine transport in porous media with an adaptive-grid method. *Int. J. Numer. Meth. Fluids 16,* 43–63.

VERWER, J. G. AND TROMPERT, R. A. 1992. An adaptive-grid finite-difference method for time-dependent partial differential equations. In *Proceedings of the 14th Biennial Dundee Conference on Numerical Analysis,* D. F. Griffiths and G. A. Watson, Eds. Pitman Research Notes in Mathematics Series, vol. 260. Pitman, Harlow, U. K., 267–284.

VERWER, J. G. AND TROMPERT, R. A. 1993. Analysis of local uniform grid refinement. *Appl. Numer. Math. 13,* 251–270.

VAN DER VORST, H. A. 1992. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput. 13,* 631–644.

Received March 1994; revised May 1995; accepted September 1995