A Parallel Software-Platform for Solving Problems of Partial Differential Equations using Unstructured Grids and Adaptive Multigrid Methods

Peter Bastian, Klaus Birken, Klaus Johannsen, Stefan Lang, Volker Reichenberger, Christian Wieners, Gabriel Wittum, and Christian Wrobel

Universität Stuttgart, Institut für Computeranwendungen III,
Pfaffenwaldring 27, D-70569 Stuttgart, Germany

Abstract. The goal of this work is the development of a parallel software-platform for solving partial differential equation problems. State-of-the-art numerical methods have been developed and implemented for the efficient and comfortable solution of real-world application problems. Emphasis is laid on the following topics: distributed unstructured grids, adaptive grid refinement, derefinement/coarsening, robust parallel multigrid methods, various FE and FV discretizations, dynamic load balancing, mapping and grid partitioning. Some important application examples will be presented including structural mechanics, two-phase flow in porous media, Navier-Stokes problems (CFD) and density-driven groundwater flow.

1 Introduction

Over the past two decades, very efficient techniques for the numerical solution of partial differential equations have been developed. Most notably these are:

- use of unstructured meshes for the approximation of complex geometries;
- adaptive local grid refinement in order to minimize the number of degrees of freedom required for a certain accuracy;
- robust multigrid methods for the fast solution of systems of linear equations;
- parallelization of these algorithms on MIMD type machines.

Up to now, these innovative techniques have been implemented mostly in university research codes [2,3,8,13] and only very few commercial codes use them, *e.g.* [14]. The reason for this is twofold. First, the construction of fast and robust iterative solvers is still a problem. Multigrid methods have been applied very successfully in the field of computational fluid dynamics [10,12, 16] but the application to problems from nonlinear structural mechanics or multiphase–flow in porous media is still in its infancy.

The second reason is that the integration of all the above–mentioned techniques in a single code requires a major coding effort of the order of several tens of man–years. Moreover, the structure of existing codes is often not

suited to incorporate all these methods since they require a strong interaction between mesh generator, error estimator, solver and load balancer.

The software package *UG* (shorthand for *U*nstructured *G*rids) has been designed to overcome these problems by providing reusable software tools that simplify the implementation of parallel adaptive multigrid methods on unstructured meshes for complex engineering applications. The heart of *UG* is its unstructured grid data structure. It allows one to create meshes consisting of triangular, quadrilateral, tetrahedral, pyramidal, hexahedral and prism elements in two and three space dimensions. The mesh data structure is hierarchical and elements can be refined and removed locally.

The geometric data structure is complemented by the algebraic data structure used to represent sparse matrices and vectors. The degrees of freedom can be associated with nodes, edges, faces and elements of the mesh, thus also allowing the implementation of nonconforming, mixed or higher–order finite element discretizations. A large number of linear algebra subroutines, iterative kernels and multigrid components is available. For standard situations, like conforming finite elements, the user does not have to write a single line of code in order to use the multigrid method (even for *systems* of partial differential equations). The implementation of discretization schemes is simplified by a large collection of routines providing shape functions and their derivatives, quadrature formulas, finite volume constructions etc.

UG is intended primarily to be a tool to explore new discretization schemes, solvers and error estimators. A powerful graphical user interface can help to reduce development time significantly. *UG* has a built–in shell with command interpreter and allows the user to open any number of windows on his screen. Meshes, contour lines, color plots and vector plots can be displayed in two and three dimensions. Hidden line removal in 3D efficiently uses the hierarchical data representation.

A further great advantage of *UG* is its support for parallelism. The experiences from a first parallel version described in [3] have led to a new programming model *DDD* that can be used for the parallelization of applications with graph–like data structures as described in [6]. *DDD* is the basis of the parallel *UG* version but can also be used independently of it. *UG* will allow a very smooth transition from sequential to parallel computation.

The computational and networking infrastructure at HLRS in Stuttgart has been used for two major goals in the course of the corresponding project:

- During the final development phase of the parallel *UG* software package it was necessary to test functionality and performance of the parallel code on real-world configurations. Especially the Cray T3E with 512 processors at HLRS/Stuttgart provided valuable testing scenarios with high processor numbers and large main memory.
- Due to the hierarchical approach of *UG*'s software design, it was possible to develop and test a variety of applications on a small number of processors of a smaller (and cheaper!) parallel platform (*e.g.*, the Intel Paragon

at RUS/Stuttgart) or even on a single workstation using the sequential build of the code. After that phase, performance tests and – most important – real-world application problems require powerful high-performance platforms, as those provided by HLRS.

The remainder of this paper is organized as follows. The next section gives a rough overview of the parallel software platform developed in the corresponding project. In the third section exemplary applications are presented in order to show the power of this approach. Quality and efficiency of the parallel applications are demonstrated by performance figures. The paper ends with a short conclusion.

2 Parallel Software Architecture

A large software system like *UG* is usually described at a number of different levels of abstraction. In this section, we move through this hierarchy from top to bottom. *UG* knows three design levels which are called *architectural design*, *subsystem design* and *component design*.

At least on the architecture and subsystem level, *UG* is a modular design and the information hiding principle is used extensively. All state information is distributed among the subsystems. *UG* is implemented mostly in the ANSI C programming language, some parts have been implemented in C++.

2.1 Architecture Design

The highest level of abstraction in *UG* is the architecture design level. Its decomposition is motivated as follows:

UG Library
: The *UG* library is *completely independent* of the partial differential equation to be solved. It provides the geometric and algebraic data structures and a huge number of mesh manipulation options, numerical algorithms, visualization techniques and the user interface.

Problem Class Libraries
: This part provides discretization, error estimator and, if required, non–standard solvers for a particular set of partial differential equations.

Applications
: The application finally provides the domain description, boundary conditions and coefficient functions in order to complete the problem description. A simulation run is typically controlled by a script file that is interpreted by *UG*'s user interface.

2.2 UG Library Subsystem Design

Each of the building blocks of the architectural design is decomposed into several subsystems. We now give an informal specification of the services provided by each subsystem.

User Interface
: The user interface provides the user with a "shell–like" command language. All operations of the *UG* library can usually be executed either via a command typed into the shell or by calling a C function within the code. A scripting language is available to control complex simulation runs. Multiple graphics windows can be opened to visualize simulation results. In two space dimensions the mesh can be manipulated interactively. The user interface is based on the portable device interface described below.

Graphics
: The graphics subsystem provides some elementary visualization methods like mesh plots, contour plots, color plots or vector fields. In three dimensions planar cuts and hidden line removal have been implemented. The advantages of an integrated graphics package are that no intermediate data files have to be written and also that information like matrix structure and entries can be displayed easily.

Numerics
: The numerics subsystem provides numerical algorithms in a modular form ranging from basic linear algebra (level 1 and 2) up to methods for the solution of nonlinear time–dependent partial differential equations. In addition, it provides support for the discretization process, *e.g.*, quadrature rules.

Domain Manager
: The purpose of the domain manager is to provide functionality for the description of general two– and three–dimensional domains as well as functions on the surface (boundary conditions) and the interior (coefficients) of a domain. The general approach is that a d–dimensional domain Ω is described by its boundary $\partial\Omega$ which is a $d-1$–dimensional hypersurface. This is very natural in the context of partial differential equations since boundary conditions have to be provided anyway. The standard way of describing the boundary is through local maps $f_i : \mathbb{R}^{d-1} \to \mathbb{R}^d$ with $i = 1 \ldots n$ and n the number of patches. Another approach consists of decomposing the boundary in a number of patches where each patch is given by a simplicial surface mesh. In that case, no easy mapping f_i exists for a patch. The domain interface is also used to access CAD data.

Grid Manager
: The grid manager subsystem provides the unstructured mesh and sparse matrix data structures together with functionality for their manipulation.

This includes the generation of two– and three–dimensional simplicial triangulations. The complete grid manager functionality has been parallelized based on the *DDD* library described below. Complicated multigrid structures can reside in a distributed manner in the memories of a parallel supercomputer platform and can be redistributed efficiently.

DEVICE MANAGER
: The device manager provides a default device called "screen" that allows at least basic character input/output. Optionally, the screen device also has interactive graphics capabilities. The screen device has been implemented for the standard C library, X11, remote X11 (uses socket communication and an X11 capable daemon on a remote machine), and Apple Macintosh. Write-only graphical output is available in postscript and a portable binary format ("metafile").

LOW
: This subsystem provides some basic functionality like memory management, a simple database tool and portable file input/output. Furthermore, some debugging tools are included.

LOAD BALANCER
: The load balancer subsystem is intended to solve graph partitioning and scheduling problems that arise when topological and numerical data must be mapped to processors in a parallel environment. The current implementation uses CHACO [9, 11] for that purpose.

DYNAMIC DISTRIBUTED DATA (DDD)
: The *DDD* subsystem implements an innovative parallel programming model that is especially suited for managing distributed, graph–like data structures. Distributed data objects can be created, deleted and transferred between processes easily. Communication among distributed objects is supported in a flexible and efficient way.

PARALLEL PROCESSOR INTERFACE (PPIF)
: PPIF is a portable message passing interface used by *DDD*. It has been implemented for PVM, MPI, PARIX, NX and the T3D/T3E. PPIF has very little overhead when used with fast native communication (*e.g.*, shared memory get/put on the Cray T3E).

2.3 Hierarchical Structure and Component Design

The various subsystems are ordered in a hierarchical structure, *e.g.*, the Grid Manager subsystem relies on the *DDD* subsystem, which again relies on the Parallel Processing Interface. Due to this hierarchy of implementation levels, the resulting program code with about 360.000 lines is still suited for maintenance; further development is possible, sequential and parallel version use the identical program code.

Each subsystem itself consists of a variety of components. A detailed description of this component design level is skipped here due to space limitations (refer to [4] for a detailed explanation).

3 Exemplary Applications

This section describes some exemplary *UG* applications, each accompagnied by numerical and visualized results computed on HLRS platforms.

3.1 Finite Element Applications

For a wide range of applications it is necessary to choose an appropriate discretization for the given problem in order to represent the inherent characteristics of the continuous problem in its approximation. Therefore, in *UG* a general finite element library is realized. It supports various different discretizations, such as linear and quadratic conforming elements, Crouzieux-Raviart elements, Raviart-Thomas elements, BDM-elements, Taylor-Hood elements and Morley elements. For this purpose, degrees of freedom can be associated to nodes, edges, faces and elements. Furthermore, in different parts of the computational domain different models can be chosen, and for nonmatching grids they can be coupled by the introduction of additional Lagrange multipliers via the Mortar finite element method.

The list of provided finite elements can be easily extended: essentially, the assembling of the local stiffness matrices and a local interpolation for every element must be implemented for every discretization, then all other tools of the *UG*-library can be applied. The parallel linear algebra model analysed in [15] guarantees that the parallel solvers provided by *UG* can be applied to all finite elements.

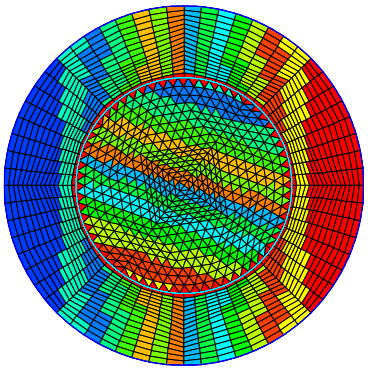
Table 1. Parallel multigrid performance for Morley elements.

level	6	7	8	9
number of unknowns	66049	263169	1050625	4198401
number of inverse iterations	3	3	4	4
time for linear solver (sec.)	0.99	3.65	14.6	62.3
first eigenvalue	1068.18	1071.48	1072.35	1072.59

The parallel performance on the Cray T3E at HLRS/Stuttgart is demonstrated by three examples. For the first example, we consider the eigenvalue computation of a clamped plate. In our test case, we consider the unit square $\Omega = (0,1)^2$ and the Poisson ratio $\nu = 1/3$. The problem is discretized with Morley elements, where pointwise values of u are associated to the element corners, and the normal derivative of u across the edges is associated to the edge midpoints; locally this defines a piecewise quadratic function. The total computation time for the first eigenvalue on the finest level was 7:28 min. on 128 processors, including the setup phase and the uniform mesh refinement, see Table 1 for more details.

In the next example, the application to a rotating geometry is demonstrated, where the coupling at the interface is realized by Mortar elements. For the resulting saddle point problem, the multigrid method discussed in [7] requires a local solution at the interface. Therefore, we use a load balancing scheme where all elements at the interface are collected on one processor, *cf.* Table 2.

Table 2. Parallel multigrid performance for Mortar finite elements on 128 processors (right) and load balancing on 32 processors (left).



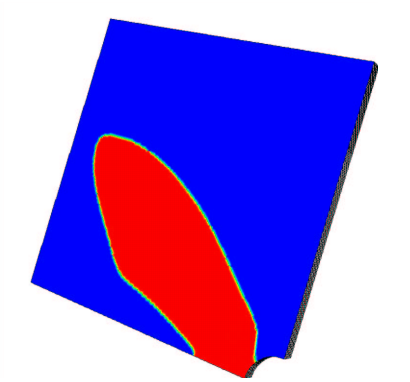
level	elements	conv. rate	time per V-cycle
6	32768	0.22	5.0 sec.
7	131072	0.18	10.3 sec.
8	524288	0.18	21.1 sec.
9	2097152	0.18	57.2 sec.

Finally, we give an example for a simulation of Prandtl-Reuß-plasticity without hardening, realized via a return mapping function. Here, additional data in the elements must be stored for the material history, and the evaluation of the material behaviour is a time consuming task within the total algorithm. Due to the nonsmooth character of this physical model, in particular in 3D a fine resolution of the plastic zone is required for a correct prediction of the resulting deformation after a complete loading cycle, *cf.* Table 3.

3.2 Incompressible Flow and Navier-Stokes Equations

Computational Fluid Dynamics has long been recognized as one of the most prominent examples for grand challenge problems. A problem class for the simulation of incompressible flow modelled by the Navier-Stokes equations has been implemented with *UG* and is a core part of the distribution. The discretization scheme uses 'colocated' variables based on dual finite volumes which are implemented in a fully coupled manner. The discretization is consistently stabilized by either a second or fourth order pressure term, both entering the equation of continuity. An upwind discretization is realized by a skewed upwind approach with an additional physical advection correction. In

Table 3. Parallel performance for Prandtl-Reuß-plasticity in 2D and 3D.



space dimension	2D	3D
elements	262144	202752
nb. of procs	256	96
unknowns	526338	692955
loading cycles	30	18
total time	15:35 min.	21:21 min.
Newton steps	161	55
nonlin. red.	0.000001	0.001
multigrid cycle	1757	223

contrast to many other accurate upwind schemes, this method does not need information from neighbour elements during the element assembly procedure, which is advantageous for unstructured grids and parallelization.

Table 4. Laminar flow around a cylinder: parallel execution times.

processors	elements	t_{lin}	t/it_{lin}	t_{total}
4	6912	60.47	1.63	277
32	52296	102.07	1.76	386
256	442368	158.83	2.06	560

The nonlinear stationary system is solved by a fixed point iteration and the linearized systems are solved either by a geometrical multigrid method or by Krylov subspace methods like BiCGSTAB or GMRES with the multigrid method as a preconditioner. Different smoothers can be employed for the multigrid method, like the incomplete LU factorization with β modification in the example shown below.

Figure 1 shows pressure iso-surfaces and the velocity for the calculation of a laminar flow around a cylinder at a Reynolds number of 20. Parallel performance results can be found in Table 4. A BiCGSTAB solver with a linear multi-grid preconditioner and an ILU$_\beta$ smoother was employed for the solution of the linear systems within the quasi-Newton nonlinear solver, which itself was used within a nested iteration scheme. t_{lin} denotes the total time spent for solving the linear problem, t/it_{lin} is the time for one multigrid cycle and t_{total} the total solution time on all levels. Almost 1.8 million degrees of freedom were used for the computation on 256 processors. As expected, the multigrid solver scales very nicely and the total solution time only increased by a factor of 2 for a problem with 64 times the size of the initial problem.

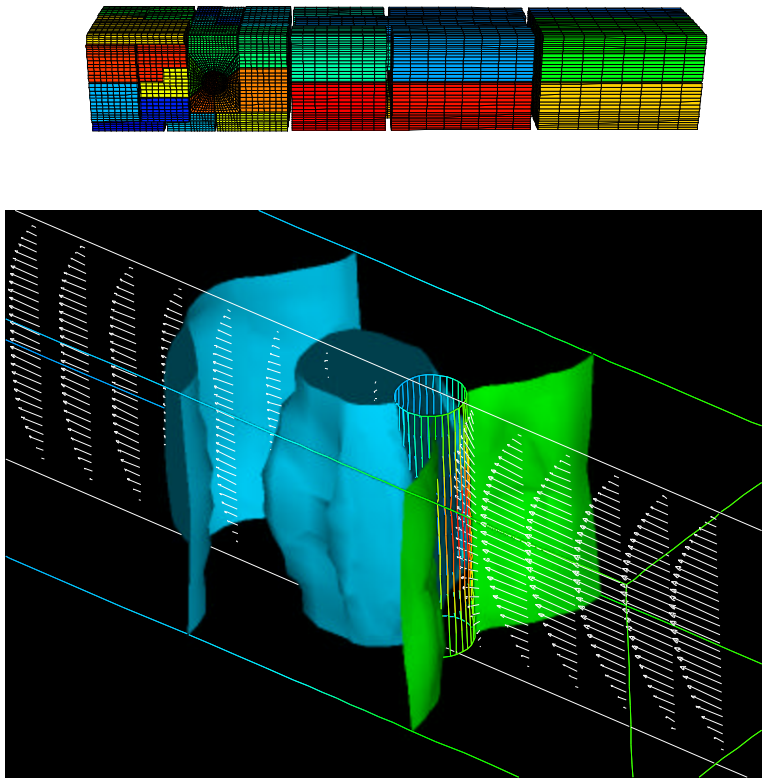


Fig. 1. Flow around a cylinder: Visualization of the result (bottom) and load balancing (top) for 32 processors.

3.3 Two–phase Flow in Porous Media

The flow of two immiscible fluids in a porous medium is described by two coupled highly non–linear time–dependent partial differential equations, see [1] for an introduction. These equations play an important role in oil reservoir simulation, the development of effective in–situ remediation techniques and the security assessment of underground waste repositories. Due to the hyperbolic/parabolic character of the equations, strong heterogeneities and high non–linearity, they pose a challenging problem for multigrid solution.

A problem class has been developed that solves the two–phase flow equations in a fully implicit / fully–coupled manner using either phase pressure–saturation or a global pressure–saturation formulation [5]. A finite volume and a control–volume–finite–element discretization with first–order upwinding have been implemented. Entry pressure effects at porous medium discon-

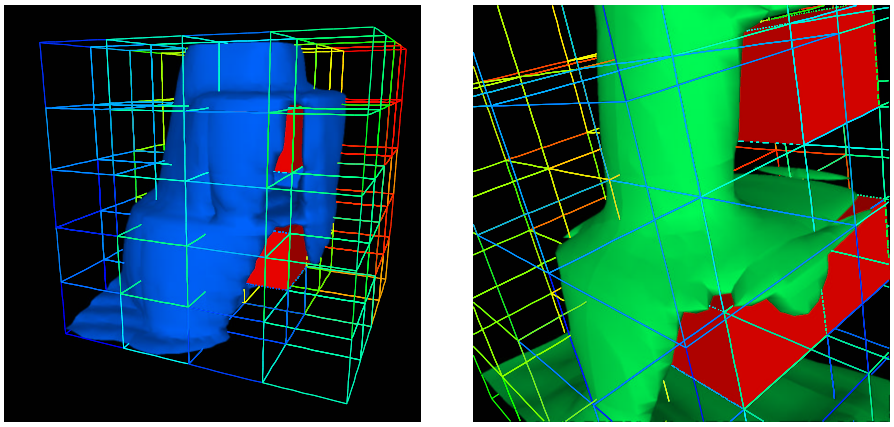


Fig. 2. 3D DNAPL infiltration into a heterogeneous porous medium. 10% saturation isosurface after 800 s shown left and 38% saturation after 1000 s right.

tinuities are handled by incorporating appropriate interface conditions. Both discretizations support all element types in two and three dimensions. Time discretization is fully implicit, resulting in a large set of nonlinear algebraic equations per time step. The nonlinear equations are then solved iteratively by a Newton–Multigrid technique. A line search method is used to achieve global convergence. Several multigrid techniques have been implemented in *UG* to handle coefficient jumps induced by saturation fronts and absolute permeability variations. These jumps are in general *not* aligned with coarse grid lines. In the simulations below, a multigrid method with truncated restriction, point–block ILU smoother and a V-cycle has been used.

Figure 2 shows saturation iso–surfaces for a three–dimensional DNAPL (dense non–aqueous phase liquid, a fluid with density higher than water and immiscible with it) infiltration into a heterogeneous porous medium. Two blocks of low permeability have been inserted into the reservoir. Entry pressure effects prevent the DNAPL from infiltrating the low permeability lenses. The performance data for this problem is shown in Table 5. In the table, MESH denotes the number of hexahedral elements (number of unknowns is twice this number), EXECT the total execution time in seconds, NLIT the number of Newton iterations for all time steps on the finest mesh, AVG the average number of multigrid cycles (within BiCGSTAB) per Newton iteration and Tit is the time for *one* multigrid cycle. Nested iteration is used to obtain good initial guesses on the fine mesh. The example shows that good parallel *and* overall efficiencies are obtained for large scale problems.

Table 5. Parallel performance of the T3E for a 3D computation on a hexahedral mesh with increasing number of elements and processors (scaled computation). 50 time steps of 20 [s] have been computed. A V–cycle multigrid algorithm with ILU smoother with two pre and two post–smoothing step is used as a preconditioner in BiCGSTAB.

T3E	MESH	EXECT	NLIT	AVG	Tit
1	5120	4187	218	1.6	2.10
4	40960	11589	243	2.5	4.69
32	327680	13214	264	3.5	4.76
256	2621440	14719	255	4.3	4.82

3.4 Density–driven Flow in Porous Media

In many cases, groundwater flow in porous media involves the transport of solutes that affect liquid density. If density variations exceed 20%, which occurs near salt domes or bedded salt formations, flow and transport are strongly coupled. The primary coupling arises in the equations through the body–force term of the fluid equation and the advection term of the transport equation. A second coupling arises from the velocity–dependent hydrodynamic dispersion in the transport equation. These couplings cause nonlinearities in the equations that preclude analytical solutions and are a challenge for numerical simulations.

Density–driven flow problems can be described by two nonlinear, coupled, time–dependent differential equations, a continuity equation for the fluid and a continuity equation for the solute transport. The fluid continuity equation is written in terms of pressure, assuming that Darcy's law is valid. Both equations are discretized on vertex-centered finite volumes using different constructions for the control volumes. In cases of dominant convection, an aligned finite volume method, where the finite volumes are aligned to a given velocity, is preferable to the standard finite volume method. Furthermore, a consistent velocity approximation of terms involved in the fluid velocity calculation is implemented. The transient equations are solved with a fully implicit time-stepping scheme with time step control. The nonlinear equations are solved in a fully coupled mode using an approximative Newton multigrid method where the linearized system is solved with a linear multigrid method.

Figure 3 shows the flow around a saltdome (on the bottom of the domain), which consists of four layers with different permeabilities. The top picture illustrates the grid and its distribution on 128 processors. The middle and bottom pictures show the velocity and salt distribution at time $t = 10a$, respectively. The parallel performance of the computations carried out on the Cray T3E is depicted in Table 6. The toplevel of the multigrid varies between grid level 2 and 4 and is scaled to the number of processors used. For performance reasons, the coarse grid is agglomerated on one dedicated processor to avoid communications during the solution of the small coarse

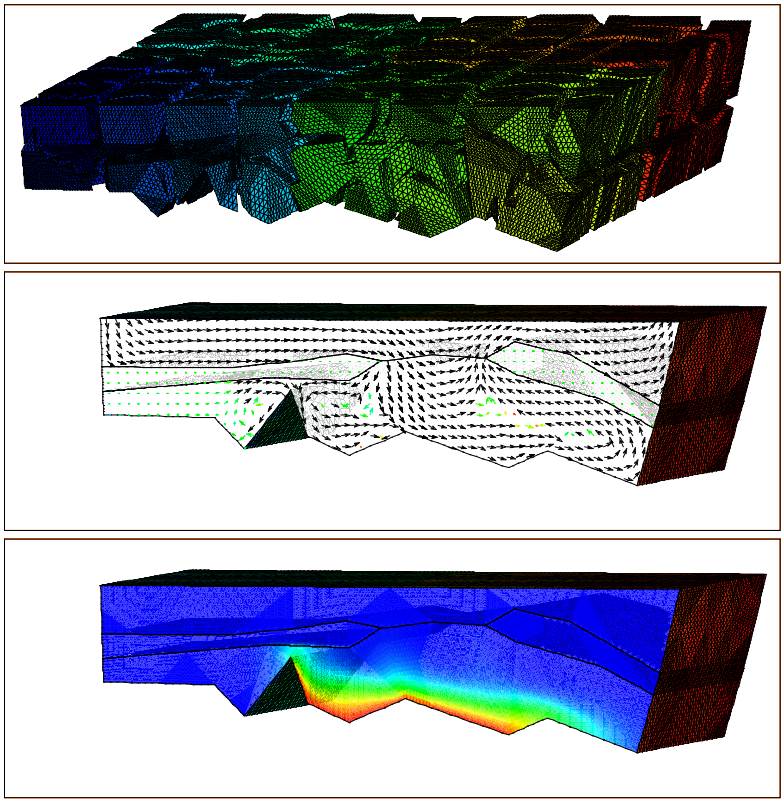


Fig. 3. Density–driven flow around a saltdome. From top to bottom: the mesh and its distribution on 128 processors, velocity field ($t = 10a$) and salt mass fraction ($t = 10a$).

grid problem. Therefore, the performance measure E_{par} shows the relative efficiency in comparison with the optimal balanced computation on 32 processors. Towards both a lower or a higher processor count efficieny degrades in the same amount. In the 4-processor case this is an effect of the coarse grid agglomeration, while the performance loss on 256 processors is mainly due to the degradation of the solver's convergence rate.

Figure 4 shows a parallel adaptive computation of the same problem after 8 time steps with parallel grid adaption (refinement and coarsening) and load balancing. The multigrid has 6 adaptive grid levels. The left upper picture illustrates the grid adaption. Colored (yellow, green and red elements) exists on the highest grid level, white elements reside on lower grid levels. The load balancer was a cheap and simple coordinate based method (Recursive

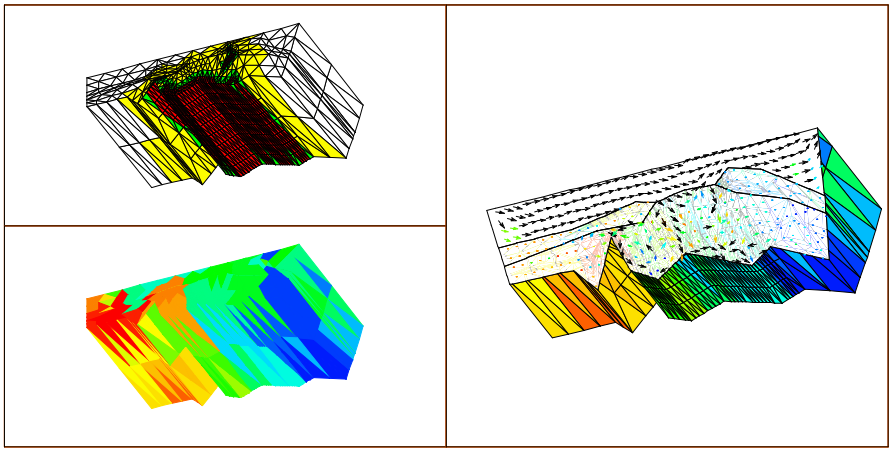


Fig. 4. Parallel adaptive computation on 32 T3E processors: adapted multigrid (left upper), load balancing (left lower) and velocity field of the solution (right picture).

Table 6. Density–driven flow around a saltdome: parallel performance on the Cray T3E at HLRS/Stuttgart.

processors	top-level	tetrahedra	nodes	time-steps	execution time[sec]	E_{par}
4	2	138944	26429	23	50534	0.77
32	3	1111552	198089	23	38728	1.00
256	4	8892416	1532881	23	50115	0.77

Inertial Bisection). The coloring of the left lower picture shows the multigrid partitions on the different processors. Finally, on the right picture the velocity field is shown.

4 Conclusions

This paper described the basic ideas and the software design structure of the *UG* package. Several exemplary applications have been shown together with numerical results on high-performance computing platforms at HLRS in Stuttgart. The resulting performance shows that it is practical and for many application problems even necessary to combine the computational power and main memory sizes of parallel computers with state-of-the-art numerical techniques for the solution of partial differential equations.

The layered, hierarchical approach of the *UG* software design leads to the following advantages:

– The resulting software is portable on a wide range of platforms.

- For new applications based on *UG* there is only minimal effort for parallelization.
- Using the *UG* approach, high-performance computing platforms can be exploited efficiently for a wide spectrum of different application areas.

References

1. K. Aziz and A. Settari. *Petroleum Reservoir Simulation.* Elsevier, 1979.
2. R. Bank, *PLTMG Users Guide Version 7.0*, SIAM, 1994.
3. P. Bastian, *Parallele adaptive Mehrgitterverfahren*, Teubner Skripten zur Numerik, Teubner-Verlag, 1996.
4. P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners. UG – a flexible software toolbox for solving partial differential equations. *Computation and Visualization in Science*, (1), 1997.
5. P. Bastian and R. Helmig. *Efficient Fully-Coupled Solution Techniques for Two-Phase Flow in Porous Media.* Advances in Water Resources Research, 1997 (submitted).
6. K. Birken. An efficient programming model for parallel and adaptive CFD-algorithms. In *Proceedings of Parallel CFD Conference 1994*, Kyoto, Japan, 1995. Elsevier Science.
7. D. Braess, W. Dahmen, and C. Wieners, *A multigrid algorithm for the mortar finite element method.* submitted.
8. P. Deuflhard, P. Leinen, and H. Yserentant, *Concepts of an adaptive hierarchical finite element code*, IMPACT of Computing in Science and Engineering, 1 (1989), pp. 3–35.
9. Hendrickson and R. Leland, *The chaco user's guide version 1.0*, Tech. Rep. SAND93-2339, Sandia National Laboratory, October 1993.
10. E. H. Hirschel, ed., *Flow Simulation with High-Performance Computers II*, Vieweg Verlag, Braunschweig, 1996.
11. S. Lang, *Lastverteilung für paralle adaptive Mehrgitterberechnungen*, Master's thesis, Universität Erlangen-Nürnberg, IMMD III, 1994.
12. D. J. Mavripilis, *Three–dimensional Multigrid Reynolds–averaged Navier–Stokes solver for unstructured meshes*, AIAA Journal, 33 (1995).
13. L. C. McInnes and B. Smith, *PetSc2.0: A case study of using MPI to develop numerical software libraries*, in Proc. of the MPI Developers Conference, Notre Dame, IN, 1995.
14. M. Raw, *A coupled algebraic multigrid solver for the 3D Navier–Stokes equations*, in Proc. of the 10^{th} GAMM Seminar Kiel, Notes on Numerical Fluid Mechanics, G. W. W. Hackbusch, ed., vol. 49, Vieweg–Verlag, 1995.
15. C. Wieners, *Parallel linear algebra and the application to multigrid methods*, in NNFM, vol. 56, W. Hackbusch and G. Wittum, eds., Viewig Verlag, 1999. in preparation.
16. G. Wittum, *Multigrid methods for Stokes- and Navier-Stokes equations*, Numer. Math., 54 (1989), pp. 543–563.