Advances in High-Performance Computing: Multigrid Methods for Partial Differential Equations and its Applications

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

IWR/Technical Simulation, University of Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

Abstract. The program package *UG* provides a software platform for discretizing and solving partial differential equations. It supports high level numerical methods for unstructured grids on massively parallel computers. Various applications of complex up to real-world problems have been realized, like Navier-Stokes problems with turbulence modeling, combustion problems, two-phase flow, density driven flow and multi-component transport in porous media. Here we report on new developments for a parallel algebraic multigrid solver and applications to an eigenvalue solver, to flow in porous media and to a simulation of Navier-Stokes equations with turbulence modeling.

1 Introduction

In many cases the modeling of physical and technical problems leads to a description by partial differential equations. Due to the complexity of the equations and the geometry involved, often the mathematical problem can be treated only by numerical methods. Appropriate discretizations lead to large systems of (non-linear) equations to be solved. To make the numerical treatment feasable, advanced numerical methods in conjunction with High Performance Computing have to be applied. Unstructured grids, adaptivity, multigrid methods, and parallelism have proven to be an efficient approach. Unstructured grids are required by the complex geometries. Adaptivity has to be used to reduce the computational cost especially for three-dimensional simulations. Finally the use of both multigrid methods and High Performance Computing on massively parallel MIMD machines is indispensable to reduce the computing time.

To bring together the features mentioned above the program package *UG* [4] has been designed during the last decade. It provides a platform for the discretization and the solution of partial differential equations on parallel computers. The kernel of *UG* has reached a mature state and state-of-the-art numerical methods can be applied to complex and real-world problems. Applications to the Navier-Stokes equations with turbulence modeling, to combustion problems, to two-phase flow, density driven flow and multi-component transport in porous media and to flow in fractured rock have been realized, see also [5, 6].

In this paper we report on some of the recent developments. In Section 2 developments of the parallel algebraic multigrid solver [6] are discussed. Section 3 is devoted to the parallel solving of eigenvalue problems. Section 4 reports on a porous media application with special emphasis on parallel performance. Section 5 presents a parallel application to density driven flow in porous media. Finally, in Section 6 an application to the Navier-Stokes equations with turbulence modeling is given.

2 Parallel Algebraic Multigrid FAMG

Besides classical (or geometric) multigrid methods purely algebraic multigrid methods (AMG) are desirable for many reasons. AMG considers only the stiffness matrix and no geometric information is needed. It has the potential to solve even very complicated problems. Furthermore AMG can be used in a geometric multigrid as the coarse grid solver. A third point of interest is the coupling of multigrid methods with already existing programs. The smallest possible interface—the matrix itself—is already feasible for AMG and the solver has no interaction with the geometry.

The starting point for our parallel AMG is the filtering AMG (FAMG) by Wagner [3,25]. The main idea is to choose for each node appropriate pairs of parent nodes for eventual elimination which ensures a certain filtering condition and leads to exactness on a given subspace. The best pairs are selected to eliminate the corresponding nodes; these parent nodes persist on the next coarser grid level and restriction and prolongation matrix entries with individually calculated values are installed between the nodes and their parents. The recursive application of this process yields a grid hierarchy on which standard multigrid algorithms can be realized.

For parallelizing this FAMG several additional steps have to be done. Since the elimination of a node influences its neighborhood, and thus the following selections for elimination, the cardinal point for the parallel FAMG will be to break up this sequential order in a suitable way. Due to the locality of the direct influence of a node, our approach will divide the nodes into two classes: those which are influenced directly by nodes on other processors and the rest. Whereas the latter can be eliminated locally on each processor, the processing of the first ones needs a synchronization. Therefore a parallel graph coloring method is used [13]. For further details see [6].

First results of the new parallel FAMG are obtained by examining the Laplace operator on the unit square with a tensor product mesh (called *structured case*) and on the shape of Lake St. Wolfgang (Austria) with an unstructured mesh from a grid generator (*unstructured case*). The structured mesh is distributed by a special load balancer to achieve tensor product like processor configurations, the unstructured mesh is distributed by Recursive Cordinate Bisection. For anisotropic model problems the equation $\epsilon u_{xx} + u_{yy} = f$ is solved with the anisotropy parameter ϵ.

Tab. 1 shows that the convergence rate is almost independent from the mesh width/the number of nodes. This is essential to solve large problems efficiently.

	structured							unstructured			
	isotropic			anisotropic 10^{-6}				isotropic			
unknowns	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]	unknowns	conv. rate	nr. it.	time [sec]
16641	0.053	7	6.6	10^{-5}	2	4.2	14095	0.057	7	4.6
66049	0.054	7	9.7	0.001	3	6.3	55637	0.123	9	9.4
263169	0.054	7	17.3	0.055	7	13.2	221065	0.206	12	20.7
1050625	0.057	7	38.1	0.044	6	31.8	881297	0.246	14	51.7
4198401	0.064	7	110.1	0.041	6	78.0	3519265	0.256	14	137.2
16785409	0.068	7	295.3	0.036	6	233.9	14065217	0.258	14	391.6

Table 1: The convergence rate depends only weakly upon mesh width (structured grid on a 16x8 processor configuration, unstructured grid on 128 processors).

Fig. 1 shows that the solver is very robust with respect to the variation of the anisotropy parameter. The convergence rate is bounded by 0.15 even for 128 processors and the solution time is nearly independent of this parameter; this holds for low load up to full load examples.

Next we evaluate the quality of the parallelization. First we consider the speedup where the same problem is solved on different numbers of processors. Tab. 2 indicates an almost constant convergence rate and the solution time decreases nearly proportional to the number of processors. Second we look at the scaleup in Tab. 3. In practice the scaleup is often more important than the speedup since large parallel computers are mainly used to solver larger

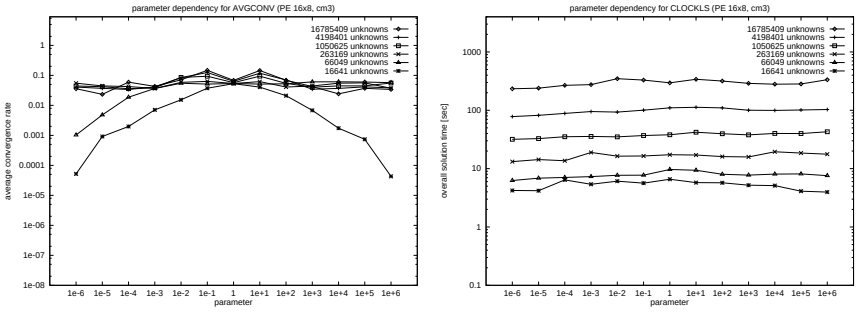

Fig. 1: Robustness against anisotropy (128 processors, different loads), *structured case.*

	structured						unstructured		
	isotropic			anisotropic 10^{-6}			isotropic		
PE	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]
8	0.058	7	220.6	10^{-4}	2	140.7	0.237	13	235.8
16	0.067	7	135.8	0.053	7	123.2	0.194	12	179.6
32	0.062	7	81.4	0.046	6	70.7	0.241	13	97.6
64	0.059	7	58.3	0.043	6	43.7	0.240	13	78.4
128	0.057	7	38.1	0.044	6	31.8	0.246	14	51.7

Table 2: Speedup on different processor configurations (structured grid with 1 million unknowns, unstructured grid with 880000 unknowns).

	structured						unstructured		
	isotropic			anisotropic 10^{-6}			isotropic		
PE	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]
2	0.054	7	162.0	10^{-6}	2	94.2	0.187	11	167.6
8	0.058	7	220.6	10^{-4}	2	140.7	0.237	13	235.8
32	0.067	7	263.2	0.041	6	229.9	0.257	14	311.1
128	0.068	7	295.3	0.036	6	233.9	0.258	14	391.6

Table 3: Parallel performance for scaled computations (constant load per PE: structured case 131000 unknowns, unstructured case 110000 unknowns).

problems instead of reducing the computational time. Again the convergence rate is (nearly) constant and the solution time increases only by a factor of about 2 from 2 up to 128 processors. Only in highly anisotropic cases it is not possible to hold the extremely good convergence rates for larger numbers of processors.

We have seen very promising features of the new parallel FAMG. Compared with the results presented in [6] we have realized substantial improvements by a factor of 2 to 3 in the performance. Nevertheless, further model problems (e. g. jumping coefficients and convection) should be examined to explore the benefits and the limitations of the new method. At the end real world problems should be solved.

The advantages of the IBM RS/6000 SP are its 2 GB nodes which enable very large serial calculations for several comparisons and the large memory on the many parallel nodes enables us to do various experiments with minor memory limitations. Already in the phase of development a large amount of computing time on many processors is required.

3 The efficient and reliable computation of eigenvalues and eigenmodes

In *UG*, a general finite element library is included [26] which supports various discretizations, and which is especially adapted to problems in nonlinear solid mechanics [27]. Here, the applicability and flexibility of the finite element module is illustrated on the following model problem in eigenvalue computations.

Let $\Omega \subset \mathbf{R}^d$, $d = 2, 3$ be a domain. We solve the eigenvalue problem:
Find $(w_i, \lambda_i) \in H^1(\Omega) \times \mathbf{R}$ *such that*

$$\int_\Omega \nabla w_i \cdot \nabla v \, dx = \lambda_i \int_\Omega w_i v \, dx, \qquad v \in H^1(\Omega).$$

This corresponds to the eigenvalue problem

$$-\Delta w_i = \lambda_i w_i \text{ in } \Omega, \qquad w_i \cdot n = 0 \text{ on } \partial\Omega$$

in the strong formulation; in particular, we impose Neumann boundary conditions. Thus, we know a priori the first trivial eigenpair $w_0 \equiv 1$ and $\lambda_0 = 0$. Note that this results in a singular stiffness matrix (which, has severe consequences for the solution process).

The eigenmode approximations are computed by a block inverse iteration with a full Ritz-Galerkin orthogonalization in every step. Starting from initial guesses $\tilde{w}_1, ..., \tilde{w}_m$, we perform the following algorithm:

a) Ritz-Galerkin step: We form the small matrices

$$M = \left(\int_\Omega \nabla \tilde{w}_i \cdot \nabla \tilde{w}_j \, dx \right), \qquad N = \left(\int_\Omega \tilde{w}_i \tilde{w}_j \, dx \right),$$

and we solve the generalized eigenvalue problem

$$M x_i = \lambda_i N x_i, \qquad x_i \in \mathbf{R}^m;$$

then, we define approximations by

$$w_i = \frac{1}{x_i^T N x_i} \sum_{j=1}^{m} x_{ij} \, \tilde{w}_j, \qquad i = 1, ..., m.$$

b) Block inverse iteration step: We update the space of test functions by solving

$$\int_\Omega \nabla \tilde{w}_i \cdot \nabla v \, dx = \lambda_i \int_\Omega w_i v \, dx, \qquad v \in H^1(\Omega), i = 1, ..., m$$

and return to a).

For the solution of the linear problems, we use a fixed number of parallel multiplicative multigrid cycles with Krylov acceleration and Gauß-Seidel

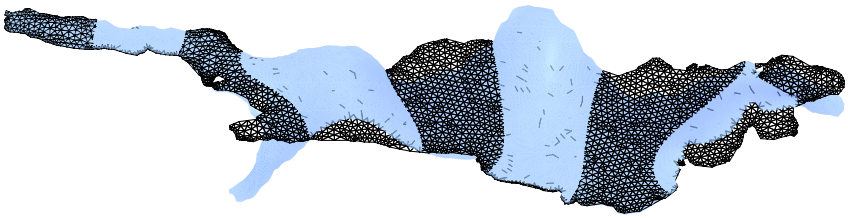

Fig. 2: The 11th eigenmode w_{11} of lake Constance, Germany.

smoother, where we project in every step the solution onto the space which is orthogonal to w_0.

As an example, we present the parallel results of the eigenmode computations on a domain representing the surface of lake Constance, Germany. The shape of the eigenmode w_{11} is visualized in Fig. 2 (using *GRAPE* [22]), the first 16 eigenvalues are listed in Tab. 5.

level	2	3	4	5
elements	101312	405248	1620992	6483968
unknowns	205745	816737	3254465	12992897
32 processors	21 min.	49 min.		
	[21 sec.]	[50 sec.]	[190 sec.]	
64 processors	11 min.	28 min.	93 min.	
	[13 sec.]	[26 sec.]	[99 sec.]	
128 processors	7 min.	15 min.	47 min.	
	[9 sec.]	[14 sec.]	[44 sec.]	[169.8 sec.]

Table 4: Parallel performance of the eigenvalue computation (average time for the linear solver in every step) on IBM RS/6000 for P_2-elements. Due to the cpu-time-limit, not sufficiently many steps of the block inverse iteration could be performed on level 4 (32 proc.) and level 5 (128 proc.).

The parallel performance (see Tab. 4) underlines the efficiency of the method, which is asymptotically of optimal complexity and optimal parallel scalability (due to the large coarse grid problem with 6332 elements we do not obtain optimal multigrid efficiency in the first refinement step). Since we use a second order discretization and millions of unknowns, the results are reliable due to the monotone asymptotic convergence; by simple extrapolation, we expect an accuracy of at least 0.1%.

Nevertheless, fully reliable results can be obtained only by the computation of rigorous eigenvalue inclusions; this is realized—using interval

discretization	P_1	P_2	P_2	P_2
elements	1620992	101312	405248	1620992
unknowns	816737	205745	816737	3254465
λ_0	0.00000e-00	0.00000e-00	0.00000e-00	0.00000e-00
λ_1	8.88113e-04	8.88274e-04	8.89473e-04	8.88088e-04
λ_2	2.46599e-03	2.46597e-03	2.46582e-03	2.46576e-03
λ_3	4.77479e-03	4.77490e-03	4.77461e-03	4.77447e-03
λ_4	8.60307e-03	8.60324e-03	8.60273e-03	8.60253e-03
λ_5	1.27349e-02	1.27350e-02	1.27344e-02	1.27342e-02
λ_6	1.69387e-02	1.67759e-02	1.67748e-02	1.67744e-02
λ_7	1.95241e-02	1.95241e-02	1.95234e-02	1.95230e-02
λ_8	2.49470e-02	2.49471e-02	2.49464e-02	2.49462e-02
λ_9	3.01706e-02	3.01705e-02	3.01696e-02	3.01692e-02
λ_{10}	3.23264e-02	3.23267e-02	3.23240e-02	3.23240e-02
λ_{11}	3.86922e-02	3.87021e-02	3.86901e-02	3.86892e-02
λ_{12}	4.09167e-02	4.09226e-02	4.09123e-02	4.09111e-02
λ_{13}	4.59070e-02	4.60913e-02	4.59268e-02	4.58889e-02
λ_{14}	4.89770e-02	4.98155e-02	4.91613e-02	4.90074e-02
λ_{15}	5.14639e-02	5.19207e-02	5.15270e-02	5.14699e-02
λ_{16}	6.09801e-02	6.90931e-02	6.80042e-02	6.70623e-02

Table 5: Comparison of the numerical solution with different discretizations and different mesh sizes of the 16 smallest eigenvalues of the Bodensee.

arithmetic—on IBM RS/6000, and first results (on simpler geometries) are already documented in [9].

4 Two-Phase Flow

This section evaluates the speedup, which can be achieved when a fixed size problem has to be computed fast for productivity reasons. Fast computation of a given problem is for example advantageous, if parameter studies are necessary for optimization.

Two phase flow problems with phases consisting of liquids with different viscosity form fronts which tend to be instable [1]. These instabilities can be observed in real live or physical experiments as fingers. The fingering results from a statistical behaviour, where interactions between the two phases on a very small scale influence the structure of the viscous front on a visible level [2].

As a test scenario we choose a two-phase flow problem, where water displaces oil in a cubic channel (see figure 3). Interesting are different fingering patterns evolving from different viscoscity parameters of the two phases. For details about the numerical schemes of discretization and solution see [7]. Since the resolution of the highly nonlinear effects at the front must be very

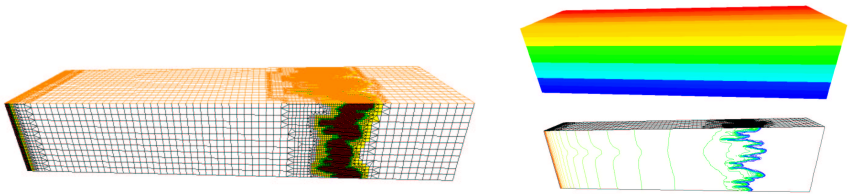

Fig. 3: Adaptively refined multigrid with hexa/tetrahedra and pyramids; 16×16 stripped load balancing in flow direction; Vertical cutted domain with isolines of concentration.

high, the combination of parallelism and adaptivity tends to be an optimal approach. As load balancer a simple Recursive Orthogonal Bisection Method is applied to distribute the coarse grid levels in flow direction. Thus important coupling information of the problem is kept local to the processors to avoid degradation of the solver's convergence rate. No load balancing needs to be done during computation, since the load remains equally distributed between the processors when the front moves.

PROCS	1	2	4	8	16	32	64	128	256
NLSOLVE [s]	233921	145842	72575	37962	17597	9091	4599	2423	1354
$S_{NLSOLVE}$	1	1.60	3.22	6.16	13.3	25.7	50.9	96.6	173
ADAPT [s]	3413	2426	1541	1238	691	502	347	314	271
S_{ADAPT}	1	1.41	2.21	2.76	4.94	6.80	9.84	10.9	12.6
TOTAL [s]	237394	148268	74116	39200	18288	9593	4946	2737	1625
S_{TOTAL}	1	1.6	3.20	6.06	13.0	24.7	48.0	86.7	146

Table 6: Fixed speedup for two-phase flow: NLSOLVE (nonlinear solution in seconds), $S_{NLSOLVE}$ (speedup for this phase), ADAPT (grid adaption in seconds), S_{ADAPT} (speedup for this phase), TOTAL (overall time in seconds), S_{TOTAL} (overall speedup).

Tab. 4 shows the speedup of the displacement example for the first 34 timesteps. The minimal amount of serial main memory needed is about 2 GB to allow the front to finger after about 100 timesteps. This corresponds to about 1 million unknowns which have to treated in each timestep. Results for one to eight processors are computed on a IBM SP-256, from 8 to 256 processors on a Cray T3E. Timings are scaled by a constant factor, which is gained from the comparison of the 8 processor run on both machines.

The nonlinear solution process scales good, giving a speedup of 173 for 256 processors. Speedup losses have two reasons: The first is the introduction of communication in the linear solver, when going from one to two processors; speedup loss is 20%. Second the convergence rates become worse with

increasing processor count. The cycle time of one linear iteration step scales good, which shows a stable scaling of the machine itself.

Grid adaption shows a scaling which is not comparable to that of the nonlinear solution. A speedup factor of 12.6 could be achieved in the 256 processor configuration. Analysis of the losses has shown that the grid manager itself has a better speedup behaviour, but the underlying programming model DDD has high constant algorithmic costs for several modules. This means that the algorithmic costs, which behave linearly with the number of refined or coarsened elements, are dominated by the fixed constant costs as processor count increases. Thus the fundamental approach of the grid manager is valid and has no principal weakness, but the efficiency should be improved by reimplementation of some lower DDD layers. This can be done without changing the grid adaption process itself. E.g., the rebuilding of the interfaces for communication is one bottleneck during adaption, which can be speeded up by reimplementation of the DDD interface module.

Fixed speedup of this experiment behaves well up to 64 processor, only in the 128 and 256 processor case grid adaption has a significant impact on the efficiency. The total efficiency of the 256 processor run is 0.57 (speedup 146), which can be reguarded as a quite satisfying overall result for a fixed sized problem with adaptive mesh refinement.

5 Density–Driven Flow in Porous Media

The density driven flow in porous media can be described by two coupled non-linear time-dependent partial differential equations, see [8, 16, 19]. In the case of strong coupling, i.e. if the transported solute affects the flow field, the fully coupled equations have to be solved. These situations arise frequently in practice and are of significant importance. To mention are the problem of coastal saltwater intrusion, the groundwater flow in the vicinity of salt domes, upconing of saltwater in a stable system of saltwater and freshwater layers.

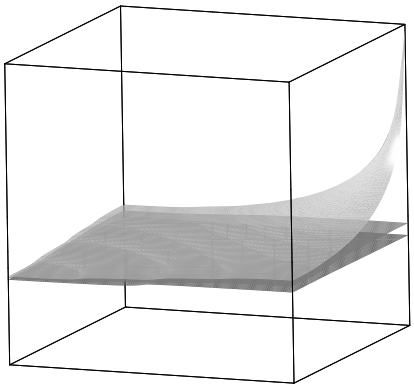

Fig. 4: Iso-surfaces for the saltpool, case 2 (graphics by GRAPE [22]).

We discretize and solve the equations using the program package d^3f [10], a simulator based on *UG*. The equations are discretized by means of mass conserving finite volumes using a consistent velocity approximation [11]. The semi-discrete equations are solved by a

diagonally implicit Runge-Kutta method [14,18]. The discretization is second order consistent both in space and time. The solution of the discrete system is done by a fully coupled/fully implicit solving strategy. After linearization the linear subproblems are solved by a multigrid V-cyle iteration using a modified Vanca-type SSOR smoother [17]. We report on numerical results for the benchmark problem *saltpool, case 2* defined in [18]. It is a mathematical model for an upconing experiment described in [20]. Starting with a stable system of a saltwater below a freshwater layer, freshwater charges at one of the upper corners of the cube and water discharges at the diagonally opposite upper corner. The upconing effect can easily be seen in Fig. 4, showing three iso-surfaces of the solute concentration ($c = 0.1,\ 0.5,\ 0.9$) at the end of the simulation ($t = 9495s$). To obtain an accurate numerical result a fine grid spacing had to be used (2097152 hexahedral elements). Due to memory requirements as well as due to computational time the simulation had to be carried out on a parallel computer (Cray T3E).

6 Navier-Stokes Equations and Turbulence Modelling

The Navier-Stokes library in *UG* uses a Control Volume Finite Element method with colocated variables. Since a colocated scheme is not stable, a special stabilization scheme is applied which introduces a physical advection correction scheme to couple the momentum equation and the pressure equation. This results in a Laplacian term for the pressure in the continuity equation scaled with the mesh size squared and therefore tends to zero as the grid is refined. This physical advection correction scheme called FIELDS was developed by Raw [23]. The idea is to solve in each element a Finite Difference approximation of the linearized momentum equation at all integration points. The resulting integration point velocities depend on all corner values of velocity and pressure. After insertion in the continuity equation a pressure dependence and full coupling of all equations is gained. This can be done independently on all elements and is therefore very advantageous for parallelization.

A special problem class in the Navier-Stokes community is turbulence modelling. A very promising way to simulate turbulent flow characteristics is the so-called Large Eddy Simulation (LES). In contrast to Reynolds averaged turbulence models (RANS), it is not based on time averages but on local volume averages. This means that large structures have to be resolved and only the small ones are modelled. RANS methods model all structures and therefore a suitable turbulence model is not easy to design. LES models use the fact that small structures are nearly isotropic and more universal than large structures and hence the modelling becomes simpler. Nevertheless a high spatial resolution is neccessary, requiring the usage of a High Performance Computer.

The averaged equations for LES are derived by applying filter operators (for example a volume-average box filter) to the governing equations. For the

momentum equation this results in:

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial}{\partial x_j}\left(\overline{u_i}\,\overline{u_j}\right) + \frac{\partial \overline{p}}{\partial x_i} - \frac{\partial}{\partial x_j}\left(\nu\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right)\right) + \frac{\partial}{\partial x_j}\tau_{ij} = 0,$$

where an overbar denotes an average value. The subgrid scale stress tensor τ_{ij} is modelled by the dynamic model of Germano [12]. This model, in contrast to the Smagorinsky model [24], for which a constant and universal model parameter is assumed, is truly local and hence it is able to reflect approximately local flow phenomena. Finally the model has the form:

$$\tau_{ij} - \frac{1}{3}\delta_{ij}\tau_{kk} = -2C\Delta^2|\overline{S}|\overline{S_{ij}}$$

with

$$\overline{S_{ij}} = \frac{1}{2}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right), \quad |\overline{S}| = \sqrt{2\overline{S_{ij}}\,\overline{S_{ij}}}.$$

An interesting problem is the flow around an infinitely long square cylinder as described in the workshop of Rodi et al. [21]. The infinite dimension in axis direction can be numerically realized using periodic boundary conditions since the flow structures in that direction can be assumed to be almost periodic. A sketch of the domain is given in Fig. 5. For implementing periodic boundary conditions in *UG* each element at the periodic boundary and its partner at the opposite side have to be assigned to the same processor. For this reason the load balancing strategy RCB of CHACO [15] has been modified to handle periodicity. An example for 8 processors can be seen in Fig. 5.

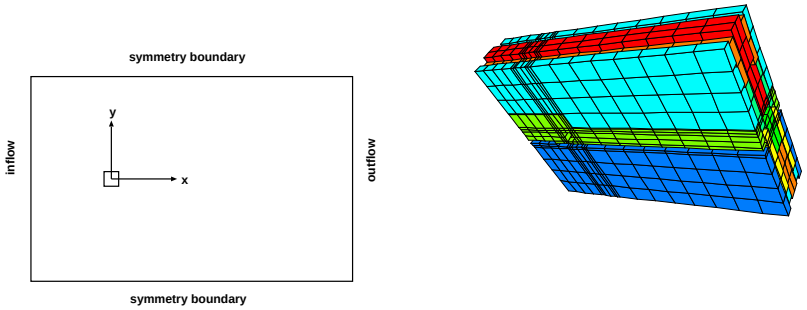

Fig. 5: Sketch of the domain and load balance on 8 processors.

The Reynolds number for this problem is 1000. The grid is built of 800000 hexahedrons which correspond to 3 million unknowns. The nonlinear system

is linearized by a Quasi-Newton method and the resulting linear system is solved by the Kryloc subspace method BiCGSTAB with multigrid as preconditioner. As smoother the incomplete LU factorization with β modification is employed and the time solver is a diagonally implicit Runge-Kutta-method of second order. In Fig. 6 the solution of the velocity component in x-direction on a line through the midpoint of the cylinder in mean flow direction is shown at 6 successive points of time illustrating the complex behaviour of the flow.

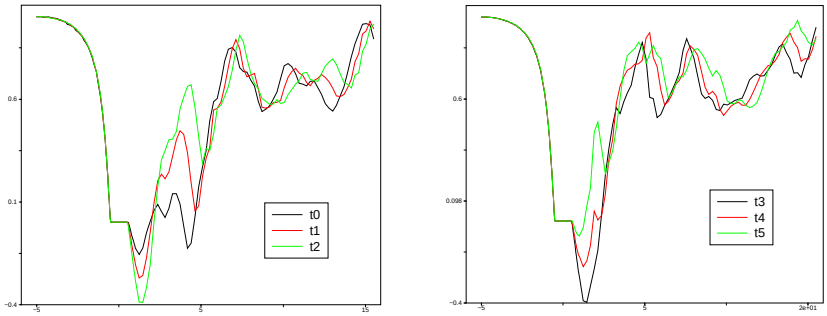

Fig. 6: u-component of the velocity vector.

7 Conclusions

With the program package *UG* we follow the strategy of combining advanced numerical methods with the advantages of high-performance computers like the IBM RS/6000 SP and the Cray T3E. It has been shown that on the base of the portable code for massively parallel computers both the development of new, highly efficient numerical algorithms and the solution of complex problems is feasible.

On the one hand we have reported on the development of algorithmic aspects, i.e. further advances in parallel algebraic multigrid methods. It has been shown that an efficient multigrid algorithm on a purely algebraic base for elliptic and singular perturbed problems can be implemented on (massively) parallel computers without loss of its performance properties (section 2). On the other hand several applications have been presented. They range from the calculation of eigen-values on unstructured grids to turbulence modelling. Due to memory requirements and/or due to the expense of computational time the applications presented here took great benfits from the usage of High-Performance Computers.

References

1. K. Aziz and A. Settari. *Petroleum Reservoir Simulation.* Elsevier, 1979.
2. E. O. Frind B. H. Kueper. An overview of immiscible fingering in porous media. *J. of Contaminant Hydrology*, 2:95–110, 1988.
3. R. E. Bank and C. Wagner. Multilevel ILU decomposition. *Numerische Mathematik*, 82:543–576, 1999.
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. *Computing and Visualization in Science*, 1:27–40, 1997.
5. P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, C. Wieners, G. Wittum, and C. Wrobel. A parallel software-platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods. In E. Krause and W. Jäger, editors, *High performance computing in science and engineering*, pages 326–339. Springer, 1999.
6. P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, C. Wieners, G. Wittum, and C. Wrobel. Parallel solutions of partial differential equations with adaptive multigrid methods on unstructured grids. In *High performance computing in science and engineering II*, 1999. submitted.
7. P. Bastian and R. Helmig. Efficient fully-coupled solution techniques for two-phase flow in porous media. *Advances in Water Resources Research*, 23:199–216, 1999.
8. J. Bear and Y. Bachmat. *Introduction to Modeling of Transport Phenomena in Porous Media.* Kluwer Academic Publishers, 1991.
9. H. Behnke, U. Mertins, M. Plum, and C. Wieners. Eigenvalue inclusions via domain decomposition. 1999. submitted.
10. E. Fein(ed). d3f - Ein Programmpaket zur Modellierung von Dichteströmungen. *GRS, Braunschweig*, GRS - 139, ISBN 3 - 923875 - 97 - 5, 1998.
11. P. Frolkovic. Consistent velocity approximation for density driven flow and transport. Technical report, Advanced Computational Methods in Engineering, Ghent 1998, to be published.
12. M. Germano. Turbulence: the filtering approach. *Journal of Fluid Mechanics*, 238:325–336, 1992.
13. R. K. Gjertsen Jr., M. T. Jones, and P. E. Plassmann. Parallel heuristics for improved, balanced graph colorings. *Journal of Parallel and Distributed Computing*, to appear.
14. E. Hairer and G. Wanner. *Solving ordinary differential equations II.* Springer-Verlag, Berlin, 1991.
15. B. Hendrickson and R. Leland. The chaco user's guide version 1.0. Technical Report SAND 93-2339, Sandia National Laboratory, 1993.
16. Ekkehard O. Holzbecher. *Modeling Density-Driven Flow in Porous Media.* Springer-Verlag, Berlin, Heidelberg, 1998.
17. K. Johannsen. Modified SSOR - a smoother for non M-matrices. *in preparation.*
18. K. Johannsen, W. Kinzelbach, S. Oswald, and G. Wittum. Numerical simulation of density driven flow in porous media. *in preparation.*
19. A. Leijnse. *Three-dimensional modeling of coupled flow and transport in porous media.* PhD thesis, University of Notre Dame, Indiana, 1992.
20. S. E. Oswald, M. Scheidegger, and W. Kinzelbach. A three-dimensional physical benchmark test for verification of variable-density driven flow in time. *Water resources research*, to be published.

21. W. Rodi, J. H. Ferziger, M. Breuer, and M. Pourquié. Status of large eddy simulation: Results of a workshop. *Journal of Fluids Engineering*, 119:248–262, 1997.
22. M. Rumpf and A. Wierse. Grape, eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forschung und Entwicklung*, 7:145–151, 1992.
23. G. E. Schneider and M. J. Raw. Control volume finite-element method for heat transfer and fluid flow using colocated variables. *Numerical Heat Transfer*, 11:363–390, 1987.
24. J. Smagorinsky. General circulation experiments with the primitive equations i. the basic experiment. *Monthly Weather Review*, 91:99–164, 1963.
25. Christian Wagner. On the algebraic construction of multilevel transfer operators. *Computing*, 2000, to appear.
26. C. Wieners. The implementation of adaptive multigrid methods for finite elements. Technical report, Universität Stuttgart, SFB 404 Preprint 97/12, 1997.
27. C. Wieners. Multigrid methods for Prandtl-Reuß-plasticity. *Numer. Lin. Alg. Appl.*, 6:457–478, 1999.