PARALLEL FINITE ELEMENT METHODS FOR THE INCOMPRESSIBLE NAVIER–STOKES EQUATIONS

O. Dorok, V. John, U. Risch, F. Schieweck, L. Tobiska
Institut für Analysis und Numerik
Otto-von-Guericke Universität Magdeburg
Postfach 4120, D-39016 Magdeburg

SUMMARY

We consider parallel and adaptive algorithms for the incompressible Navier-Stokes equations discretized by an upwind type finite element method. Two parallelization concepts are used, a first one based on a static domain decomposition into macroelements and a second one based on a dynamic load balancing strategy. We investigate questions of the scalability up to the massive parallel case and the use of a posteriori error estimators. The arising discrete systems are solved by parallelized multigrid methods which are applied either directly to the coupled system or within a projection method.

1. INTRODUCTION

The reliable numerical prediction of complex flows requires the use of the most efficient numerical methods together with the most powerful computer architectures. Concerning the hardware, only parallel computers can offer the necessary computational performance and memory capacity for future needs to provide solutions with acceptable accuracy. During the last years multigrid methods have become a powerful technique for solving the corresponding algebraic equations and adaptive finite elements have been used to get a better resolution of boundary and interiour layer regions.

In this paper we consider efficient finite element methods for solving the incompressible Navier–Stokes equations in a two-dimensional domain Ω

$$-\frac{1}{\mathrm{Re}}\Delta u + (u\cdot\nabla)u + \nabla p = f, \quad \nabla\cdot u = 0 \quad \text{in } \Omega, \tag{1}$$

with given velocity at the Lipschitz continuous boundary $\partial\Omega$ in the case of higher Reynolds numbers and their parallel implementation on MIMD-machines. For other publications from this project see [3],[13].

We use two different approaches of parallelization. A first one, presented in Section 3, is based on a fixed decomposition of the domain into macroelements which are uniformly refined and assigned to different processors. The second one, desribed in Section 4, uses a posteriori error estimates for a local mesh refinement and requires dynamic load balancing strategies. The implementation of this full adaptive algorithm on a parallel machine has been realized using modules of the package *ugp1.0* [1] which is also used in other projects of the DFG Priority Research Program "Flow Simulation with High-Performance Computers" [7],[8].

2. DISCRETIZATION METHOD

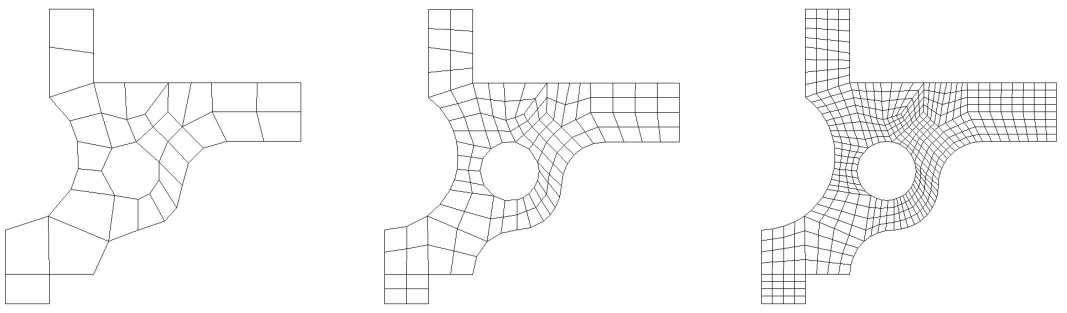
For the discretization of the weak formulation of the Navier–Stokes equations (1) we use two non–conforming elements, namely the P_1/P_0 triangular element of Crouzeix/Raviart [2] and the rotated bilinear quadrilateral Q_1/Q_0 element of Rannacher/Turek [9]. Both element pairs guarantee the discrete Babuška–Brezzi stability condition. Furthermore, in both cases the unknowns are the velocity values at the midpoints of the element edges and the pressure values at the element centres. This fact is advantageous for parallelization since a velocity node belongs to only two elements and a pressure node to only one element which leads to a very cheap local communication. In order to handle the effect of dominated convection we use an upwind stabilization of the convective term which has been developed in [12] for a triangular mesh. This technique contains both the simple first order and the Samarskij upwinding.

3. A PARALLELIZATION STRATEGY BASED ON MACRO ELEMENTS

Here, we use our finite element approximation of upwind type [12] on a quadrilateral mesh with nonconforming rotated bilinear elements for the velocity and piecewise constant elements for the pressure [9]. The algorithmical basis for our parallel Navier–Stokes solver is a robust and efficient sequential multigrid method [10], [11]. We use the domain decomposition into macroelements $\{M_i\}$ and distribute the whole work and data of the sequential algorithm to different tasks $\{T_i\}$ on a processor network where the task T_i is associated with the macroelement M_i. To avoid recursive data dependencies a modification has to be made in the smoothing procedure. However, this has just a slight influence on the convergence rate of the parallel algorithm compared to the sequential one (see [11]). Thus, we have a parallel solver with a good numerical efficiency up to the massive parallel case. We discuss the influence of the communication overhead and the coarsest grid solver on the parallel efficiency for an increasing number of macroelements and processors.

Sequential Multigrid Solver

For the definition of a multilevel grid sequence we start with a decomposition of the domain Ω into quadrilateral elements M_i which we call *macroelements* and which define the finite element mesh on the coarsest grid level ℓ=1. Then, all finer grid levels are defined by uniform refinement with the usual modification at curved boundary parts.



grid level 1 grid level 2 grid level 3

Figure 1: Example of a multilevel grid sequence

This can be done in parallel on each macroelement. Figure 1 shows an example of such a grid generation. For this kind of blockstructured grids we have on the one hand a relatively high flexibility for complex geometries and on the other hand the local regularity of the grid on the macroelements. This opens the possibility for vectorization on each macroelement and it guarantees the regularity assumption for the global mesh which is necessary for stability and convergence of our element pair [9]. Furthermore, we get a natural and nearly optimal load balancing for the parallelization if we assign one processor to each macroelement.

Now we want to solve the discrete Navier–Stokes problem on some finest grid level $\ell{=}k$, i.e. we want to find on this grid a solution vector $y{=}(u,p)^{\mathsf{T}}$, consisting of the velocity vector u and the pressure vector p, which satisfies the following nonlinear system of equations

$$L(y)\,y = F \tag{2}$$

with

$$L(y) := \begin{pmatrix} A(u) & B \\ B^{\mathrm{T}} & 0 \end{pmatrix}, \quad y := \begin{pmatrix} u \\ p \end{pmatrix}, \quad F := \begin{pmatrix} f \\ g \end{pmatrix} \tag{3}$$

where the matrix $A(u)$ contains the upwind-discretization of the nonlinear convection term. To solve this nonlinear problem we use a fixed-point iteration where in the n-th step we calculate $\bar{y}^n$ as an approximate solution of the linear system

$$L(y^{n-1})\,\bar{y}^n = F \tag{4}$$

and take the new iterate

$$y^n = y^{n-1} + \omega(\bar{y}^n - y^{n-1})$$

with some damping parameter ω (in numerical calculations we found the value $\omega{=}0.9$ to be suitable).

For solving the linear problem (4) we apply a certain number of multigrid cycles. This multigrid iteration is stopped if the initial defect is damped by a factor of 0.1 or after a maximum of 5 iterations. We will only briefly mention the components of the multigrid algorithm (for details see [10]). We take an F–cycle where the coarse grid correction is damped by a factor of 0.8. For the grid transfer we use standard finite element prolongation and restriction with a natural modification at points where the nonconforming finite element functions are discontinuous. The coarsest grid problems are solved by an augmented Lagrange algorithm. We use a blockwise Gauss–Seidel smoother similar to that proposed by Vanka [15].

In the following we want to explain the smoother in a little more detail. In order to perform one smoothing iteration on some actual grid level we pass through all elements K of that grid and update for each K the corresponding local block vector $y_K=(u_K, p_K)^{\mathsf{T}}$ which consists of all unknowns belonging to that element K (these are the 8 velocity values at the 4 midpoints of the edges of K and the one constant pressure value p_K on element K). We update y_K by solving a modified local system (see [10])

$$\tilde{L}_K y_K = \tilde{F}_K \qquad (5)$$

which is derived from the global system (4) on the current grid level by considering the equations that are associated with element K. Let us note that for this element loop we use a Gauss-Seidel-like update mechanism, i.e. within one smoothing step we can have a spreading of new updated information over the whole domain. At the end of each smoothing iteration we apply an underrelaxation step.

Parallelization

At first we construct a task network $\{T_i\}$ which is topologically equivalent to the macroelement decomposition $\{M_i\}$ of our domain (see Figure 2). That means we have a mapping

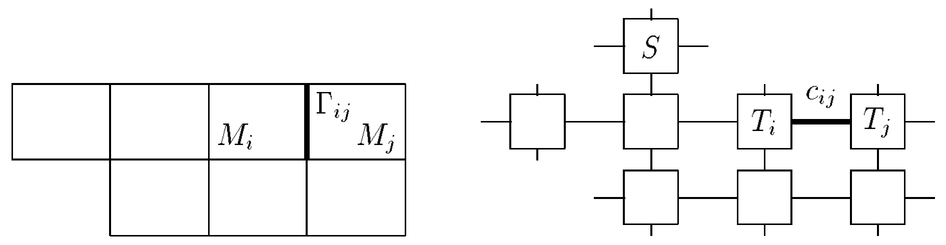


Figure 2: Macroelement decomposition $\{M_i\}$ and the related task network $\{T_i\}$ with an extra solver task S

$M_i \to T_i$ where any two neighbouring macroelements M_i and M_j having a common edge Γ_{ij} are assigned to neighbouring tasks T_i and T_j connected by a communication channel c_{ij}. Then, we map the set of all tasks $\{T_i\}$ to a given set of processors $\{P_j\}$. If we have enough processors we will assign one task to one processor otherwise we have a load balancing problem to get a suitable mapping. Now our parallelization is characterized as follows. For the algorithm we use the sequential one with a slight modification in the smoother (see below). Each task T_i is responsible for all operations of the algorithm with data belonging to macroelement M_i. For operations with ∂M_i–data on the boundary of M_i the task T_i exchanges its Γ_{ij}–data with all neighbouring tasks T_j via the corresponding channels c_{ij}. We call this *local communication*. For operations with global Ω–data (as for instance the calculation of a global defect norm or the solution of the coarsest grid problem) the task T_i exchanges data with the whole task network $\{T_i\}$. This kind of communication we will call *global communication*.

Parallel Smoother

In order to avoid recursive data dependencies between the macroelement tasks we split the one loop of the sequential smoothing procedure over all elements in the whole domain Ω into several loops over the elements in the macroelements M_i which can be executed independently and in parallel. That means all tasks T_i update at the same time their own unknowns on M_i. As a consequence, neighbouring tasks T_i and T_j produce different velocity values at the nodes x on the common macroelement boundary $\Gamma_{ij}=\partial M_i \cap \partial M_j$. Therefore, the different values are exchanged by local communication and then each two tasks T_i and T_j compute for all nodes $x \in \Gamma_{ij}$ the following equalized value

$$u(x) := \frac{1}{2}\left(u(x)_{|M_i} + u(x)_{|M_j}\right) . \tag{6}$$

So, our parallel smoother differs from the sequential one by the fact that within one smoothing iteration the spreading of new updated information is restricted only to the macroelements. Therefore, we would expect a slower convergence of the parallel multigrid algorithm compared to the sequential one. However, numerical tests show (see [11]) that there is only a small difference in the convergence rates of both methods such that we achieve a good numerical efficiency for our parallel solver even in the case of a large number of macroelements.

Coarsest Grid Solver

In order to solve the global coarsest grid problems we add an extra solver task S on an extra processor P_S to the processor network such that all data dependencies are concentrated on S. At the beginning of the n-th nonlinear step the task S waits for the restricted velocity vector $u^{\text{old}}=u^{n-1}$ on grid level $\ell=1$ which will be transmitted by a global communication from the task network $\{T_i\}$. Once u^{old} has been received, the matrix $A=A(u^{\text{old}})$ and the augmented Lagrange matrix $C=A+rBB^{\mathsf{T}}$ are generated and the LU–factorization of C is computed. There is an additional parallelism between the solver task S and the macroelement tasks T_i in the following sense. At the same time, when S deals with the generation of A and C and the factorization of C, the tasks T_i can continue their work within the first multigrid cycle until the first coarsest grid problem occurs. Then, within the multigrid cycle, task S waits to get from the task network $\{T_i\}$ the global right hand side vector of the coarsest grid problem. After this vector has been received, the solution vector is computed by the augmented Lagrange iteration and will be sent back to all other tasks T_i via global communication.

Results

In order to study the scalability properties of our parallelization we consider as a test example the driven cavity problem with Reynolds number Re=2000 where the macroelements are defined by an $(N\times N)$-decomposition of the unit square Ω. Then, the total number of elements on the finest grid level ℓ is $N^2 4^{\ell-1}$ and the number of used processors is N^2+1, i.e. with N we simultaneously increase the problem size and the number of processors.

In Table 1 we show the behaviour of our parallel code for two different hardware situations, the Parsytec machines GCel with processor T805 and the GCPP with processor MPC601. t_{mg} denotes the CPU-time in seconds for one multigrid cycle on the finest grid level ℓ=6. t_l, t_g and t_s are the times for local communication, global communication and for solving the coarsest grid problems within one multigrid cycle. t_g only measures the time for computing global norms whereas the time for the global communication of coarsest grid

Table 1: CPU times on two different hardware systems for level ℓ=6

	GCel					GCPP				
NxN	t^{all}	t_{mg}	t_l	t_g	t_s	t^{all}	t_{mg}	t_l	t_g	t_s
2x2	893	10.7	.50	.003	.09	41	.50	.17	.001	.01
3x3	953	11.2	.88	.008	.25	54	.64	.30	.003	.03
4x4	1144	10.5	.96	.012	.47	73	.68	.37	.004	.04
5x5	1274	11.6	1.08	.017	1.04	88	.81	.47	.007	.07
6x6	1397	13.3	1.18	.019	2.12	95	.91	.50	.008	.12
8x8	1936	18.6	1.61	.031	6.97	109	1.03	.53	.009	.21

vectors between the macroelement-tasks and the solver-task is contained in t_s. By t^{all} we denote the total CPU-time for solving the driven cavity problem. Switching from the GCel to the GCPP the communication becomes between 2 and 3 times faster and the speed of arithmetical operations is increased by a factor of about 34. Therefore, on the GCPP the time t_l for local communication is the dominating part of t_{mg} whereas on the GCel for an increasing number of macroelements the time t_s becomes the largest portion since the one solver-task needs more and more time to solve the coarsest grid problems of increasing size. Nevertheless on the GCPP the time t_s would become dominant too if we would further increase the number of macroelements.

Table 2: Loss of efficiency on the GCel

	Level 4			Level 5			Level 6		
NxN	β_c	β_s	$\tilde{e}$	β_c	β_s	$\tilde{e}$	β_c	β_s	$\tilde{e}$
2x2	0.16	0.06	0.77	0.09	0.02	0.89	0.05	0.01	0.95
3x3	0.23	0.14	0.63	0.14	0.06	0.80	0.08	0.02	0.90
4x4	0.24	0.24	0.52	0.16	0.11	0.73	0.09	0.04	0.87
5x5	0.18	0.45	0.37	0.15	0.20	0.65	0.09	0.09	0.82
6x6	0.12	0.65	0.23	0.12	0.37	0.51	0.09	0.15	0.76
8x8	0.06	0.85	0.09	0.09	0.66	0.25	0.09	0.37	0.55

Table 3: Loss of efficiency on the GCPP

	Level 6			Level 7			Level 8		
NxN	β_c	β_s	$\tilde{e}$	β_c	β_s	$\tilde{e}$	β_c	β_s	$\tilde{e}$
2x2	0.33	0.025	0.65	0.15	0.009	0.84	0.07	0.003	0.93
3x3	0.46	0.045	0.49	0.25	0.018	0.73	0.11	0.006	0.88
4x4	0.54	0.057	0.41	0.30	0.024	0.68	0.15	0.008	0.84
5x5	0.57	0.084	0.34	0.34	0.039	0.62	0.16	0.014	0.82
6x6	0.54	0.127	0.33	0.34	0.061	0.60	0.17	0.022	0.80
8x8	0.51	0.198	0.29	0.34	0.105	0.56	0.18	0.039	0.78

In Table 2 and Table 3 we show for different finest grid levels the loss of efficiency on the GCel and the GCPP, respectively. Here, β_c is defined as the portion of t^{all} for local and global communication (i.e. $\beta_c t^{all}$ is the time for communication). β_s denotes the portion for solving the coarsest grid problems. The quantities β_c and β_s can be regarded approximately as the losses of efficiency (see [10]) such that the value $\tilde{e} := 1 - \beta_c - \beta_s$ gives a rough measure for the parallel efficiency. We see that the losses of efficiency β_c and β_s are reduced if we have a larger finest grid level ℓ. That means, in order to achieve a good parallel effieciency we have to assign a sufficiently large amount of arithmetical work to each processor. If we have a large number of macroelements and a small finest grid level ℓ then it would be more efficient to map the macroelement-tasks to a smaller number of processors. Furthermore, we see that for fixed finest grid level ℓ and increasing number of macroelements the solver portion β_s is the most rapidly increasing part among the losses of efficiency. On the GCel β_s becomes dominant already for a relatively small number of macroelements whereas on the GCPP this would happen only for a much larger number.

The Massive Parallel Case

Now we want to consider the case when we have a large number of macroelements. We have seen that the most important thing is to reduce the time for the coarsest grid solver.

One way could be to choose a better numerical method for this solver. Table 4 shows the total CPU-time t^{all} for the finest grid level ℓ=6 if we choose the following coarsest grid solvers: (a) the above defined smoother "Vanka", (b) the SIMPLE-iteration "Simple", (c) the augmented Lagrange method "augm.L." and (d) the void solver "no-slv" which does nothing, i.e. where the coarsest-grid-solver-step is omitted.

We see that for a larger number of macroelements the SIMPLE-solver beats the augmented Lagrange method. However, in the massive parallel case all coarsest grid solvers on a single solver-task are too expensive such that the do-nothing-version despite of its poor convergence rate gives better results.

Table 4: Influence of the solver-method for one solver-task on the CPU-time for ℓ=6

NxN	Vanka	Simple	augm.L.	no-slv
2x2	897	888	893	960
3x3	882	832	953	1238
4x4	1165	1160	1144	1684
5x5	1429	1311	1274	2102
6x6	1648	1478	1397	2533
8x8	2606	1698	1936	3357
10x10	4507	2214	3998	4138
20x20	42990	11910	*	7360

Another way is to use more than one processor to solve the coarsest grid problems. By recursive bisection of the set of all macroelements $\{M_i\}$ we subdivide the domain Ω into 2^s solver-task-regions $\Omega_n := \bigcup_{j \in I_n} M_j$. Then, we split the work and data of the one solver-task S into 2^s solver-tasks $\{S_n\}$ where S_n is responsible for all operations of the solver with data associated with Ω_n. The task S_n is mapped to the same processor as the root-task $T_{r(n)}$ of the subtree of all macroelement-tasks T_i with $i \in I_n$ such that the total number of processors used for the tasks $\{T_i\}$ is not increased.

Table 5: Total CPU-times for single and parallelized coarsest grid solvers for ℓ=6

	Vanka-1	Vanka-par		Simple-1	Simple-par		Vanka-all
NxN	t^{all}	$\#S$	t^{all}	t^{all}	$\#S$	t^{all}	t^{all}
2x2	897	4	885	888	2	897	969
3x3	882	4	848	832	2	841	980
4x4	1165	8	1095	1160	4	1134	1250
5x5	1429	8	1246	1311	2	1228	1380
6x6	1648	16	1302	1478	4	1369	1324
8x8	2606	32	1452	1698	4	1433	1488
10x10	4507	32	1933	2214	4	1774	1766
20x20	42990	128	2873	11910	16	3578	2347
24x24	*	256	2274	19150	64	3747	1979
32x32	*	256	2661	*	512	3694	2374

Table 5 shows the total CPU-times t^{all} for solving the driven cavity problem for the Vanka-solver with one solver-task ("Vanka-1"), the parallelized Vanka-solver with $\#S$ solver-tasks ("Vanka-par"), the SIMPLE-solver with one solver-task ("Simple-1"), the parallelized SIMPLE-solver ("Simple-par") and the version ("Vanka-all") where the work of the Vanka-solver is distributed to all macroelement-tasks $\{T_i\}$ without using a solver task. With $\#S$ we present the number of used solver-tasks that gave the best results, i.e. the minimum total time t^{all}.

We see that for a large number of macroelements the parallelization of the coarsest grid solver leads to an essential improvement of the parallel efficiency. The parallelized SIMPLE-solver is not so efficient as the parallelized Vanka-solver since it needs much more communication. It is somewhat surprising that the version "Vanka-all" gives the best results in the case of a large number of macroelements.

4. ADAPTIVE PARALLEL METHOD

Adaptive methods are used in order to reduce the size of the arising system of equations. The philosophy is to put the computational work, i.e. the unknowns, into regions where a large local error of the computed solution is assumed. Accordingly, for these methods one needs additional components like a posteriori error estimators, local mesh refinement and in the parallel context dynamic load balancing.

A posteriori error estimators

Here we use the non–conforming $P1/P0$ triangular element pair. Elements which should be refined are divided into four smaller triangles by connecting the midpoints of the edges ("red refinement"). Thus, we obtain hanging nodes which are handled by several closure rules ("green refinement").

In order to decide which triangles have to be refined an a posteriori error estimator is needed. Residual based error estimators for the Navier–Stokes equations and non–conforming elements have been proposed and analyzed in [16], [17].

Let n_E be the unit outer normal vector on an edge E of an element T. Then $[u]_E$ denotes the jump of a function u across E in the direction of n_E

$$[u]_E = \lim_{t\to+0} u(x+tn_E) - \lim_{t\to+0} u(x-tn_E) \quad \forall x \in E.$$

Moreover, f_h is an approximation to f obtained by numerical integration, h_T the size of T and h_E the length of E.

If (u,p) is the solution of the Navier–Stokes equations (1) and (u_h,p_h) the computed discrete solution then the error estimate for the graph norm becomes

$$\|u-u_h\|_1 + \|p-p_h\|_0 \le C\left(\sum_{T\in\mathcal{T}} \eta_T^2\right)^{1/2} \tag{7}$$

with

$$\eta_T^2 = h_T^2\|f_h - (u_h\cdot\nabla)u_h\|_{0,T}^2 + \|\nabla\cdot u_h\|_{0,T}^2 + \sum_{E\in\partial T} h_E\|\left[\frac{1}{Re}n_E\cdot\nabla u_h - p_h n_E\right]_E\|_{0,E}^2.$$

The error in the L^2–norm for the velocity can be estimated in the form

$$\|u-u_h\|_0 \le C\left(\sum_{T\in\mathcal{T}} \tilde\eta_T^2\right)^{1/2} \tag{8}$$

with

$$\tilde\eta_T^2 = h_T^4\|f_h - (u_h\cdot\nabla)u_h\|_{0,T}^2 + h_T^2\|\nabla\cdot u_h\|_{0,T}^2 + \sum_{E\in\partial T} h_E^3\|\left[\frac{1}{Re}n_E\cdot\nabla u_h - p_h n_E\right]_E\|_{0,E}^2.$$

Both estimators η_T and $\tilde\eta_T$ contain three terms. The first and the second one are the residuals of the momentum and continuity equation of the strong formulation of the Navier–Stokes equations. The third term, which contains the jumps across edges, is a connection between the weak and strong formulation for non–smooth functions (like the discrete solution).

The above a posteriori error estimators have been used only to generate adaptively refined meshes. If we want to use them really for estimating the error we need quantitative estimates for the constants C in (7) and (8). The great difficulties connected with this problem have been focussed [4] and [6], the research in this direction is just at the beginning. Both error estimators, η_T and $\tilde\eta_T$, which are different only in the powers of h_T and h_E, in general produce different meshes with different solutions. This can be seen already for convection diffusion problems. As an example we show in Figure 3 the mesh and the resolution of an interior boundary layer for the model problem

$$\begin{aligned} -10^{-6}\Delta u + (3,1)\cdot\nabla u + u &= 100r(r-0.5)(r-\sqrt{2}/2) \quad \text{in} \quad \Omega \\ u &= 0 \quad \text{on} \quad \partial\Omega \end{aligned}$$

where Ω is an L-shaped domain and $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$.

A more complete study for different error estimators and error indicators for convection diffusion problems is contained in [5].

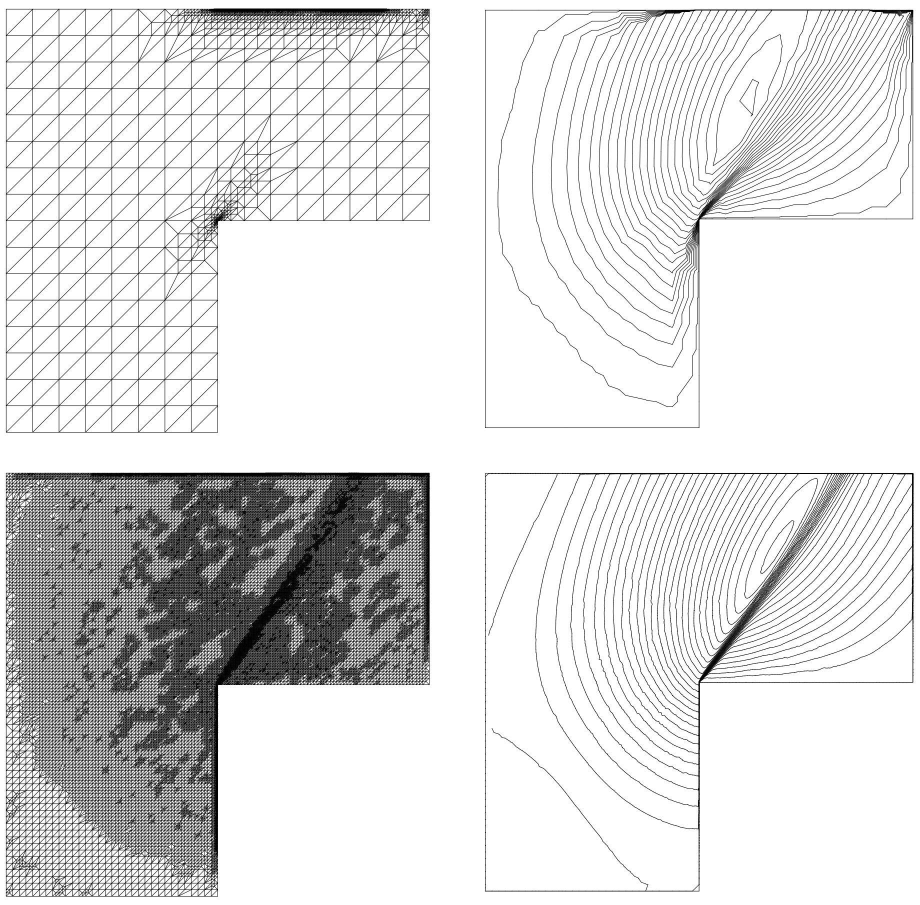


Figure 3: Comparison between grids and solution produced with H^1–norm error estimator (above) and a L^2–norm error estimator (below)

The solver

We use the fixed–point iteration (4) as the outer nonlinear iteration. Alternatively to the above approach where the multigrid technique is applied directly to the coupled system we use a different concept which seems to be more suitable for non–stationary problems, [14].

We consider the Schur complement system with respect to the pressure and solve the arising equations by a preconditioned Richardson iteration

$$p^{l+1} = p^l + \alpha(B^T C^{-1} B)^{-1}(-B^T A^{-1} B p^l + B^T A^{-1} f - g), \quad l = 0, 1, \ldots, L-1\,, \tag{9}$$

where C is an invertible matrix and $\alpha > 0$ a damping factor. In each iteration step one has to solve a linear system of the abstract form

$$B^T C^{-1} B x = s. \tag{10}$$

We choose $C = \hat{A}$ where $\hat{A}^{-1}$ corresponds to a number of multigrid cycles applied to a system of the form $Ay = b$ and we solve (10) by GMRES. All components of the solver, multigrid as well as GMRES, are well suited for parallel computing. After having stopped the iteration (9) with p^L we compute u from the equation

$$Au = f - Bp^L. \tag{11}$$

An important ingredient of the solver is the multigrid method for approximating A^{-1} in the right hand side of (9) and for solving (11). As grid transfer operation we use the L^2–projection between meshes. The smoother is a damped block Jacobi with inner ILU(β) iterations. The number and form of the blocks depend on the distribution of the elements on the processors. Unknowns on the interfaces between processors can be handled by one processor only or by both and afterwards the results are averaged. The coarse grid system is solved either directly or iteratively, depending on its size.

The implementation of the algorithm described above has been done using the program package *ugp1.0* developed by P. Bastian [1]. In particular data structures, load balancing routines and the parallel overhead of this program have been used. However, the existing data structures had to be extended in order to handle multidimensional data and non–conforming elements.

5. EXTENSIONS

Extensions of the proposed method to the Boussinesq approximation of the Navier–Stokes equations and to the non-stationary case have been studied too. In the first case, additionally the energy equation

$$-\frac{1}{\mathrm{RePr}}\Delta T + u \cdot \nabla T = 0 \quad \text{in } \Omega \tag{12}$$

has to be included and the right hand side f in (1) is given by the buoyancy force

$$f = \frac{\mathrm{Ra}}{\mathrm{Re}^2\mathrm{Pr}} T\vec{g}\,. \tag{13}$$

The resulting coupled problem for the unknowns u, p, T is solved by a multigrid method which is similar to the one described in Section 3. As an example the flow through a channel over a hot obstacle has been calculated for two different Rayleigh numbers $\mathrm{Ra} = 10^6$ and $\mathrm{Ra} = 10^8$ where $\mathrm{Pr} = 0.71$ and $\mathrm{Ra} = \mathrm{Re}\,\mathrm{Pr}^2$. In Figure 4 and 5 the streamlines of the velocity field and the isolines for the temperature, respectively, are shown.

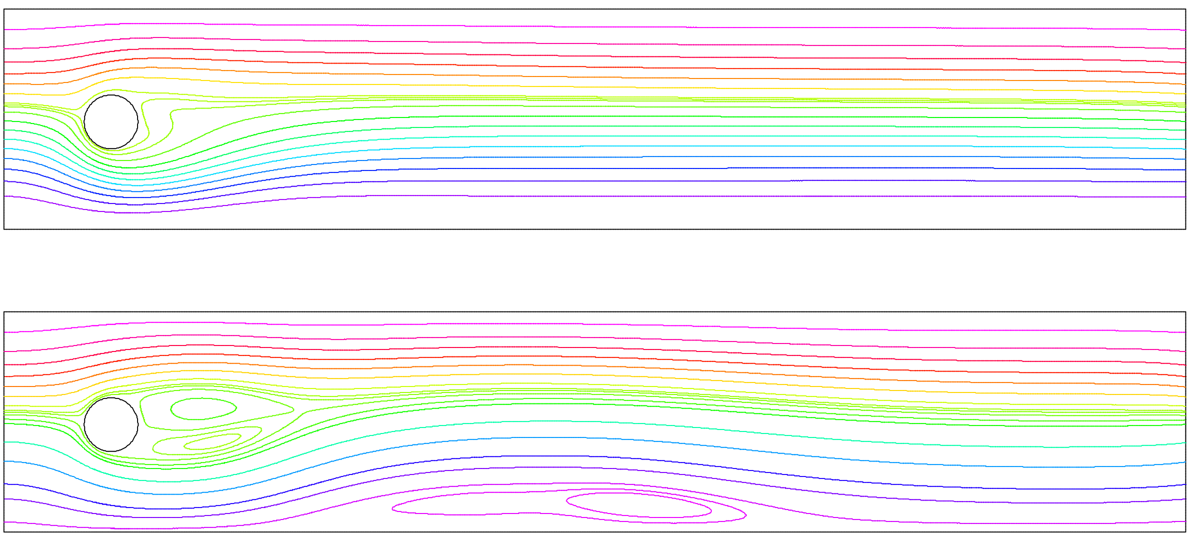


Figure 4: Streamlines, $Ra = 10^6$ and $Ra = 10^8$

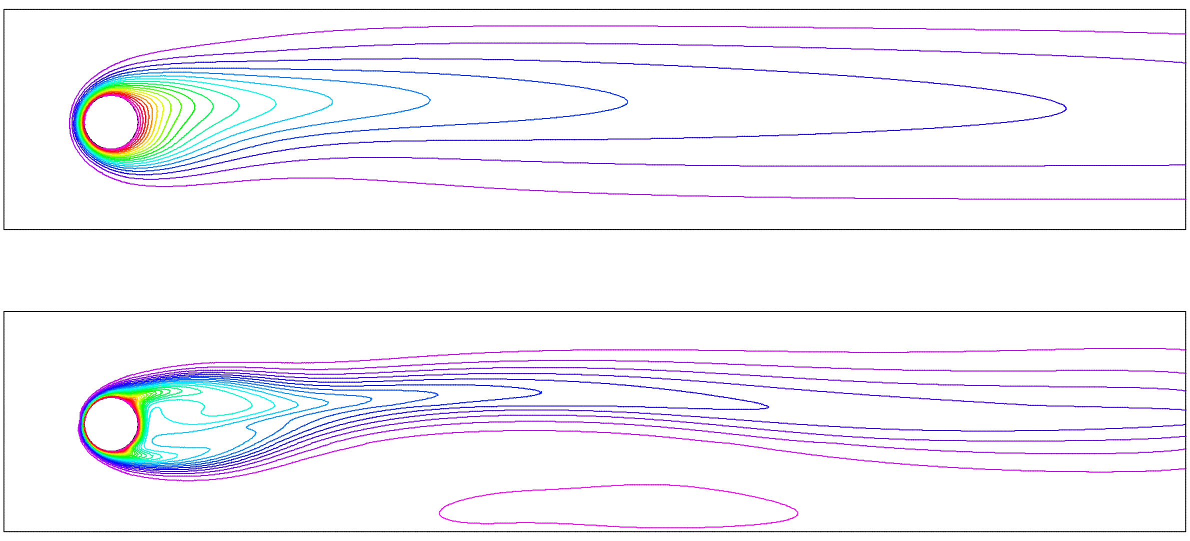


Figure 5: Iso-temperature lines, $Ra = 10^6$ and $Ra = 10^8$

For solving the non-stationary Navier–Stokes equations, an implicit second order in time discretization based on the backward difference formula

$$\frac{\partial u}{\partial t}\Big|_{t=t_n} \approx \frac{3u(t_n) - 4u(t_{n-1}) + u(t_{n-2})}{2\Delta t}\,. \tag{14}$$

is used. The arising problem for $(u(t_n), p(t_n))$ is solved by the stationary solver described in Section 4.

As an example we present the flow through the channel $[0,4]\times[0,1]$ with $Re = 100$, changing inflow parts and fixed outflow part. At the outflow part $\{(4,y) : 0.25 \le y \le 0.75\}$ we assume a fixed developed parabolic velocity profile u_o. The inflow parts are $\{(0,y) : 0.25 \le y \le 0.75\}$ with the inflow profile $(1-t)\,u_o$ and $\{(x,0) : 1 \le x \le 1.5\}$ with $t\,u_o$ for $t \in [0,1]$. For $t \ge 1$ the inflow does not change any longer. We have started with the solution of the stationary equations. The mesh has been adapted by an L^2-error estimator similar to that presented in Section 4 with an additional term that estimates the error in time (for more details see [17]). The resulting meshes for some time steps are shown in Figure 6 together with the corresponding vector plots of the velocity.

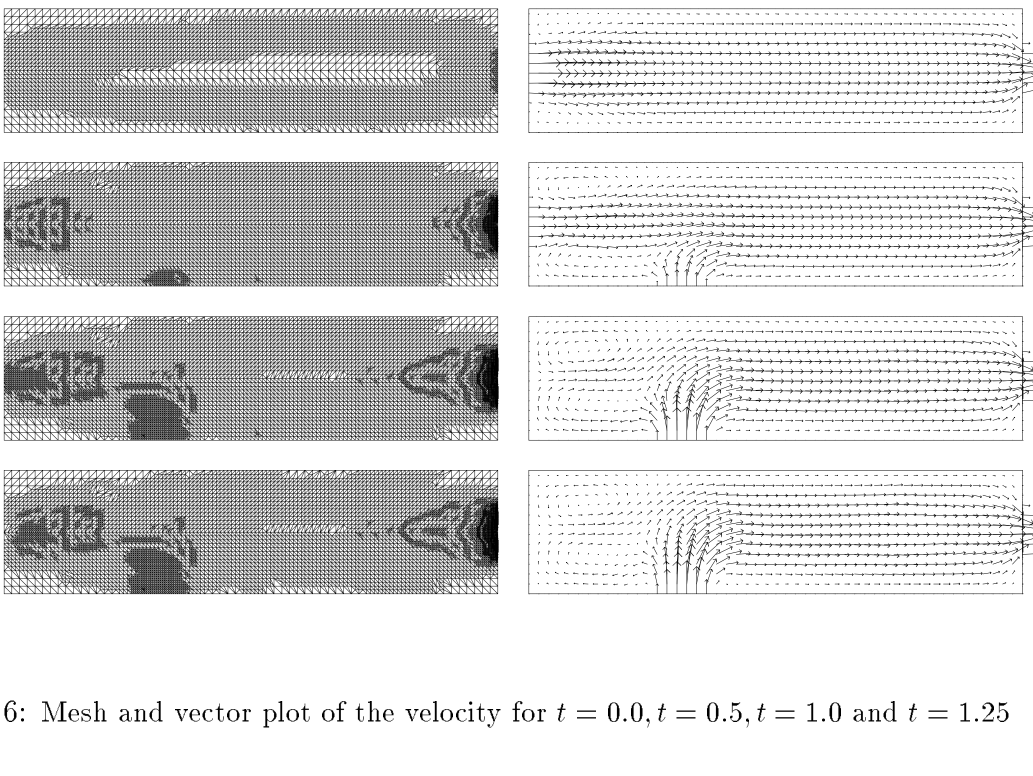


Figure 6: Mesh and vector plot of the velocity for $t = 0.0, t = 0.5, t = 1.0$ and $t = 1.25$

ACKNOWLEDGEMENT

For our numerical tests we used the GCel-1024 of the Paderborn Center for Parallel Computing. Some of the pictures have been produced by the package GRAPE.

References

REFERENCES

[1] P. Bastian, *Parallele adaptive Mehrgitterverfahren.* PhD thesis, Universität Heidelberg, 1994.

[2] M. Crouzeix, P.-A. Raviart, Conforming and nonconforming Finite Element Methods for solving the stationary Stokes equations. *RAIRO Anal. Numer.* **7,** 33-76 (1973).

[3] O. Dorok, G. Lube, U. Risch, F. Schieweck, L. Tobiska, Finite Element discretization of the Navier–Stokes equations. In: E.H. Hirschel (Ed.): Flow Simulation with High–Performance Computers I, *Notes on Numerical Fluid Mechanics*, Vol. 38, Vieweg-Verlag 1993, pp. 67-78.

[4] K. Eriksson, D. Estep, P. Hansbo, C. Johnson, Introduction to adaptive methods for differential equations. *Acta Numerica*, 1-54 (1995).

[5] V. John, A comparison of some error estimators for convection diffusion problems on a parallel computer. Preprint 12/94, Otto-von-Guericke Universität Magdeburg, Fakultät für Mathematik, 1994.

[6] C. Johnson, On computability and error control in CFD. *Int. Jour. Num. Meth. Fluids* **20,** 777-788 (1995).

[7] H. Reichert, G. Wittum, Solving the Navier-Stokes equations on unstructured grids. In: E.H. Hirschel (Ed.): Flow Simulation with High–Performance Computers I, *Notes on Numerical Fluid Mechanics*, Vol. 38, Vieweg-Verlag 1993, pp. 321-333.

[8] H. Rentz-Reichert, G. Wittum, A comparison of smoothers and numbering strategies for laminar flow around a cylinder. In this publication.

[9] R. Rannacher, S. Turek, Simple Nonconforming Quadrilateral Stokes Element. *Numer. Methods Partial Differential Equations* **8,** 97-111 (1992).

[10] F. Schieweck, A parallel multigrid algorithm for solving the Navier–Stokes equations. *IMPACT Comput. Sci. Engrg.* **5,** 345-378 (1993).

[11] F. Schieweck, Multigrid Convergence Rates of a Sequential and a Parallel Navier–Stokes Solver. In: Hackbusch, W., Wittum, G. (Eds.): Fast Solvers for Flow Problems, *Notes on Numerical Fluid Mechanics*, Vol.49, Vieweg-Verlag 1995, pp. 251-262.

[12] F. Schieweck, L. Tobiska, A nonconforming finite element method of upstream type applied to the stationary Navier–Stokes equation. *RAIRO Modél. Math. Anal. Numér.* **23,** 627–647 (1989).

[13] L. Tobiska, A note on the artificial viscosity of numerical schemes. *Comp. Fluid Dynamics* **5,** 281–290 (1995).

[14] St. Turek, A comparative study of time stepping techniques for the incompressible Navier-Stokes equations: From fully implicit nonlinear schemes to semi-implicit projection methods. to appear, 1995.

[15] S. Vanka, Block-implicit multigrid calculation of two-dimensional recirculating flows. *Comput. Methods Appl. Mech. Engrg.* **59,** 29-48 (1986).

[16] R. Verfürth, A posteriori error estimators for the Stokes Stokes equations. II Nonconforming discretizations. *Numer. Math.* **60,** 235-249 (1991).

[17] R. Verfürth, A posteriori error estimates for nonlinear problems. Finite element discretizations of parabolic equations. Bericht 180, Ruhr-Universität Bochum, Fakultät für Mathematik, 1995.