MULTIGRID CONVERGENCE RATES OF A SEQUENTIAL AND A PARALLEL NAVIER–STOKES SOLVER*

Friedhelm Schieweck
Institute for Analysis and Numerical Mathematics, University Magdeburg
PSF 4120, D-39016 Magdeburg, Germany

Abstract

We consider a sequential and a parallel version of a multigrid method for a finite element discretization of the stationary incompressible Navier–Stokes equations. The sequential algorithm is based on a blockwise Gauss–Seidel smoother which passes successively through all elements of the domain using the already updated values from the previous elements. In the parallel version this strategy is only possible within each subdomain of some domain decomposition. We study how this algorithmical difference affects the convergence rate of the parallel multigrid method compared with the sequential one and investigate its effect on the numerical efficiency of our parallel solver. Especially we are interested in the case of a large number of subdomains.

1 Introduction

If we today consider the field of Computational Fluid Dynamics (CFD) and its requirements for the future we see that we always have to do with an increasing problem size. One reason is that we wish to have more and more grid points in order to get a higher accuracy of the solution. That means, for solving the corresponding problems, we need more and more storage capacity and CPU power. In order to satisfy these increasing requirements we need for the future a parallel computer with more and more processors. Therefore, we should construct our parallel CFD solver in such a way that it will be also reasonable for larger problems with a larger number p of employed processors. That means we want the *parallel efficiency* E to behave acceptable if we increase both the problem size and the number p of processors. This efficiency E is defined by

$$E = \frac{T_1^{seq}}{pT_p^{par}} \tag{1}$$

*to be published in: Proceedings of the 10th GAMM-Seminar, Kiel, January 14-16, 1994, W.Hackbusch, G.Wittum (eds), *Fast Solvers for Flow Problems*, Vieweg-Verlag.

where T_1^{seq} denotes the CPU time of the sequential algorithm for solving the actual problem on one processor and T_p^{par} the CPU time of the parallel algorithm on p processors. For the sequential algorithm we should take a fast one (for instance a multigrid solver) in order to get a realistic impression about the quality of the parallel solver. The efficiency E can be regarded as the product of two other efficiencies

$$E = E^{num} E^{par} \tag{2}$$

with

$$E^{num} = \frac{T_1^{seq}}{T_1^{par}}\,, \qquad E^{par} = \frac{T_1^{par}}{pT_p^{par}}\,, \tag{3}$$

where T_1^{par} denotes the CPU time of the parallel algorithm on one processor (i.e. there are no communication costs). E^{num} is called the *numerical efficiency*. It shows us how fast the parallel algorithm converges compared with the sequential one. The quantity E^{par} is the *efficiency of the parallel implementation* which tells us how large the losses are that come from communication and idle times in the implementation of the parallel algorithm on p processors.

Our parallel solver is based on a decomposition of the domain into several subdomains each of which is assigned to another processor. In order to get a reasonable efficiency of the parallel implementation we have to avoid recursive data dependencies between the subdomain processors. That means, in the parallel version we have to modify the Gauss–Seidel type smoother of our sequential multigrid solver such that the unknowns are successively updated only within a subdomain. This leads to a difference in the convergence rates of the parallel and sequential algorithm. Hence the numerical efficiency E^{num} depends on the subdomain decomposition and therefore on the number p of processors. Thus, if we want to achieve a reasonable efficiency E also for large p, we have to ensure at least the condition that (because of $E^{par} \leq 1$) the numerical efficiency E^{num} does not become small for increasing p. The aim of this paper is to check this condition by means of a model problem. This leads to the investigation of the multigrid convergence rates for the sequential and the parallel algorithm. Of course, we have to guarantee also that E^{par} does not tend to zero for increasing p which is a nontrivial problem that has to be solved for the future (see [3] for a discussion). Therefore, the above condition for E^{num} is only a necessary condition for our parallel solver to be reasonable for the future where we want to solve larger problems on a larger number of processors.

In this paper we consider as a CFD model the stationary incompressible Navier–Stokes equations in primitive variables

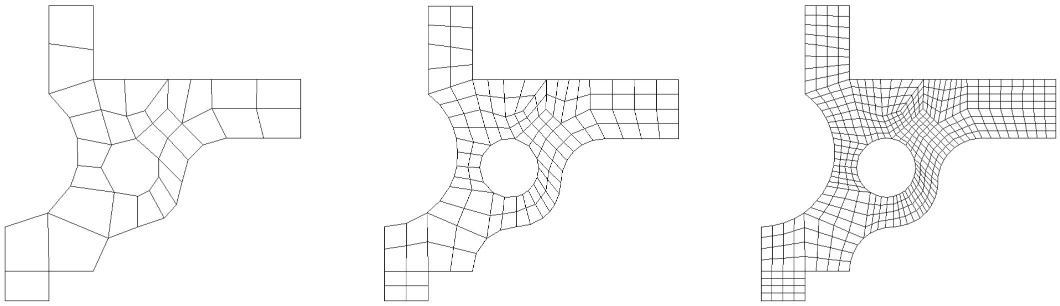
$$-\,\nu\triangle u + (u\cdot\nabla)u + \nabla p = f\,, \quad \nabla\cdot u = 0 \quad \text{in } \Omega\,, \qquad u = g \quad \text{on } \partial\Omega \tag{4}$$

where u and p denote the unknown velocity and pressure field, respectively, in a bounded domain $\Omega \subset \mathbf{R}^2$, f a given body force, g the prescribed Dirichlet boundary data and ν=$1/Re$ with the *Reynolds number Re*.

2 Grids and Discretization

We use a finite element method based on quadrilateral elements. For our multigrid algorithm we need a sequence of different fine grids. We start with a decomposition $\Omega = \bigcup_i M_i$

of the domain Ω into several quadrilateral elements M_i which are called *macroelements*. These macroelements define the finite element mesh on grid level $l = 1$. All finer grids on a level $l>1$ can be defined separately (i.e. also in parallel) on each macroelement by uniformly dividing the coarse grid cells of the previous grid level $l-1$ into four finer cells. At curved boundary parts we make the usual modification that we shift the bisection point of the corresponding coarse grid edge to the boundary. Figure 1 shows an example of such a grid generation. Thus we have defined some kind of blockstructured grids



grid level 1 grid level 2 grid level 3

Figure 1: *Example of a multilevel grid sequence*

where the blocks correspond to the macroelements M_i. Such grids on the one hand give a relatively high flexibility for complex geometries and on the other hand they guarantee the local regularity of the grid on each macroelement which is important for the stability and convergence of our finite element discretization. Moreover, we get a natural and nearly optimal load balancing for a parallelization since we have the same number of elements on all macroelements. Therefore, it is reasonable to assign one processor to each macroelement.

For the finite element approximation of the Navier–Stokes problem (4) we use nonconforming rotated bilinear Q_1^{rot} elements for the velocity and piecewise constant Q_0 elements for the pressure. The degrees of freedom are the velocity values at the midpoints of the element edges and the pressure values at the centres of the elements (see Figure 2). This

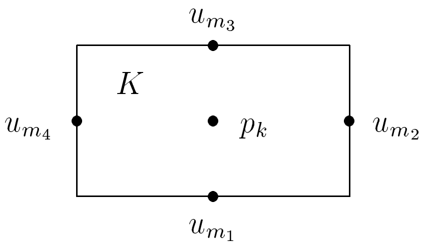


Figure 2: *Unknowns belonging to an element K*

element pair was introduced and analyzed by Rannacher and Turek [2] for the Stokes problem. They showed the uniform Babuška-Brezzi stability and optimal error estimates under the assumption that the mesh is sufficiently regular which is satisfied for our blockstructured grids. For the Navier–Stokes problem we use an upwind discretization of the additional convective term in order to stabilize the numerical scheme for high Reynolds numbers. We will omit here the details of the discretization (see for instance [5],[3],[4]) and give only the structure of the corresponding discrete Navier–Stokes problem. On some finest grid level $l{=}k$ we want to find 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

$$L(y)\,y = F \tag{5}$$

with

$$L(y) := \begin{pmatrix} A(u) & B \\ B^{\mathsf{T}} & 0 \end{pmatrix}, \quad y := \begin{pmatrix} u \\ p \end{pmatrix}, \quad F := \begin{pmatrix} f \\ g \end{pmatrix} \tag{6}$$

where the matrix block $A(u)$ contains the discretization of the nonlinear convection term.

3 Sequential Multigrid Solver

We use a linear multigrid method within an outer nonlinear iteration. On the finest grid level $l{=}k$ we solve our nonlinear problem (5) by means of the *simple iteration* where the idea is to linearize in the n-th step the nonlinear convection term by

$$(u \cdot \nabla)\,u \;\approx\; (u^{n-1} \cdot \nabla)\,u^n\,.$$

That means, in the n-th step of the outer iteration we first solve the linear system of equations

$$L(y^{n-1})\,\bar{y}^n = F \tag{7}$$

and get the new iterate $y^n{=}(u^n{,}p^n)^{\mathsf{T}}$ after an underrelaxation step

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

with some parameter ω (we have used $\omega{=}0.9$).

For solving the linear problems (7) we apply a multigrid algorithm which is characterized as follows (for details see [3]). We use an F–cycle since it is a good compromise between the cheap V–cycle and the more accurate but expensive W–cycle. The coarse grid correction is damped by a factor $\alpha{=}0.8$. For the grid transfer we use standard finite element prolongation and restriction with a natural modification at points, where the finite element function is discontinuous (note that we have nonconforming elements). The coarsest grid problems are solved by an augmented Lagrange algorithm due to Fortin/Glowinski [1]. We use a blockwise Gauss–Seidel smoother similar to that proposed by Vanka [6].

In the following we want to explain the smoother in a little more detail. Let us indicate by

$$Ly \;=\; F \tag{8}$$

with $L{=}L(y^{n-1})$ the global system of equations which has to be solved on some actual grid level within the multigrid algorithm. With any finite element K on the current mesh we can associate the following modified local system

$$\tilde{L}_K y_K \;=\; \tilde{F}_K \tag{9}$$

with

$$\tilde{L}_K := \begin{pmatrix} D_K & b_K \\ b_K^\mathsf{T} & 0 \end{pmatrix}, \quad y_K := \begin{pmatrix} u_K \\ p_K \end{pmatrix}, \quad \tilde{F}_K := \begin{pmatrix} \tilde{f}_K \\ \tilde{g}_K \end{pmatrix} \tag{10}$$

which corresponds to the local block y_K of all 9 unknowns belonging to the element K. These unknowns are the 8 velocity values at the 4 midpoints of the element edges and the one constant pressure value p_K on element K (see Figure 2). D_K denotes the diagonal matrix consisting of the diagonal entries of the corresponding 8×8 block A_K of the matrix block A of L in (8) and b_K the block of the column vector of matrix B which is associated with the unknown p_K. $\tilde{F}_K$ is derived from the corresponding block vector F_K of F in (8) by taking the parts of Ly, which are not associated with $\tilde{L}_K$, from the left to the right hand side. Therefore, $\tilde{F}_K$ depends on the vector y which was used for its generation. The local 9×9 system (9) has a very simple structure and can be solved by a few arithmetical operations. Now, our smoother in the sequential multigrid version is defined as follows.

<u>*Sequential Smoothing Procedure:* $\;y^{\text{old}} \rightarrow y^{\text{new}}$</u>

(a) initialize $y := y^{\text{old}}$;
for all elements $K \subset \Omega$ **do**
  solve the local system $\tilde{L}_K y_K = \tilde{F}_K$ generated with y;
  update $y_K \rightarrow y$;
enddo

(b) $y^{\text{new}} := y^{\text{old}} + \omega_S (y - y^{\text{old}})$;

In the underrelaxation step (b) we have used the value $\omega_S = 0.9$.

4 Parallel Multigrid Solver

We assign to each macroelement M_i a task T_i which is responsible for all operations of our algorithm with data belonging to $\overline{M}_i := M_i \cup \partial M_i$. Then we map the set of all tasks $\{T_i\}$ to a given set of processors $\{P_j\}$. Let us assume that we have enough processors such that we can assign to each task another processor (otherwise we would have to solve a load balancing problem to get a suitable mapping). Thus, the whole work and data of the multigrid algorithm is distributed to several tasks T_i which are executed in parallel on different processors. In order to perform operations with global Ω-data or with data belonging to the common edge of two macroelements the tasks have to exchange data by communication. However, the communication and also idle times for the synchronization within the algorithm are a matter of the parallel implementation and do not have any influence on the numerical algorithm.

From the algorithmical point of view, which is our primary interest in this paper, the parallel and sequential multigrid solver are different only in the smoothing procedure. All other parts are equivalent and differ only by their implementation. In the parallel smoother, we split up the one loop of the sequential version over all elements of the whole domain Ω into several loops over the elements within the macroelements which are executed in parallel. That means, all tasks T_i update at the same time their own unknowns on $\overline{M}_i$. Consequently, at velocity nodes on a common edge of two macroelements M_i and M_j the tasks T_i and T_j produce different values. In an additional step, these different values are equalized by taking the mean value. Let us denote by $\Theta(i)$ the set of all indices k of the global solution vector $y{=}(u,p)^{\mathsf{T}}$ for which the component y_k belongs to $\overline{M}_i$. Then, our parallel smoother can be described as follows.

Parallel Smoothing Procedure: $y^{\text{old}} \to y^{\text{new}}$

for all macroelements M_i **do parallel**

(a) initialize $y^{M_i} = (u^{M_i}, p^{M_i})^{\mathsf{T}}$ by $y^{M_i} := y^{\text{old}}$;
  for all elements $K \subset M_i$ **do**
    solve the local system $\tilde{L}_K y_K = \tilde{F}_K$ generated with y^{M_i} ;
    update $y_K \to y^{M_i}$;
  enddo

(b) { update u at the edges of M_i }
  for all macroedges $\Gamma_{ij} = \partial M_i \cap \partial M_j$, $j \neq i$, of M_i **do**
    for all u-nodes $x_m \in \Gamma_{ij}$ **do**
      $u_m^{M_i} := \frac{1}{2}\left(u_m^{M_i} + u_m^{M_j}\right)$;
    enddo
  enddo

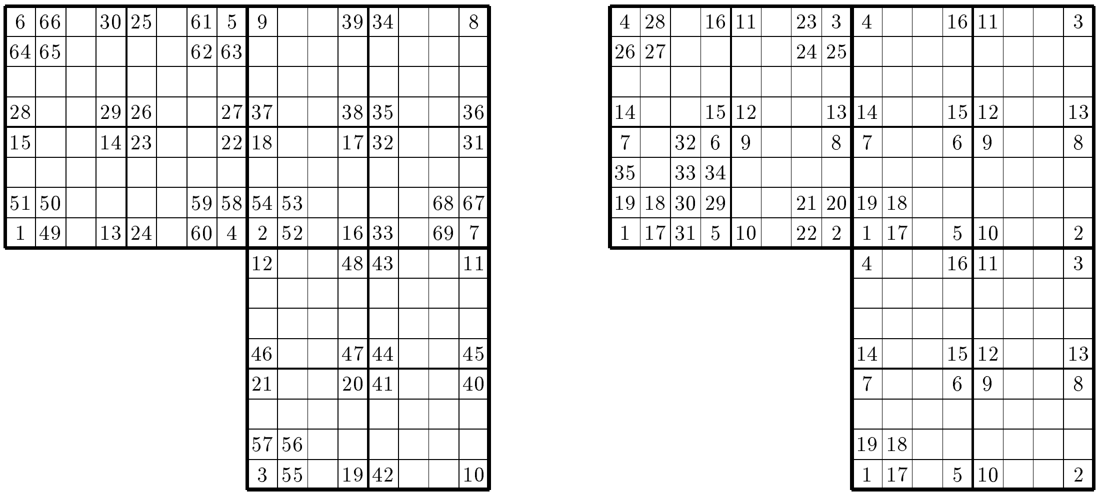
(c) **for** all $k \in \Theta(i)$ **do**
    $y_k^{\text{new}} := y_k^{\text{old}} + \omega_S\,(y_k^{M_i} - y_k^{\text{old}})$;
  enddo

enddo parallel

In contrast with the sequential smoother there is no spreading of new updated information over the macroelement boundaries within one iteration of the parallel version. This difference in general will affect the convergence rate of the parallel multigrid algorithm compared with the sequential one. We investigate this effect by means of numerical experiments in Section 6. Figure 3 shows for the sequential and the parallel smoother the order for passing through the elements in the case of 3 macroelements and grid level $l{=}4$. We use a multilevel order where we start with the elements located at the vertices of the level-1-elements. Then we take the elements at the vertices of the level-2-elements (in counterclockwise sense) followed by those at the vertices of the level-3-elements and so on. In some numerical tests, where we have compared different ordering strategies (red-black-like, zebra-like, row-wise within each macroelement), we obtained the best convergence results for the multilevel order. However, the differences in the convergence rates for the various ordering strategies have been relatively small.

5 A Jacobi Type Smoother

For the purpose of a comparison we consider a further multigrid method with a blockwise Jacobi type smoother. This smoother is defined as follows.



sequential version *parallel version*

Figure 3: *Multilevel order for passing through the elements within the smoother*

Jacobi Type Smoothing Procedure: $y^{\text{old}} \rightarrow y^{\text{new}}$

(a) **for** all elements $K \subset \Omega$ **do**
  solve the local system $\tilde{L}_K y_K = \tilde{F}_K$ generated with y^{old};
  save u_K; update $y_K \rightarrow y$;
enddo

(b) { update u at inner nodes }
for all inner u-nodes $x_m \in \Omega$ **do**

$$u_m := \tfrac{1}{2}\left(u_{K_1}(m) + u_{K_2}(m)\right) ;$$

{ where K_1, K_2 denote the two elements with $x_m \in \partial K_1 \cap \partial K_2$
and $u_{K_i}(m)$, i=1,2, the velocity values in the local block vectors u_{K_i}
corresponding to the node x_m }
enddo

(c) $y^{\text{new}} := y^{\text{old}} + \omega_S(y - y^{\text{old}})$;

In the above defined smoother there is no spreading of new updated information from one element to another within one iteration step. The sequential version of the corresponding Jacobi type multigrid method and the parallelized one (according to the macroelement decomposition) would be algorithmically equivalent. Thus, we would expect that the convergence rate of this multigrid method is an upper bound for the convergence rates of our sequential and parallel multigrid algorithm for any macroelement decomposition of the domain, i.e. also for a large number of subdomains and processors, respectively.

6 Numerical Results

For our investigations we choose as a test model the well-known driven cavity problem with $\Omega = (0,1)^2$, $f = 0$, $g(x,y)_{|\partial\Omega} = 1$ for $y = 1$ and $g(x,y)_{|\partial\Omega} = 0$ otherwise. We consider uniform $N \times N$-decompositions of Ω into macroelements (see Figure 4) with $N \in \{2,4,5,6,8,10,12,20,24\}$, i.e. the number of macroelements $N^{mac} = N^2$ ranges be-

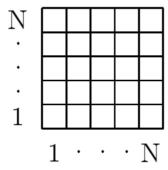


Figure 4: *Macroelement decomposition for the driven cavity problem*

tween 4 and 576. The numerical multigrid convergence rate ρ which we will compare for our different multigrid methods is defined as follows. Let y^i be the approximation of the solution y of (5) on the finest grid level after the i-th nonlinear iteration step and let $y^{i,0}$ and y^{i,m_i} be the approximations of y^i at the beginning and at the end of the linear multigrid iteration within the i-th nonlinear step. Then we define ρ by

$$\rho = \frac{1}{n}\sum_{i=1}^{n} \rho_i \qquad \text{with} \qquad \rho_i = \left(\frac{||L(y^{i-1})\, y^{i,m_i} - F||}{||L(y^{i-1})\, y^{i,0} \quad - F||} \right)^{1/m_i} \tag{11}$$

where $|| \cdot ||$ denotes the Euclidean norm and n the number of the nonlinear iteration steps. Figure 5 shows for the fixed finest grid level l=4 and the different Reynolds numbers Re=20,200,2000 the behaviour of the numerical convergence rates ρ of the sequential, parallel and Jacobi type multigrid algorithm for an increasing number N^{mac} of macroelements. Figure 6 shows the same for the fixed Reynolds number Re=2000 and the different finest grid levels l=4,5,6. Because of memory restrictions of our computer the highest possible value for N^{mac} is decreasing for an increasing finest grid level l. We see that our sequential multigrid solver works quite robust and efficient for the various Reynolds numbers and grid levels. The differences between the numerical convergence rates of the parallel and sequential version are in general very small. Only for Re=2000 and moderate values of N^{mac} they are somewhat larger. However, the differences decrease if the number N^{mac} of macroelements is increasing which is important for massive parallel applications. The differences to the convergence rate of the Jacobi type multigrid method are essentially larger especially for the high Reynolds number case. However, they decrease too for an increasing value of N^{mac} which is an interesting effect.

In the following we want to present the results for the numerical efficiency E^{num} of our parallel method compared with the sequential one. The definition (3) of E^{num} contains a problem from the practical point of view. On the one hand the CPU times T_1^{seq} and T_1^{par} have to be measured on a processor with the same performance as one node of the parallel computer. On the other hand such a processor in general has not enough memory to compute the very large global problems. Therefore, it is necessary to measure corresponding CPU times $\tilde{T}_1^{seq}$ and $\tilde{T}_1^{par}$ on a powerful serial computer with enough memory and to take $\tilde{E}^{num} = \tilde{T}_1^{seq}/\tilde{T}_1^{par}$ instead of E^{num}. We have implemented our multigrid methods on a Cray Y-MP2E/132. Since we have not used any special features of the Cray compiler we suppose that $\tilde{E}^{num}$ gives a relatively good approximation for E^{num}. Figure 7 shows for the fixed finest grid level l=4 with different Reynolds numbers and for the fixed Reynolds number Re=2000 with different finest grid levels the behaviour of the numerical efficiency E^{num} for an increasing number N^{mac} of macroelements. We obtain a good numerical efficiency for the various Reynolds numbers and grid levels

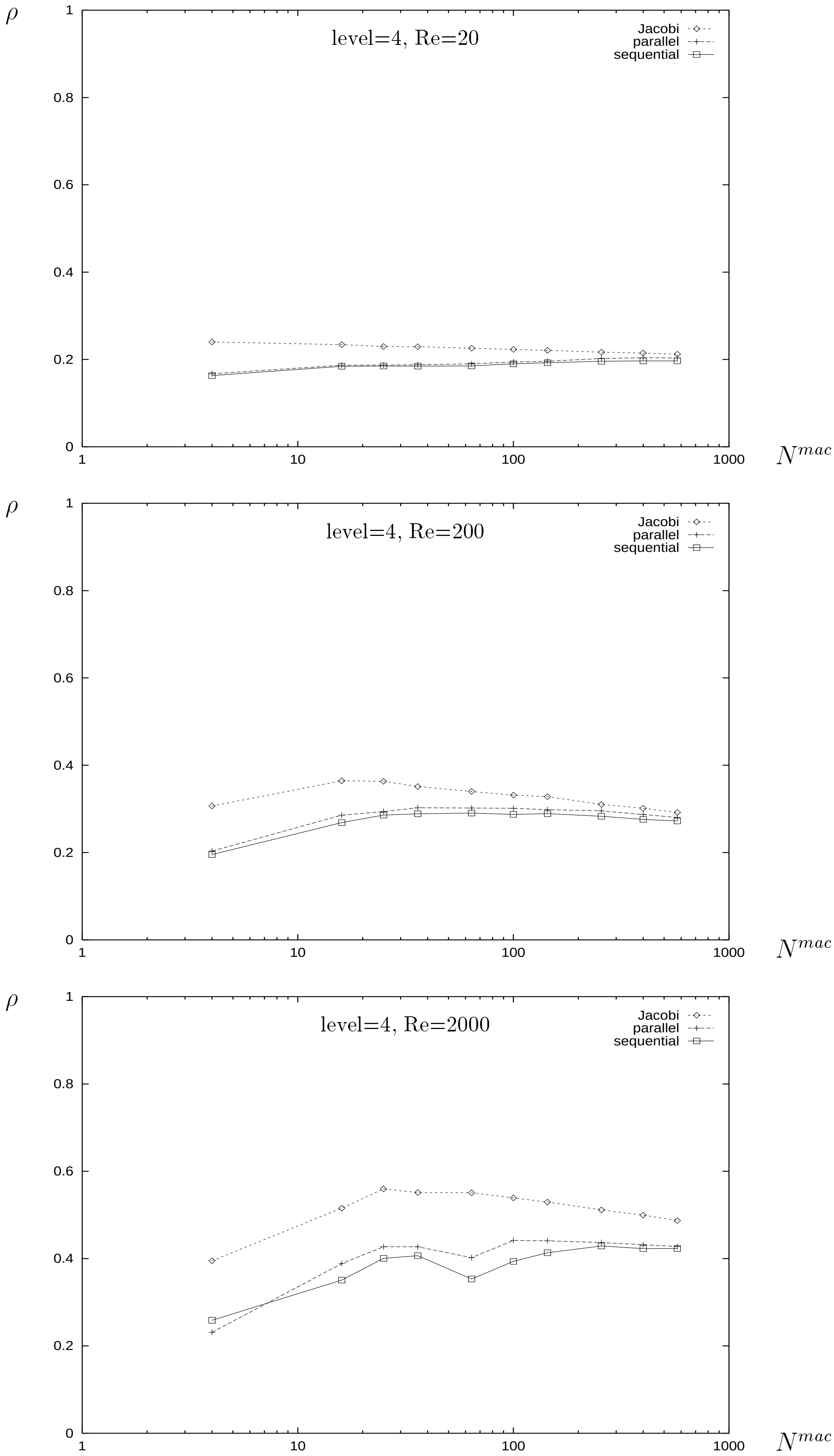


Figure 5: *Convergence rates for different Reynolds numbers*

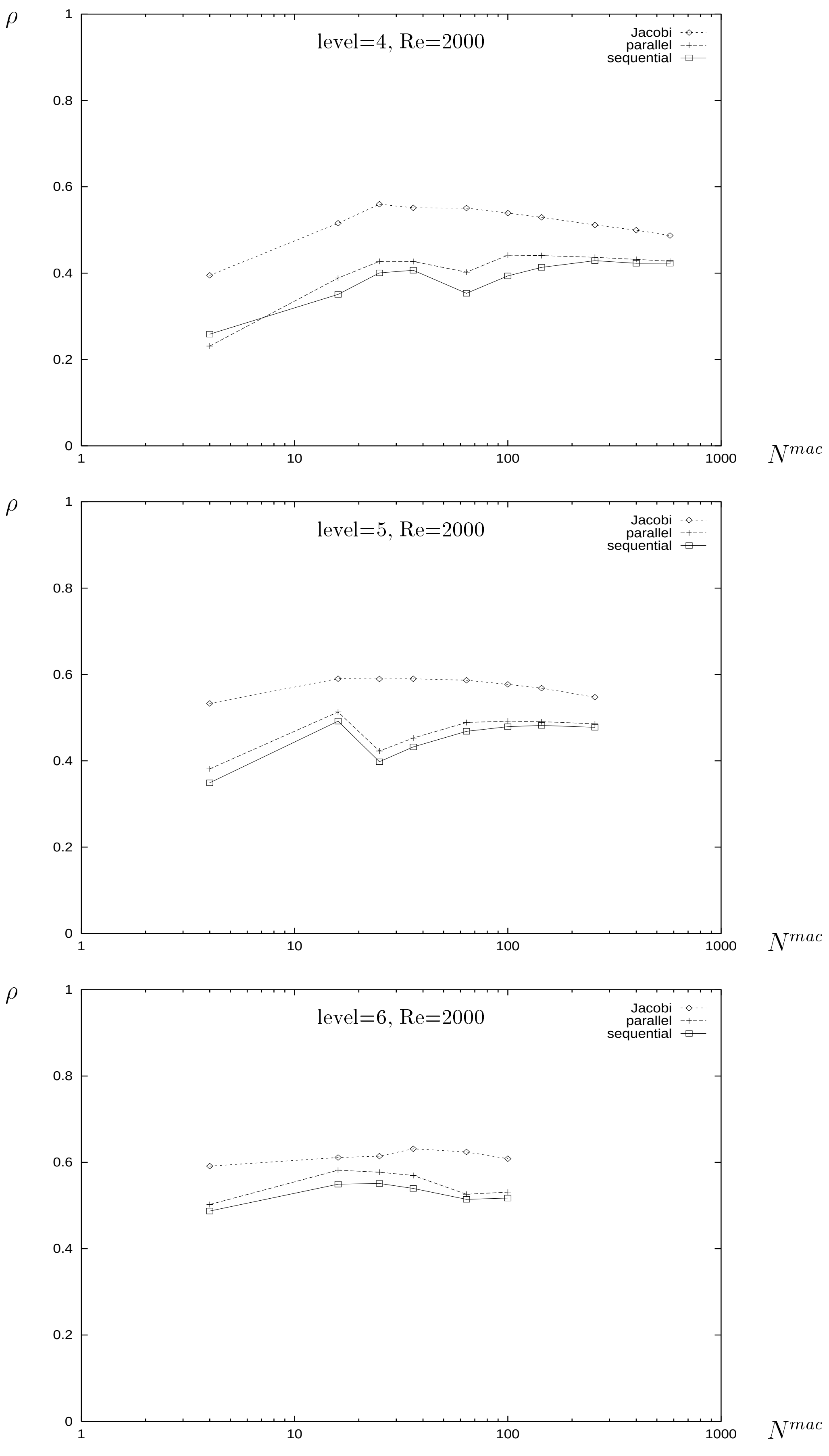


Figure 6: *Convergence rates for different grid levels*

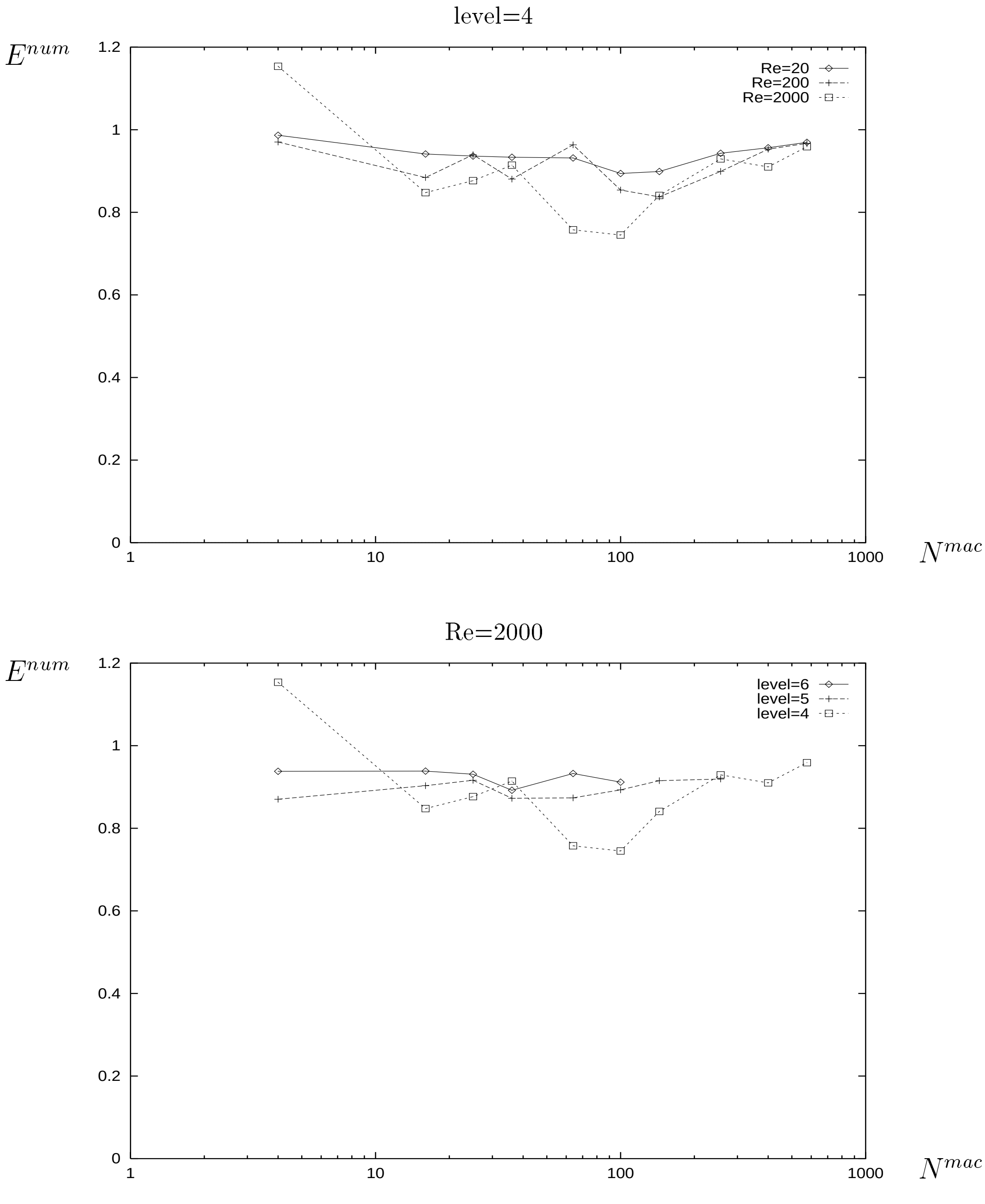


Figure 7: *Numerical efficiencies for different Reynolds numbers and grid levels*

which is a consequence of the small differences in the convergence rates of the parallel and sequential multigrid algorithm. The most important observation is that the numerical efficiency does not become small for an increasing number N^{mac} of subdomains which has been a main question of this paper.

7 Conclusions

The numerical tests indicate that for various Reynolds numbers, grid levels and macroelement decompositions of the domain the algorithmical difference between the sequential and the parallel multigrid method causes only a small difference in the multigrid convergence rates. As a consequence the parallel algorithm has a reasonable numerical efficiency even for a large number of subdomains and processors, respectively. That means, from the point of the numerical efficiency our parallel multigrid method could be a candidate for a suitable parallel Navier–Stokes solver on massive parallel systems and it has to be checked by further investigations whether it is possible to achieve also a reasonable efficiency of the parallel implementation of this solver for a large number of processors.

References

[1] M. Fortin, R. Glowinski, *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary Value Problems.* North Holland, Amsterdam, New York, Oxford (1983).

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

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

[4] 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).

[5] S. Turek, *Ein robustes und effizientes Mehrgitterverfahren zur Lösung der instationären, inkompressiblen, 2D Navier–Stokes–Gleichungen mit diskret divergenzfreien finiten Elementen.* Thesis, Preprint Nr. 642, SFB 123, Universität Heidelberg (1991).

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