A Fast Solver for Convection Diffusion Equations Based on Nested Dissection with Incomplete Elimination

Michael Bader and Christoph Zenger

TU München, Lehrstuhl für Informatik V,
D-80290 München, Germany
{bader,zenger}@in.tum.de
http://www5.in.tum.de/

Abstract. We present an approach for the efficient parallel solution of convection diffusion equations. Based on iterative nested dissection techniques [1] we extended these existing iterative algorithms to a solver based on nested dissection with incomplete elimination of the unknowns. Our elimination strategy is derived from physical properties of the convection diffusion equation, but is independent of the actual discretized operator. The resulting algorithm has a memory requirement that grows linearly with the number of unknowns. This also holds for the computational cost of the setup of the nested dissection structure and the individual relaxation cycles. We present numerical examples that indicate that the number of iterations needed to solve a convection diffusion equation grows only slightly with the number of unknowns, but is widely independent of the type and strength of the convection field.

1 Introduction

We are searching for an efficient parallel solver for the linear systems of equations arising from the discretization of the convection diffusion equation

$$-\triangle u + a(x,y) \cdot \nabla u = f. \tag{1}$$

In this paper, we focus on standard finite difference or finite element discretizations on rectangular Cartesian grids resulting in the standard five or nine point discretization stencils.

To obtain a truly efficient solver for convection diffusion equations one has to overcome three main problems. The performance of the solver should be independent of the convection field $a(x,y)$. The solver should be able to treat arbitrary geometries of the computational domain with equal efficiency. Finally, it should be possible to produce efficient parallel implementations of the solver to take optimal advantage of high performance computers.

Solvers based on geometric multigrid methods are usually quite easy to parallelize due to their structured coarse grids. On the negative side they often have difficulties in treating complicated geometries. Moreover, they often require a

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 795–805, 2000.

special treatment of curved or circular flow fields, which sometimes seems to impair the parallelization properties.
Algebraic multigrid (AMG) methods are usually very robust with respect to the strength or type of the convection or even the geometry. As AMG chooses its coarse grids according to the operator and the geometry, it often produces excellent convergence results. On the other hand, the automatic grid generation often makes it difficult to produce efficient parallel implementations of the solvers. This is especially true for the construction of the different coarse grids itself, because the most commonly used strategy [5] for the coarse grid selection is an inherently sequential process.

Our goal is therefore to find a fast (i.e. with multigrid or multigrid-like performance) method that is easy to parallelize and allows the treatment of an arbitrary geometry of the computational domain. We base our approach on previous work by Hüttl [1] and Ebner [2], whose algorithms combine ideas from domain decomposition and nested dissection techniques to iterative methods based on recursive substructuring of the computational domain.

After giving a short comprehension of these techniques in section 2, we will, in section 3, present our extension of these approaches by an incomplete elimination of the most significant couplings in the system matrices. Finally, in section 4 we will show some numerical examples that indicate the promising potential of our approach.

2 The Nested Dissection Approach

2.1 Nested Dissection as a Direct Solver

The nested dissection method was introduced by George [4] as a direct solver for the sparse linear systems of equations resulting from the discretization of PDEs with finite elements. It it based on the recursive substructuring of the computational domain by which it minimizes the fill-in that usually occurs during the solution process. Throughout this paper we will divide the nested dissection algorithm into three passes: the recursive substructuring, the static condensation, and the solution.

Pass 1: Recursive Substructuring

In the first (top-down) pass the computational domain is recursively divided into two or more subdomains that are connected via a *separator*. As shown in figure 1, we will use a separation by alternate bisection throughout this paper. Compared to substructuring in four (or even more) subdomains the alternate bisection produces linear systems of equations with a slightly smaller number of unknowns, which gives advantages in parallelization.

On each subdomain, we classify the unknowns into the set $\mathcal{I}$ of the inner unknowns and the set $\mathcal{E}$ of the outer unknowns. The inner unknowns are those unknowns on the separator that do not belong to the separator of a parent

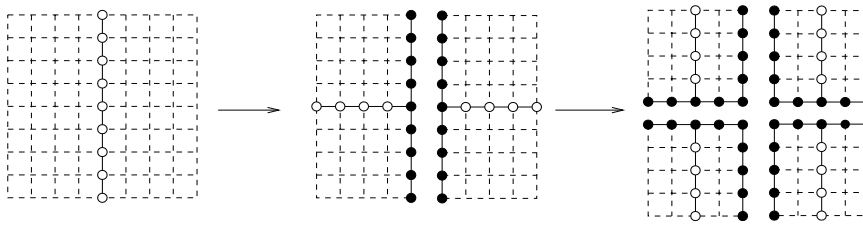


Fig. 1. Recursive substructuring by alternate bisection

domain. The outer unknowns lie on the border of the subdomain and are exactly those unknowns that will become separator unknowns on a parent domain. The other unknowns inside the subdomain can be ignored, as their couplings with inner and outer unknowns are eliminated by the static condensation (see pass 2) on the lower levels. Figure 1 illustrates this classification by painting the inner unknowns as white circles and the outer unknowns as black circles.

Pass 2: Static Condensation

The static condensation pass is a bottom-up process and computes the local systems of equations for each subdomain. On the finest level — i.e. when the subdomain has only 2×2 or 3×3 unknowns and is not further divided — the system of equations can be taken directly from the discretization. On the higher levels the system is computed from the local systems of the two subdomains. The first step is to assemble and renumber the system matrix:

$$V^T \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix} V =: \begin{pmatrix} K_{\mathcal{E}\mathcal{E}} & K_{\mathcal{E}\mathcal{I}} \\ K_{\mathcal{I}\mathcal{E}} & K_{\mathcal{I}\mathcal{I}} \end{pmatrix}. \tag{2}$$

The operator V combines the system matrices K_1 and K_2 that are taken from the two subdomains. The unknowns on the separator belong to both subdomains, thus the corresponding matrix lines are simply added up. The renumbering is such that the outer and inner unknowns form separate matrix blocks $K_{\mathcal{E}\mathcal{E}}$ and $K_{\mathcal{I}\mathcal{I}}$ to enable the following block elimination.

In this second step the couplings between the inner and the outer unknowns are eliminated by computing the Schur complement

$$\begin{pmatrix} \mathrm{Id} & -K_{\mathcal{E}\mathcal{I}}K_{\mathcal{I}\mathcal{I}}^{-1} \\ 0 & \mathrm{Id} \end{pmatrix} \begin{pmatrix} K_{\mathcal{E}\mathcal{E}} & K_{\mathcal{E}\mathcal{I}} \\ K_{\mathcal{I}\mathcal{E}} & K_{\mathcal{I}\mathcal{I}} \end{pmatrix} \begin{pmatrix} \mathrm{Id} & 0 \\ -K_{\mathcal{I}\mathcal{I}}^{-1}K_{\mathcal{I}\mathcal{E}} & \mathrm{Id} \end{pmatrix} = \begin{pmatrix} \widetilde{K}_{\mathcal{E}\mathcal{E}} & 0 \\ 0 & K_{\mathcal{I}\mathcal{I}} \end{pmatrix} \tag{3}$$

$$\text{where} \quad \widetilde{K}_{\mathcal{E}\mathcal{E}} = K_{\mathcal{E}\mathcal{E}} - K_{\mathcal{E}\mathcal{I}} \cdot K_{\mathcal{I}\mathcal{I}}^{-1} \cdot K_{\mathcal{I}\mathcal{E}}.$$

Of course, the right-hand sides have to be treated accordingly.

Pass 3: Solution

The solution itself is again a top-down process. Starting with the separator of the whole computational domain, the values of the separator unknowns are computed recursively from the local system of equations on each subdomain.

2.2 Iterative Versions of the Nested Dissection Method

In the case of a two-dimensional PDE using standard discretization with a five- or nine-point stencil, computing the direct solution of the resulting linear system of equations (with $N = n \times n$ unknowns) needs $O(N^{\frac{3}{2}})$ operations and requires $O(N \log N)$ memory [4]. While one can sometimes put up with the $O(N^{\frac{3}{2}})$ operation count and the resulting increased computing time, shortness of memory often inhibits simulations altogether as the required resolution exceeds the memory capacity. However, the nested dissection method is often used in industrial codes for reason of reliability and robustness. For Poisson type equations, Hüttl [1] and Ebner [2] introduced iterative versions of the nested dissection method that have a memory requirement that grows only linearly with the number of unknowns.

The main differences between an iterative and a direct solver occur in the condensation and the solution pass. While the assembly part of the condensation pass mainly stays the same, the elimination of the separator couplings is left out entirely or at least greatly reduced in an iterative solver. The single top-down solution pass is replaced by a series of iteration cycles. Each of those cycles consists of a bottom-up pass that transports the current residual, like the right-hand side in the condensation pass, from the finest level subdomains to the higher levels. The second part of the iteration cycle is the actual solution, where Gauss-Seidel- or Jacobi-relaxation is used for the computation of the separator unknowns. Figure 2 illustrates the complete sequence of the different passes during the iterative nested dissection.

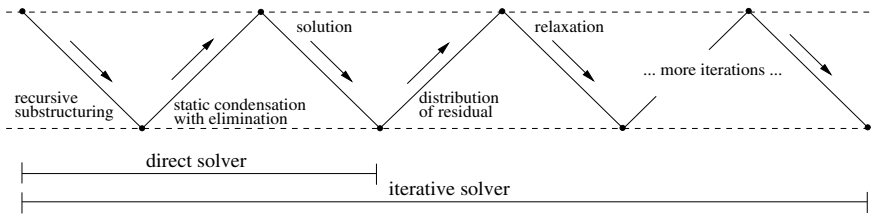


Fig. 2. The different passes of the iterative nested dissection algorithm

To get an acceptable speed of convergence, the systems of equations are preconditioned, for example by means of hierarchical bases [6]. The resulting algorithms reduce the number of operations to $O(N \log N)$ and the memory requirements back to $O(N)$. The convergence factor of such an iterative nested dissection method can be further improved by eliminating at least some of the couplings between the hierarchically highest unknowns in the local systems of equations [2]. Figure 3 shows the general structure of the resulting algorithm, which will be the basis for the algorithm introduced in section 3.

Pass 2: static condensation (set-up phase)

(S1) $K' = V^T \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix} V$	assemble system matrix from subdomains
(S2) $\bar{K} = H^T K' H$	hierarchical transformation
(S3) $K = L^{-1} \bar{K} R^{-1}$	partial elimination of the separator couplings, the elimination matrices L and R have to be stored

Pass 3: iteration cycle (bottom-up part)

(U1) $r' = V^T \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$	assemble local residual from subdomains
(U2) $\bar{r} = H^T r'$	hierarchical transformation
(U3) $r = L^{-1} \bar{r}$	right-hand side of elimination

Pass 3: iteration cycle (top-down part)

(D1) $\hat{u}_S^{[it]} = \omega \mathrm{diag}(K)^{-1} r$	relaxation step (here: weighted Jacobi)
(D2) $\bar{u} = R^{-1} \hat{u}^{[it]}$	revert elimination
(D3) $u = H \bar{u}$	hierarchical transformation
(D4) $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = V^T u$	distribute solution to subdomains

Fig. 3. Iterative Nested Dissection Algorithm: steps (S3), (U3) and (D2) are only needed if partial elimination is used

2.3 Parallel, Iterative Nested Dissection

The parallelization of iterative nested dissection algorithms like the one described in figure 3 was analysed in depth by Hüttl [1] and Ebner [3]. Figure 4 shows a typical distribution of the subdomains to different processors. Obviously, the tree structure of the subdomains and the simple bottom-up/top-down structure of the several computational cycles enable the parallel processing of all subdomains that are on the same level. However, the different levels have to be executed sequentially.

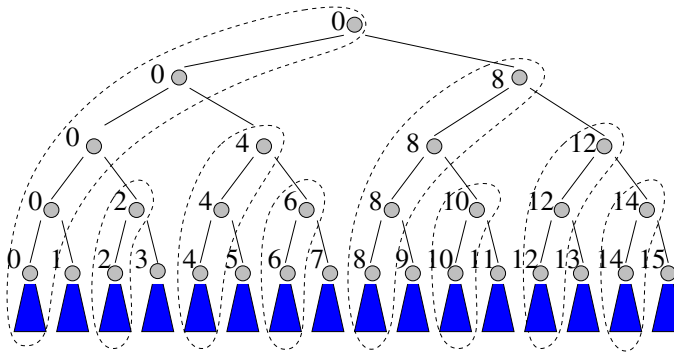


Fig. 4. Distribution of subdomains to 16 processors

The computations on each subdomain are usually not parallelized, so the size of these sequential subproblems, which is mainly determined by the size of the separator, should be kept small. We already mentioned that, in this context, the alternate bisection used in the recursive substructuring pass gives some advantage over other substructuring approaches. When using Jacobi relaxation for solution, the memory requirement on each subdomain is only linearly dependent on the number of unknowns on the separator. Therefore, these methods are particularly well suited for architectures with distributed memory.

Compared to other types of solvers (geometric multigrid, AMG, etc.), one should also emphasize that all three passes of the algorithm have equally good parallelization properties.

3 Nested Dissection with Incomplete Elimination

While the algorithms discussed in the previous section give good results for the solution of diffusion type equations, this no longer holds when they are applied to convection diffusion equations. In this section, we want to propose an approach which extends the existing iterative algorithms, such that convection diffusion equations can be treated efficiently as well.

When solving simple diffusion type equations, the elimination of the couplings between inner and outer unknowns can well be left out entirely, yet the convergence rates can be improved by eliminating just a small, fixed number of only the strongest couplings. For example, it is sufficient to eliminate the couplings between the unknown in the middle of the separator and those four unknowns that lie on the corners of the subdomain. However, for the convection diffusion case, eliminating a fixed number of couplings on each subdomain is no longer sufficient to achieve fast convergence.

In general, we can expect that the more couplings we eliminate the better the convergence rates will become, because finally, when all couplings between inner and outer unknowns are eliminated, the algorithm becomes a direct nested dissection solver again, and "converges" in one step. Of course we pay for this rapid convergence with the $O(N^{\frac{3}{2}})$ computing time for the eliminations and $O(N \log N)$ memory requirement due to the generated fill-in.

We therefore have to look for a compromise between eliminating enough couplings to achieve good convergence rates and keeping the number of eliminations small enough to maintain the $O(N)$ complexity with respect to computing time and memory requirement. A suitable heuristics for choosing the "correct" number of eliminated[1] unknowns can be found by analysing the underlying physical effects. Figure 5 shows a certain subdomain where a heat source on the left border is transported via a constant convection field towards the domain separator. Due to diffusion the former peak will extend over several mesh size steps after it has been transported to the separator. It is clear that using only one point on the separator is not enough to resolve the resulting heat profile (left picture).

[1] to "eliminate" an unknown in this context means that all couplings between "eliminated" unknowns are eliminated.

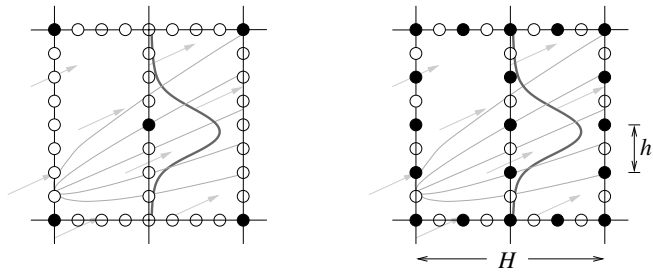


Fig. 5. Comparison of different elimination strategies: only the couplings between the black nodes are eliminated

Therefore, eliminating only the couplings between the black nodes of the left picture would not be sufficient to achieve good convergence. However, as we can see in the right picture, it is not necessary to eliminate all separator nodes like it is done in nested dissection.

The question is now how to choose the distance h between two eliminated nodes. From the underlying physics it is known that, for convection diffusion equations, the streamlines of the transported heat have a parabolic profile. Therefore, if the typical distance H between the unknowns grows by a factor of four, we can expect the heat profile to become twice as wide. Thus we can also double the distance h between the eliminated nodes. In other words, we choose $h \propto \sqrt{H}$ and the number of eliminated separator unknowns proportional to the square root of the total number of unknowns on the separator. In our algorithm, we implement this square root dependency by doubling the number of eliminated separator unknowns after every four bisection steps. This strategy is illustrated in figure 6.

The overall algorithm remains the same as in figure 3, but now the elimination matrices L and R no longer have only a small, fixed number of non-diagonal entries. If we have a subdomain with m separator unknowns, we eliminate the couplings between $O(\sqrt{m})$ inner unknowns and $O(\sqrt{m})$ outer unknowns which produces $O(m)$ entries in L and R. As a result of these extra eliminations, the

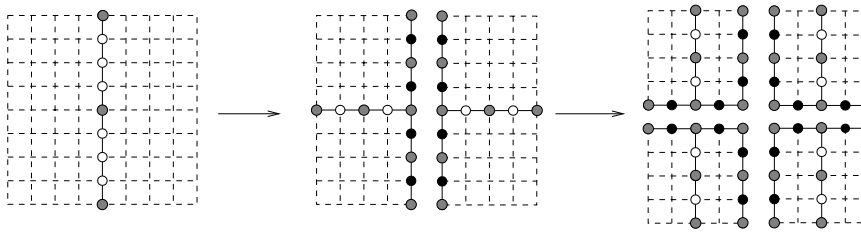


Fig. 6. Bisection with incomplete elimination, only the couplings between the grey nodes are eliminated

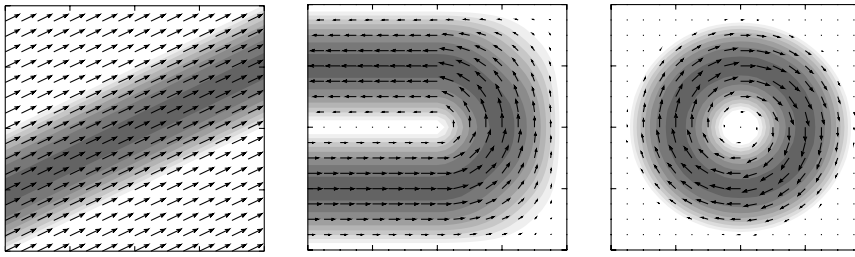
fill-in created in the system matrix K is now much bigger compared to the algorithms of section 2.2. Therefore, one can no longer afford to store the entire system matrix K for each subdomain as this would result in an $O(N \log N)$ memory requirement. However, if we choose the (weighted) Jacobi-method for relaxation, we only have to store the main diagonal of K as we can compute the residual separately (see steps U1-U3 of the algorithm in figure 3). This puts both the memory requirement and the number of operations needed for the set up of the system matrices back to $O(N)$. Of course the Jacobi-relaxation gives less satisfying convergence factors than the Gauss-Seidel-relaxation, but, as we will show in the next section, it is still possible to achieve reasonable performance.

4 Numerical Results

We tested our algorithm on the three different benchmark problems outlined in figure 7. Problem (a) indicates a flow with constant convection $a(x, y) = a$, problem (b) an entering flow that contains a 180 degree curve of the flow, and finally problem (c) a circular flow problem. Each problem was discretized on a square using standard second order finite difference discretization with a five point stencil.

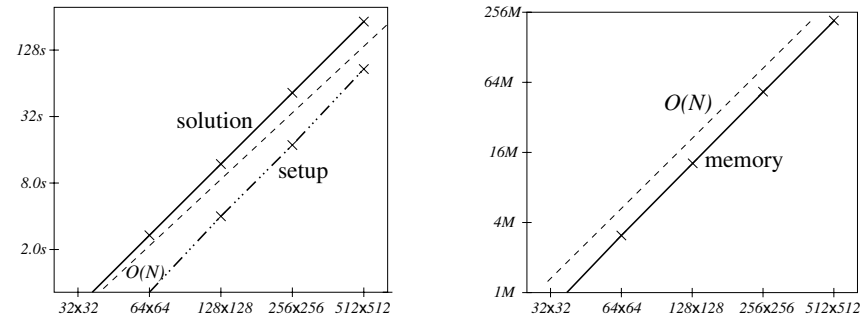
As the elimination points are chosen independently of the flow field it is clear that the computation time and the memory used are independent of the type of the test problem. Figure 8 indicates that both computation time and memory requirement rise linearly with the number of unknowns. This can also be deduced by theoretical means.

Table 1 shows the convergence rates for test problem (c), but nearly identical results are obtained for test problems (a) and (b). Table 2 shows the average convergence rates for test problem (c) if the algorithm is used as a preconditioner for the BiCGStab method [7]. For the Poisson equation it is known that preconditioning with hierarchical bases leads to a logarithmic increase of the convergence rates, which corresponds well with the behaviour we observe in tables 1 and 2. We can see that the convergence rates are largely independent on the strength of convection as long as a certain ratio between convection strength and mesh



(a) straight convection (b) bent pipe convection (c) circular convection

Fig. 7. Convection fields of the three test problems



Costs for setup and complete solution Memory requirement

Fig. 8. Computation time and memory requirement

size is not exceeded. For stronger convection the heuristics of our elimination strategy no longer holds and convergence begins to break down.

	$a=0$	$a=1$	$a=2$	$a=4$	$a=8$	$a=16$	$a=32$	$a=64$	$a=128$	$a=256$
32×32	0.635	0.635	0.636	0.637	0.644	0.661	0.743	—	—	—
64×64	0.700	0.700	0.700	0.699	0.697	0.693	0.694	0.753	—	—
128×128	0.721	0.722	0.722	0.722	0.724	0.734	0.756	0.823	0.853	—
256×256	0.746	0.746	0.746	0.746	0.746	0.746	0.744	0.757	0.818	0.848
512×512	0.783	0.783	0.783	0.783	0.783	0.783	0.784	0.795	0.859	div

Table 1. Convergence rates for test problem c (circular convection), a denotes the strength of the convection. (Jacobi-relaxation with $\omega = \frac{2}{3}$)

	$a=0$	$a=1$	$a=2$	$a=4$	$a=8$	$a=16$	$a=32$	$a=64$	$a=128$	$a=256$
32×32	0.079	0.083	0.085	0.084	0.083	0.127	0.147	0.210	0.315	0.467
64×64	0.151	0.153	0.154	0.155	0.156	0.153	0.161	0.227	0.401	0.404
128×128	0.156	0.156	0.157	0.157	0.159	0.175	0.216	0.351	0.433	0.545
256×256	0.208	0.208	0.207	0.205	0.197	0.214	0.221	0.238	0.357	0.462
512×512	0.226	0.226	0.226	0.227	0.227	0.229	0.231	0.271	0.376	0.463

Table 2. Average convergence rates for test problem c (circular convection) using preconditioned BiCGStab and upwind discretization

A parallel implementation of our iterative nested dissection code was realized using MPI as message passing protocol. The distribution of the subdomains to the processors was done as was shown in figure 4. Figure 9 illustrates speedup and parallel efficiency of the parallel implementation for a problem with 512×512 unknowns on a cluster of SUN Ultra 60 workstations.

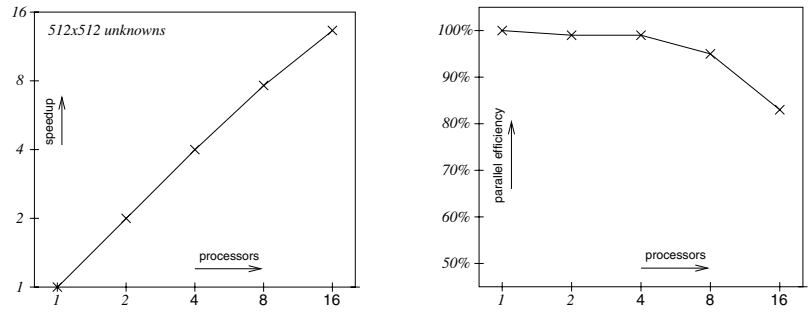


Fig. 9. Speedup and parallel efficiency on a cluster of SUN Ultra 60 Workstations

5 Present and Future Work

We are currently working on two topics which are still missing in the presented algorithm to become a suitable solver for practical convection type problems. The first topic is the treatment of complicated computational domains and of inner boundaries and obstacles, the other one is the removal of the $O(\log n)$ dependency in the convergence rates and to make them independent of the resolution of the mesh.

It seems that both topics can be successfully treated if we replace the hierarchical basis preconditioning by the usage of generating systems which turns the algorithm into a multigrid type method (see Griebel [8]). First numerical experiments indicate that the convergence rates seem to indeed become independent of mesh size, geometry and strength of convection (as long as the diffusion still dominates the convection on the finest mesh).

The efficient treatment of strongly convection dominated flow leads our list of future work. In the case of strong convection our elimination strategy is no longer appropriate as it depends on the existence of a certain amount of real diffusion. However, an efficient elimination strategy should be achievable either by eliminating couplings that are aligned with the direction of the flow or by choosing the eliminated couplings in an algebraic manner which leads to an AMG like method with fixed coarse grid selection.

References

1. Hüttl, R., Schneider, M.: Parallel Adaptive Numerical Simulation. Institut für Informatik, TU München, SFB-Bericht 342/01/94 A (1994)
2. Ebner, R.: Funktionale Programmierkonzepte für die verteilte numerische Simulation. PhD thesis, TU München (1999)
3. Ebner, R., Zenger, C.: A distributed functional framework for recursive finite element simulation. Parallel Computing **25** (1999) 813-826
4. George, A.: Nested Dissection of a Regular Finite Element Mesh. SIAM Journal on Numerical Analysis **10** (1973)

5. Ruge, J.W., Stüben, K.: Algebraic multigrid. In: McCormick, S.F. (ed.): Multigrid Methods. SIAM (1987) 73-130
6. Yserentant, H.: Hierarchical basis give conjugate gradient methods a multigrid type speed of convergence. Appl. Math. and Comput. **19** (1986) 347-358
7. van der Vorst, H.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Statist. Comput., **13** (1992) 631-644
8. Griebel, M.: Multilevel algorithms considered as iterative methods on semidefinite systems. SIAM Int. J. Sci. Stat. Comput., **15/3** (1994) 547-565