A robust and parallel Multigrid Method for Convection Diffusion Equations

Michael Bader and Christoph Zenger
Lehrstuhl für Informatik V der TU München, 80290 München, Germany
bader@in.tum.de

We present a multigrid method for the solution of convection diffusion equations that is based on the combination of recursive substructuring techniques and the discretization on hierarchical bases and generating systems. Robustness of the resulting method, at least for a variety of benchmark problems, is achieved by a partial elimination between certain "coarse grid unknowns".

The choice of these coarse grid unknowns is motivated by the 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 and the individual relaxation cycles. We present numerical examples that indicate that the number of iterations needed to solve a convection diffusion equation is widely independent of the number of unknowns and of the type and strength of the convection field.

1 Introduction

To obtain a truly efficient solver for the linear systems of equations arising from the discretization of convection diffusion equations

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

one has to overcome three major difficulties. The solver should be efficient regardless of the strength and geometry of the convection field $a(x,y)$. It should be able to treat computational domains of arbitrary geometry. And it should be easy to parallelize to take optimal advantage of high performance computers. Up to now there doesn't seem to be a solver that meets all these requirements equally well, at least not in the general 3D case:

Solvers based on geometric multigrid methods are usually quite easy to parallelize due to their structured coarse grids. However, the need for robustness poses quite severe demands on the applied smoothers, which again can impair the parallel efficiency of the method. Moreover, the treatment of complicated geometries is not always easy, especially on the coarse grids.

Algebraic multigrid (AMG) methods are usually robust even for strongly convection dominated flows and flows on very complicated geometries. The key to their excellent convergence results lies mainly in the coarse grid generation which is automatically adapted to the special requirements of the geometry and the operator. On the other hand, this automatic grid generation often makes an efficient parallel implementation a tedious task. Moreover, the most commonly used strategy [7] for the coarse grid selection is an inherently sequential process, such that the construction of the different coarse grids itself has to be modified.

In this paper we will look into an approach which combines ideas from recursive substructuring techniques with the discretisation on hierarchical bases and generating systems. The resulting multilevel method is extended by an additional partial elimination between certain *coarse grid unknowns*. These coarse grid unknowns are chosen with respect to the underlying physics of the convection diffusion equation. The overall approach can roughly be regarded as an algebraic multigrid method in which the coarse grid selection is fixed and only the coarse grid operators are constructed from the actual discretisation.

After giving a short comprehension of the recursive substructuring method in section 2, we will, in section 3, describe the extension of this approach to using an additional elimination of the most significant couplings in the local system matrices. Finally, in section 4, we will examine the potential of our approach on some typical benchmark problems.

2 Recursive Substructuring

2.1 A Direct Solver Based on Recursive Substructuring

In this section we will describe a direct solver based on recursive substructuring which is very similar to the well-known nested dissection method introduced by George [4]. The later described iterative variant will differ from the direct solver only in the incomplete elimination and the modified solution cycle. Both variants of the recursive substructuring algorithm can be divided into three passes: the actual substructuring, the static condensation, and the solution.

Recursive Substructuring

In a first (top-down) pass the computational domain is recursively divided into two subdomains (*alternate bisection*, see figure 1) that are connected by a *separator*. On each subdomain, we define the set $\mathcal{E}$ of outer unknowns (which lie on the subdomain boundary) and the set $\mathcal{I}$ of inner unknowns, that lie on the separator but do not belong to the separator of a parent domain. Figure 1 illustrates this classification by painting the inner unknowns as grey circles and the outer unknowns as black circles. The remaining unknowns inside the subdomain can be ignored, as their influence on the unknowns in $\mathcal{E}$ and $\mathcal{I}$ will be eliminated by the static condensation pass, which will be described in the following.

Static Condensation

The static condensation pass computes a local system of equations for each subdomain. For the smallest subdomains — i.e. the leaves of the substructuring tree — the system of equations is taken directly from the discretization. On the other domains the system is computed from the local systems of the two subdomains. First, the system matrix and right hand side is assembled

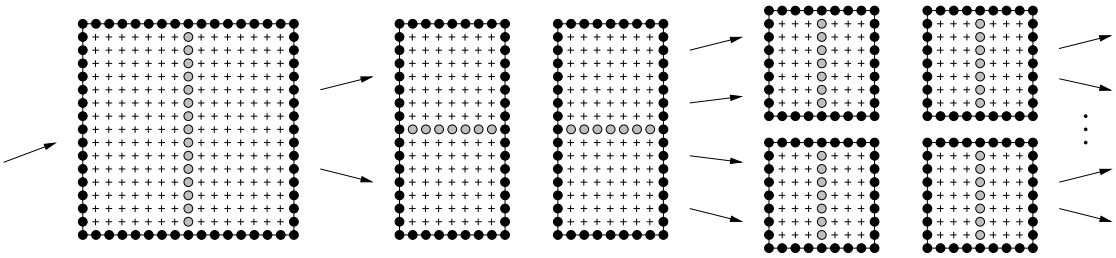


Figure 1: Recursive substructuring by alternate bisection

from those of the subdomains:

$$A := \begin{pmatrix} A_{\mathcal{E}\mathcal{E}} & A_{\mathcal{E}\mathcal{I}} \\ A_{\mathcal{I}\mathcal{E}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix} := V_{(1)}^T \left(A_{(1)}\right)_{\mathcal{E}\mathcal{E}} V_{(1)} + V_{(2)}^T \left(A_{(2)}\right)_{\mathcal{E}\mathcal{E}} V_{(2)} \qquad b := V_{(1)}^T b_{(1)} + V_{(2)}^T b_{(2)}. \quad (2)$$

Note that only the couplings between the outer unknowns of the subdomains are transferred to the parent system. The renumbering of these unknowns to their parent domain is performed by the operators $V_{(i)}$. The numbering on each subdomain separates the outer and inner unknowns to build matrix blocks ($A_{\mathcal{E}\mathcal{E}}, A_{\mathcal{I}\mathcal{I}}, \ldots$) that enable the following block elimination.

In this second step the inner and outer unknowns are decoupled,

$$\begin{pmatrix} \mathrm{Id} & -A_{\mathcal{E}\mathcal{I}}A_{\mathcal{I}\mathcal{I}}^{-1} \\ 0 & \mathrm{Id} \end{pmatrix} \begin{pmatrix} A_{\mathcal{E}\mathcal{E}} & A_{\mathcal{E}\mathcal{I}} \\ A_{\mathcal{I}\mathcal{E}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix} \begin{pmatrix} \mathrm{Id} & 0 \\ -A_{\mathcal{I}\mathcal{I}}^{-1}A_{\mathcal{I}\mathcal{E}} & \mathrm{Id} \end{pmatrix} = \begin{pmatrix} \widetilde{A}_{\mathcal{E}\mathcal{E}} & 0 \\ 0 & A_{\mathcal{I}\mathcal{I}} \end{pmatrix}, \quad (3)$$

by computing the Schur complement

$$\widetilde{A}_{\mathcal{E}\mathcal{E}} = A_{\mathcal{E}\mathcal{E}} - A_{\mathcal{E}\mathcal{I}} \cdot A_{\mathcal{I}\mathcal{I}}^{-1} \cdot A_{\mathcal{I}\mathcal{E}}. \quad (4)$$

Of course, the right-hand sides have to be treated accordingly. The matrix block $\widetilde{A}_{\mathcal{E}\mathcal{E}}$ and the according part $\widetilde{b}_{\mathcal{E}}$ of the right hand side are then transferred to the parent domain which proceeds with the setup like in equation 2.

Solution

After the setup of the local systems of equations is completed, the actual solution can be computed on the different subdomains. Starting with the separator of the entire computational domain, the values of the separator unknowns are computed recursively in a top-down process from the local systems of each subdomain.

2.2 Iterative Version of the Substructuring Method

The direct solver described in the previous section would, like the very similar *nested dissection* method [4], require $\mathcal{O}(N^{3/2})$ operations and $\mathcal{O}(N \log N)$ memory — $N = n \times n$ the number of unknowns — for both setup and solution of a 2D problem. At least for Poisson type equations, operation count and memory requirement can be reduced significantly by using an iterative recursive substructuring method that uses a hierarchical basis for preconditioning [6, 8].

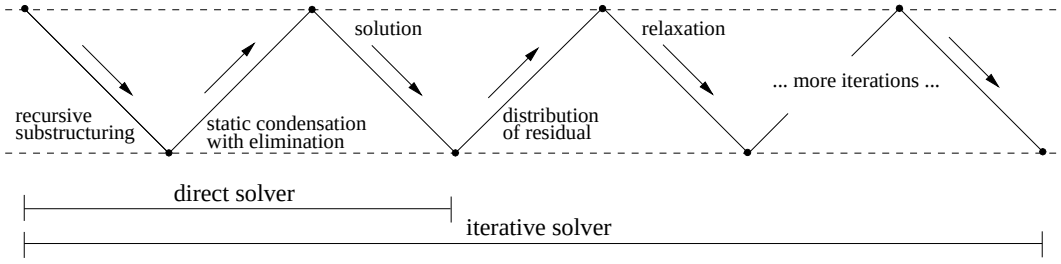


Figure 2: The different passes of the iterative substructuring algorithm

Pass 2: static condensation (setup phase)

(S1)	$A = V_{(1)}^T A_{(1)} V_{(1)} + V_{(2)}^T A_{(2)} V_{(2)}$	assemble system matrix from subdomains
(S2)	$\bar{A} = H^T A H$	hierarchical transformation
(S3)	$\widetilde{A} = L^{-1} \bar{A} R^{-1}$	partial elimination

Pass 3: iteration cycle (bottom-up part)

(U1)	$r = V_{(1)}^T r_{(1)} + V_{(2)}^T r_{(2)}$	assemble local residual from subdomains
(U2)	$\bar{r} = H^T r$	hierarchical transformation
(U3)	$\widetilde{r} = L^{-1} \bar{r}$	right-hand side of elimination

Pass 3: iteration cycle (top-down part)

(D1)	$\widetilde{u}_{\mathcal{I}} = \omega \, \mathrm{diag}(\widetilde{A}_{\mathcal{I}\mathcal{I}})^{-1} \widetilde{r}$	relaxation step (here: weighted Jacobi)
(D2)	$\bar{u} = R^{-1} \widetilde{u}$	revert elimination
(D3)	$u = H \bar{u}$	hierarchical transformation
(D4)	$u_{(1)} = V_{(1)}^T u, u_{(2)} = V_{(2)}^T u$	distribute solution to subdomains

Figure 3: Iterative substructuring algorithm: steps (S3), (U3) and (D2) are only needed if partial elimination is used

The static condensation pass is reduced to the assembly of the local system matrices and the hierarchical transformation which replaces the computation of the Schur complement. The single top-down solution pass is therefore replaced by a series of iteration cycles. Each of these cycles solves the residual equation by transporting the current residual in a bottom-up pass from the smallest subdomains to their parent and ancestor domains. On each domain a correction is then computed from the transported residual. Figure 2 illustrates the overall scheme of the iterative substructuring algorithm. The general structure of the resulting algorithm is given in figure 3. It already includes the additional partial elimination in the local systems which will be described in section 3.

Obviously, the tree structure of the subdomains and the simple bottom-up/top-down structure of the several computational cycles enable a very simple parallelization of the algorithm. The parallelization of iterative substructuring algorithms like the one described in figure 3 was analysed in depth by Hüttl [6] and Ebner [3].

3 Partial Elimination of Unknowns

For Poisson type equations the algorithm discussed in section 2.2 already shows a reasonable fast convergence that slows down only slightly with growing number of unknowns. In [2] it was demonstrated how comparable results can be achieved for convection diffusion equations. The concept used in that paper can be easily extended to the use of hierarchical generating systems instead of hierarchical bases, which turns the algorithm into a true multigrid method [5].

The key idea is to choose certain couplings — hopefully the strongest —, which should be eliminated by the partial elimination included in the algorithm in figure 3. The complete decoupling of inner and outer unknowns, like it is obtained by the computation of the Schur complement in the equations 3 and 4, is replaced by a transformation

$$\underbrace{L^{-1}AR^{-1}}_{=:\widetilde{A}}\widetilde{x} = \underbrace{L^{-1}b}_{=:\widetilde{b}}. \tag{5}$$

The elimination matrices L^{-1} and R^{-1} are chosen such that certain entries in the transformed system matrix $\widetilde{A}$ are eliminated. These eliminated couplings are not chosen individually, but instead we pick an a priori set of *coarse grid unknowns* between which the strong couplings typically occur. Then, simply all couplings between those coarse grid unknowns are eliminated.

If hierachical bases or generating systems are applied, it is natural to choose the hierarchically coarse unknowns as coarse grid unknowns. In contrast, the couplings with and between hierarchically fine unknowns are usually small as these make only a small contribution to the transport of the matter u. This is due to the fact, that the diffusion quickly smoothes out the hierarchically fine peaks and corrections in u such that a transport of u across larger distances is prohibited.

Consequently the distinction between coarse grid unknowns and fine grid unknowns is also a matter of the distance between the grid points and should therefore depend on the size of the actual subdomain. Figure 4 illustrates this by an example of a certain heat transport problem. It shows a certain subdomain where a heat source on the left border is transported by 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 the

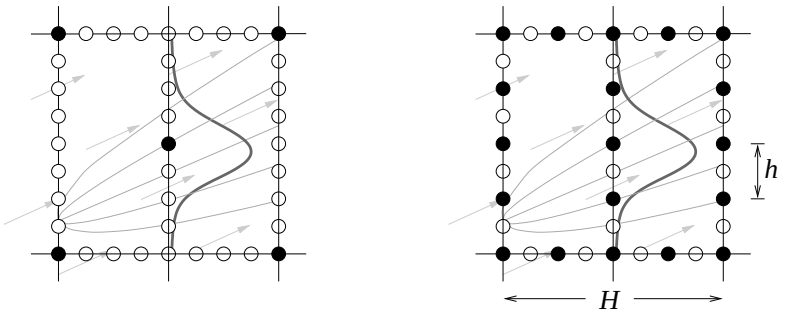


Figure 4: Comparison of different elimination strategies: only the couplings between the black nodes are eliminated

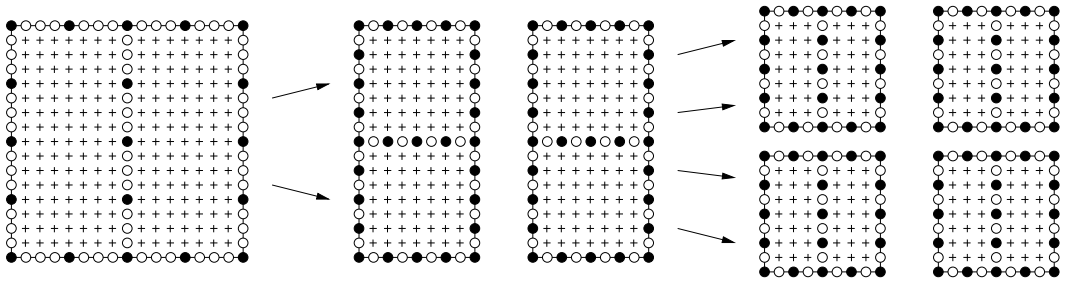


Figure 5: Bisection with incomplete elimination, only the couplings between the black nodes are eliminated

resulting temperature profile is not resolvable when using only the central point on the separator (left picture). On the other hand, it would also be unnecessary to use all the fine grid points on the separator. We therefore have to look for a good compromise for the resolution h of the coarse grid unknowns (right picture) depending on the size H of the subdomain.

From the underlying physics it is known that, for convection diffusion equations, the streamlines of the transported heat have a parabolic profile. Therefore, we should choose $h^2 \propto H$ or $h \propto \sqrt{H}$ respectively, which means that the number of number of coarse grid unknowns should grow with the square root of the number of total separator unknowns. We can implement this square root dependency by doubling the number of eliminated separator unknowns after every four bisection steps. This strategy is illustrated in figure 5.

In the algorithm of figure 3 the partial elimination is already included. A technical detail results from the fact that L and R, in contrast to their inverses L^{-1} and R^{-1} are sparse matrices[1]. They result from the successive application of line and row operations like they are used in the standard gaussian elimination. Therefore, L and R contain one non-zero matrix element for each eliminated coupling in the system matrix A. If we have a subdomain with m separator unknowns, we eliminate all couplings between $\mathcal{O}(\sqrt{m})$ inner unknowns and $\mathcal{O}(\sqrt{m})$ outer unknowns. This consequently produces $\mathcal{O}(m)$ entries in L and R, such that the storage complexity does not deteriorate.

However, as a result of these extra eliminations, the system matrices A suffer from severe fill-in and should more or less be regarded as full matrices. Therefore, one can no longer afford to store the system matrices A as this would result in an $\mathcal{O}(N \log N)$ memory requirement. This can be avoided by choosing the (weighted) Jacobi-method for relaxation. Then it is sufficient to store only the main diagonal of A as the residuals can be computed 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 setup of the system matrices back to $\mathcal{O}(N)$ (note that also the setup of the matrices A can be restricted mainly to the diagonal elements). Of course the Jacobi-relaxation gives less satisfying convergence rates than for example the Gauss-Seidel-relaxation, but, as we will show in the next section, it is still possible to achieve reasonable performance.

[1] Therefore all transformations including L^{-1} or R^{-1} are implemented via L^{-1} or R^{-1} respectively

4 Numerical Results

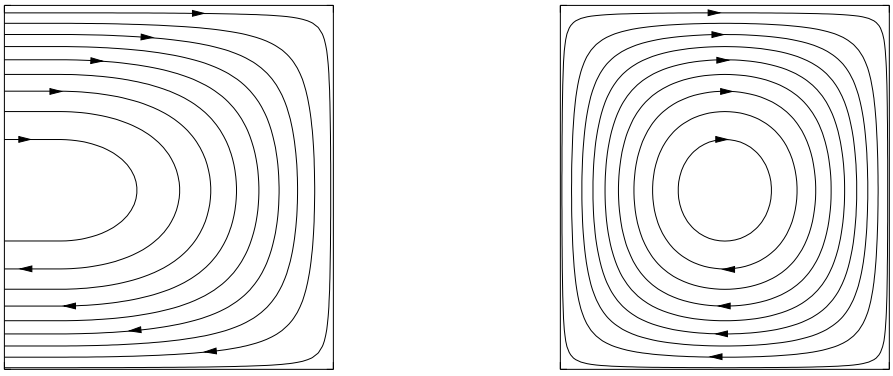
We tested our algorithm on two common benchmark problems, which are given by equation 1 on the unit square with the convection fields

$$\textbf{problem (a):} \quad a(x,y) := \begin{cases} a_0 \cdot \begin{pmatrix} (2y-1)(1-\bar{x}^2) \\ 2\bar{x}y(y-1) \end{pmatrix} & \text{for} \quad \bar{x} > 0 \\ a_0 \cdot \begin{pmatrix} 2y-1 \\ 0 \end{pmatrix} & \text{for} \quad \bar{x} \leq 0\,, \end{cases} \tag{6}$$

where $\bar{x} := 1.2x - 0.2$, and

$$\textbf{problem (b):} \quad a(x,y) := a_0 \cdot \begin{pmatrix} 4x(x-1)(1-2y) \\ -4y(y-1)(1-2x) \end{pmatrix}. \tag{7}$$

Problem (a) models an entering flow that contains a 180 degree curve. Problem (c) is an example for a circular flow problem. Figure 6 shows the streamlines of both convection fields. Each problem was discretized on the unit square using standard second order finite difference discretization with a five point stencil.

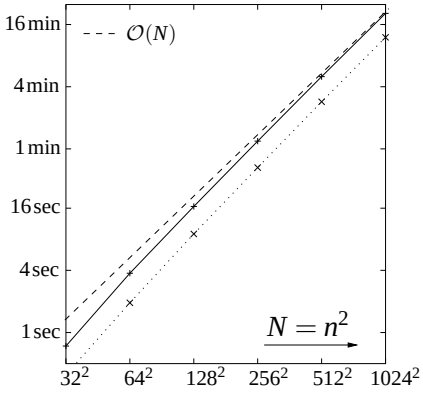


(a) bent pipe convection (b) circular convection

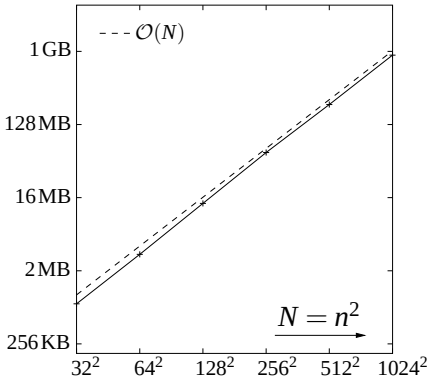
Figure 6: Convection fields of the two test problems

As the coarse grid points and eliminated couplings are chosen independently of the convection field it is obvious that computing time and memory requirement are independent of the type of the test problem. Figure 7 shows that both computing time and memory requirement rise linearly with the number of unknowns. This can also be deduced by theoretical means.

Figure 8 shows the convergence rates for the two benchmark problems. It also shows the convergence rates when the algorithm is applied as a preconditioner for the Bi-CGSTAB method [9]. We can see that the speed of convergence is independent of the type of problem. It is also independent of the number of unknowns. It is even independent of the strength of convection as long as a certain ratio between convection, diffusion and mesh size is not exceeded. More precisely the mesh Peclet number should not exceed a certain value on the finest grid. This upper bound for the Peclet number coincides with the applicability of the central differencing scheme



Computing time for setup (——)
and complete solution (···)



Memory requirement

Figure 7: Computing time and memory requirement on an SGI-ORIGIN workstation

in the discretization. For stronger convection the heuristics of our elimination strategy no longer holds and convergence begins to break down.

5 Present and Future Work

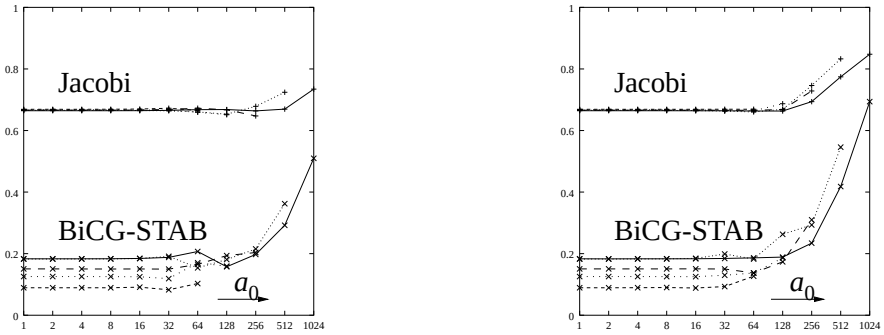
We are currently working on the treatment of arbitrary shaped computational domains including interior boundaries and obstacles. For this purpose we plan to use a quadtree or octree representation of the geometry and exploit the generated tree structure for our substructuring approach. For Poisson type equations first results will be published in [1]

Future work will include the efficient treatment of strongly convection dominated flow. In the case of strong convection our elimination strategy is no longer appropriate as it depends on the existence of a certain amount of physical diffusion. For very strong convection the distance h between the coarse grid unknowns (see figure 5) might well have to be the mesh size of the finest grid. In that case a complete elimination between all coarse grid unknowns would result in a direct nested dissection solver which would certainly be too expensive. An efficient elimination strategy should be achievable either by eliminating all couplings that are ”aligned” with the direction of the flow or by choosing the eliminated couplings in an algebraic manner which leads to methods that differ from algebraic multigrid only in the fixed coarse grid selection.

References

[1] M. Bader, A. Frank, and C. Zenger. An octree-based approach for fast elliptic solvers. In *International FORTWIHR Conference 2001 (Proceedings, to appear).*

[2] M. Bader and C. Zenger. A fast solver for convection diffusion equations based on nested dissection with incomplete elimination. In Arndt Bode et. al., editor, *Euro-Par 2000, Parallel Processing*, pages 795–805, 2000.



problem (a) problem (b)

Figure 8: Average convergence rates for the two benchmark problems using simple Jacobi or preconditioned BiCG-STAB on computational grids with 64×64 (- -), 128×128 ($\cdot\,\cdot\,\cdot$), 256×256 (– –), 512×512 ($\cdots$) and 1024×1024 (——) unknowns.

[3] R. Ebner and C. Zenger. A distributed functional framework for recursive finite element simulation. *Parallel Computing*, 25:813–826, 1999.

[4] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10, 1973.

[5] M. Griebel. Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM Journal of Scientific and Statistical Computing*, 15(3):547–565, 1994.

[6] R. Hüttl and M. Schneider. Parallel adaptive numerical simulation. SFB-Bericht 342/01/94 A, Institut für Informatik, TU München, 1994.

[7] J. Ruge and K. Stüben. Algebraic multigrid. In S.F. McCormick, editor, *Multigrid Methods*, Frontiers in Applied Mathematics, pages 73–130. SIAM, 1987.

[8] B.F. Smith and O.B. Widlund. A domain decomposition algorithm using a hierarchical basis. *SIAM Journal of Scientific and Statistical Computing*, 11(6):1212–1220, 1990.

[9] H. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 13(2):631–644, 1992.