Electronic Transactions on Numerical Analysis.
Volume 15, pp. 122-131, 2003.
Copyright © 2003, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A ROBUST AND PARALLEL MULTIGRID METHOD FOR CONVECTION DIFFUSION EQUATIONS[*]

MICHAEL BADER AND CHRISTOPH ZENGER[†]

Abstract.
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 of couplings between certain "coarse grid unknowns".

The choice of these coarse grid unknowns is motivated by the physical properties of the convection diffusion equation, but it is independent of the actual discretized operator. The resulting algorithm has a memory requirement that grows linearly with the number of unknowns; likewise do the computational costs of the setup and of the individual relaxation cycles. We present numerical examples that indicate that the number of iterations needed to solve a convection diffusion equation is also substantially independent of the number of unknowns and of the type and strength of the convection field.

Key words. convection diffusion equation, multigrid, substructuring method, parallelization

AMS subject classifications. 65N55, 76R99, 65Y05

1. Introduction.

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

$$\begin{aligned} -\triangle u + a(x,y)\cdot\nabla u &= f \qquad \text{in } \Omega\subset\mathbf{R}^2, \\ u &= g \qquad \text{on } \partial\Omega, \end{aligned} \tag{1.1}$$

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 does not 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 fairly easy to parallelize due to their structured coarse grids. However, the need for robustness poses quite severe demands on the applied smoothers [3], which again can impair the parallel efficiency of the method. Furthermore, 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 is mainly 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 [8] for the coarse grid selection is an inherently sequential process, which means that the construction of the different coarse grids itself has to be modified.

In this paper, we will present an approach which combines ideas from recursive substructuring techniques with the discretization on hierarchical bases and generating systems. The resulting multilevel method is extended by an additional partial elimination. This partial elimination is a simplified computation of the Schur complement in which only the couplings

[*] Received May 23, 2001. Accepted for publication October 21, 2001. Recommended by Jim Jones.

[†]Institut für Informatik, TU München, 80290 München, Germany, {bader,zenger}@in.tum.de

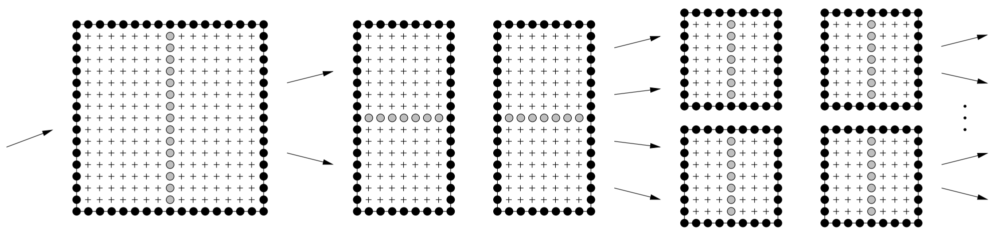


FIG. 2.1. *Recursive substructuring by alternate bisection. Outer unknowns (set $\mathcal{E}$) are given as black circles, inner unknowns (set $\mathcal{I}$) as grey circles. The tiny crosses (+) indicate eliminated unknowns that are no longer used on the respective subdomain.*

between certain *coarse grid unknowns* are eliminated. These coarse grid unknowns are chosen with respect to the underlying physics of the convection diffusion equation. The overall approach can also be regarded as a variant of an algebraic multigrid method in which the coarse grid selection is fixed and only the coarse grid operators are constructed from the actual discretization.

After a short discussion of the underlying recursive substructuring method in section 2, we will, in section 3, describe the extension of this approach 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 [5]. The later described iterative variant will differ from this 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 divided into two subdomains that are connected by a *separator*. This is repeated recursively (*alternate bisection*, see figure 2.1) until the subdomains consist of just a couple of unknowns per dimension and can not be subdivided any further.

On each subdomain, we then define the set $\mathcal{E}$ of outer unknowns (which lie on the subdomain boundary) and the set $\mathcal{I}$ of inner unknowns, which lie on the separator but do not belong to the separator of a parent domain. Figure 2.1 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 paragraph.

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 child domains. First, the system matrix and right hand side is assembled from those of the child domains:

$$A := \begin{pmatrix} A_{\mathcal{E}\mathcal{E}} & A_{\mathcal{E}\mathcal{I}} \\ A_{\mathcal{I}\mathcal{E}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix} := V_{(1)}^T \left(A_{(1)}\right)_{\mathcal{E}\mathcal{E}} V_{(1)} + V_{(2)}^T \left(A_{(2)}\right)_{\mathcal{E}\mathcal{E}} V_{(2)} \tag{2.1}$$

$$b := V_{(1)}^T b_{(1)} + V_{(2)}^T b_{(2)}. \tag{2.2}$$

Note that only those parts of A and b that belong to the outer unknowns $(\in \mathcal{E})$ 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 $(A_{\mathcal{E}\mathcal{E}}, A_{\mathcal{I}\mathcal{I}}, \dots)$ that enable the following block elimination.

In this block elimination, the inner and outer unknowns are decoupled,

$$\begin{pmatrix} \mathrm{Id} & -A_{\mathcal{E}\mathcal{I}}A_{\mathcal{I}\mathcal{I}}^{-1} \\ 0 & \mathrm{Id} \end{pmatrix} \begin{pmatrix} A_{\mathcal{E}\mathcal{E}} & A_{\mathcal{E}\mathcal{I}} \\ A_{\mathcal{I}\mathcal{E}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix} \begin{pmatrix} \mathrm{Id} & 0 \\ -A_{\mathcal{I}\mathcal{I}}^{-1}A_{\mathcal{I}\mathcal{E}} & \mathrm{Id} \end{pmatrix} = \begin{pmatrix} \widetilde{A}_{\mathcal{E}\mathcal{E}} & 0 \\ 0 & A_{\mathcal{I}\mathcal{I}} \end{pmatrix}, \tag{2.3}$$

by computing the Schur complement

$$\widetilde{A}_{\mathcal{E}\mathcal{E}} = A_{\mathcal{E}\mathcal{E}} - A_{\mathcal{E}\mathcal{I}} \cdot A_{\mathcal{I}\mathcal{I}}^{-1} \cdot A_{\mathcal{I}\mathcal{E}}. \tag{2.4}$$

Of course, the right-hand sides have to be treated accordingly, which leads to a transformed system of equations

$$\widetilde{A}\widetilde{x} = \widetilde{b}, \tag{2.5}$$

where

$$\widetilde{b} = \begin{pmatrix} \mathrm{Id} & -A_{\mathcal{E}\mathcal{I}}A_{\mathcal{I}\mathcal{I}}^{-1} \\ 0 & \mathrm{Id} \end{pmatrix} b \quad \text{and} \quad x = \begin{pmatrix} \mathrm{Id} & 0 \\ -A_{\mathcal{I}\mathcal{I}}^{-1}A_{\mathcal{I}\mathcal{E}} & \mathrm{Id} \end{pmatrix} \widetilde{x}. \tag{2.6}$$

The matrix block $\widetilde{A}_{\mathcal{E}\mathcal{E}}$ and the corresponding 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 2.1.

Solution. After the setup of the local systems of equations is completed, the actual solution can be computed in a single, top-down traversal of the subdomain tree. Starting with the separator of the entire computational domain, on each subdomain the values of the separator unknowns are computed from the local systems:

$$\widetilde{x}_{\mathcal{I}} = A_{\mathcal{I}\mathcal{I}}^{-1} b_{\mathcal{I}}. \tag{2.7}$$

After computing the actual solution x from the transformation given in equation 2.6, the solution x is transferred to the child domains which can then proceed with the solution cycle.

To compute the transformation of x in 2.6, we need to know the outer unknowns $\widetilde{x}_{\mathcal{E}}$. For all subdomains except the root domain, the values $\widetilde{x}_{\mathcal{E}}$ are transferred from the respective parent domain. For the root domain itself, the values of $\widetilde{x}_{\mathcal{E}}$ are usually given by the boundary conditions. As an alternative, the outer unknowns can also be eliminated beforehand, such that only inner unknowns are present on the root domain (this would probably be the method of choice for Neumann boundary conditions).

2.2. Iterative Version of the Substructuring Method. The direct solver described in the previous section would, like the very similar *nested dissection* method [5], 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. The most expensive part of the computational work results from computing the inverse $A_{\mathcal{I}\mathcal{I}}^{-1}$ in the Schur complement and the solution of the local systems in equation 2.7. 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 instead [7, 9].

The static condensation pass is reduced to the assembly of the local system matrices and a hierarchical transformation which replaces the computation of the Schur complement. The

ETNA
Kent State University
etna@mcs.kent.edu

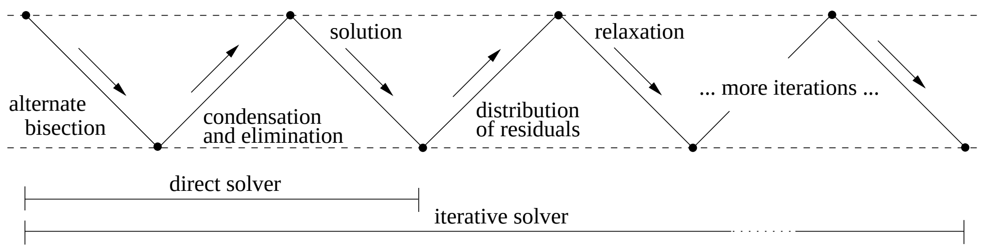


FIG. 2.2. *The different passes of the iterative substructuring algorithm*

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

FIG. 2.3. *Iterative substructuring algorithm: steps (S3), (U3) and (D2) are only needed if partial elimination is used*

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 2.2 illustrates the overall scheme of such an iterative substructuring method. The general structure of the resulting algorithm is given in figure 2.3. It already includes the additional partial elimination (see steps S3, U3, and D2) 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 facilitate the parallelization of the algorithm. The parallelization of iterative substructuring algorithms, like the one described in figure 2.3, was analysed for example by Hüttl [7] and Ebner [4]. It is principally based on a distributed memory approach where the subdomains themselves are not parallelized but can be distributed to different processors. For instance, figure 2.4 shows the distribution of a certain subdomain tree to 16 processors. As the interaction between subdomains in the subdomain tree is strictly restricted to parent-child-communications, the parallelization becomes very easy.

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

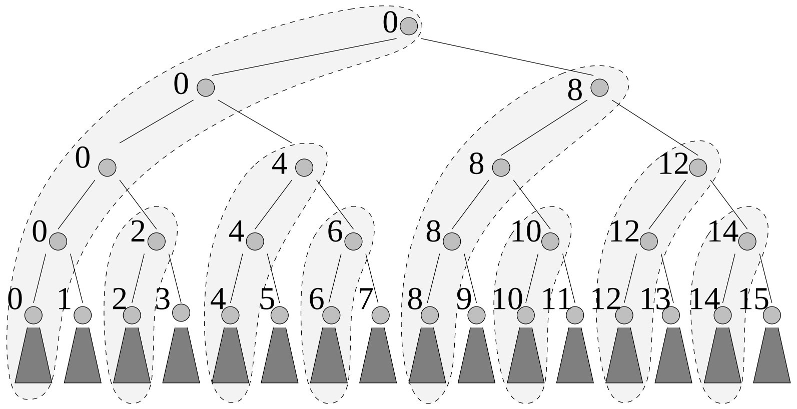


FIG. 2.4. *Distribution of the subdomains to 16 processors*

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 [6].

The key idea is to enhance the hierarchical preconditioning by an additional partial elimination, like it was already included in the algorithm in figure 2.3. Certain couplings — hopefully the strongest — in the local systems of equations are chosen which are then eliminated explicitly by the elimination matrices L and R.

The complete decoupling of inner and outer unknowns, like that obtained by the computation of the Schur complement in the equations 2.3 and 2.4, is therefore limited to a transformation

$$\underbrace{L^{-1}\bar{A}\,R^{-1}}_{=:\,\widetilde{A}}\,\widetilde{x} = \underbrace{L^{-1}b}_{=:\,\widetilde{b}}. \tag{3.1}$$

The elimination matrices L^{-1} and R^{-1} are chosen such that certain elements of 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, all couplings between those coarse grid unknowns are eliminated.

If hierarchical bases or generating systems are applied, it is natural to choose the hierarchically coarse unknowns as those coarse grid unknowns. In that case, the couplings with and between hierarchically fine unknowns are usually small, as their contribution to the transport of the quantity u is only small. 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 3.1 illustrates this by an example of some heat transport problem. It shows a certain subdomain with a heat source on the left border. The constant convection field transports the generated heat towards the domain separator. Due to diffusion, the former temperature peak will extend over several mesh points after it has been transported to the separator. It is clear that the resulting temperature profile cannot be resolved very well by using only the central point on the separator as a coarse grid unknown (left picture in fig. 3.1). On the other hand, it would also be unnecessary to use all the fine grid points on the

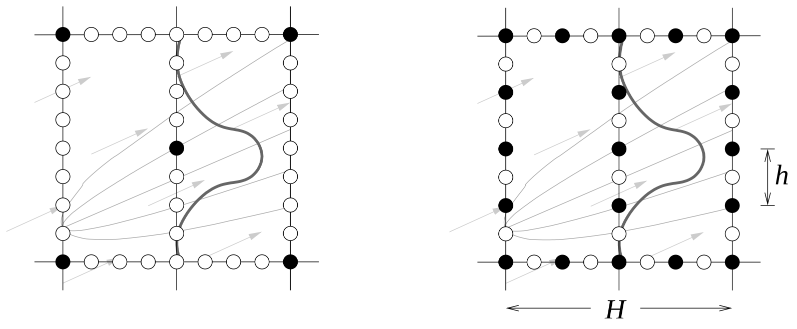


FIG. 3.1. *Comparison of different elimination strategies: only the couplings between the black nodes are eliminated*

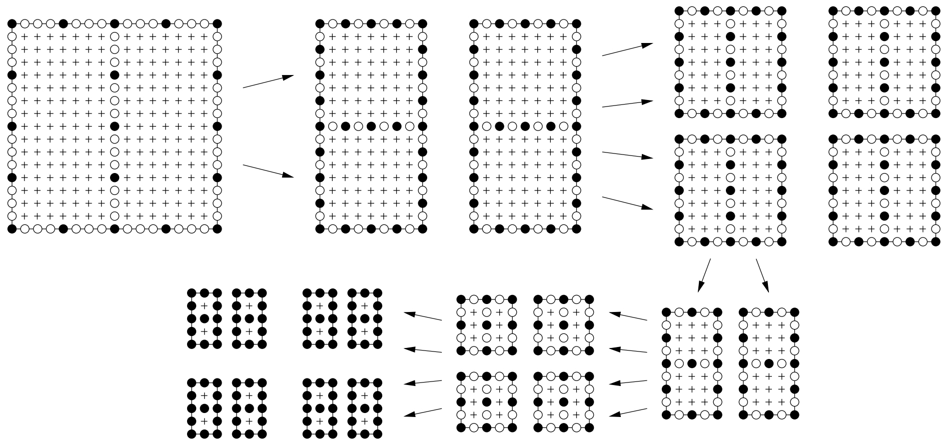


FIG. 3.2. *Bisection with incomplete elimination, only the couplings between the black nodes are eliminated*

separator. We therefore have to look for a good compromise for the resolution h of the coarse grid unknowns (right picture in fig. 3.1) 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 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 3.2.

In the algorithm of figure 2.3 the partial elimination is already included. 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 $\widetilde{A}$. 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 produces $\mathcal{O}(m)$ entries in L and R. Thus, not only the memory requirement for storing the matrices L and R, but also the computing time of the iteration cycles (steps U1–U3 and D1–D4 in figure 2.3) grows linearly with the number of unknowns.

However, as a result of the extra eliminations, the system matrices $\widetilde{A}$ suffer from severe

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

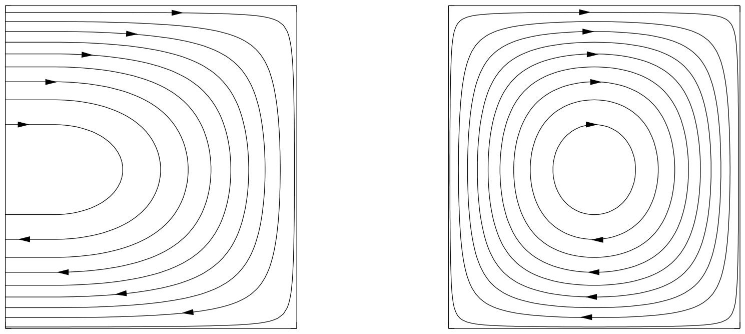

(a) bent pipe convection (b) circular convection

FIG. 4.1. *Convection fields of the two test problems*

fill-in and should more or less be regarded as full matrices. Therefore, one can no longer afford to store the entire system matrices $\widetilde{A}$ as this would result in an $\mathcal{O}(N \log N)$ memory requirement. This can be avoided by performing only one single (weighted) Jacobi-step to approximately solve the local systems of equations. Then it is sufficient to store only the main diagonal of $\widetilde{A}$, as the residuals can be computed separately (see steps U1–U3 of the algorithm in figure 2.3). This reduces both the memory requirement and the number of operations needed for the setup of the system matrices to $\mathcal{O}(N)$, again (note that also the setup of the matrices $\widetilde{A}$ can be restricted mainly to the diagonal elements). As a result of this, the algorithm turns into a purely additive multigrid method with just the Jacobi-relaxation as a “smoother”. Of course, this gives rise to less satisfying convergence rates than one would expect from multiplicative methods. As we will show in the following section, it is nevertheless possible to achieve reasonable performance.

4. Numerical Results.

We tested the resulting algorithm on a variety of benchmark problems. The results for two rather common convection diffusion problems will be presented in this section. Both problems are given by the equations 1.1 with $\Omega := (0,1) \times (0,1)$. Problem (a) models an entering flow that contains a 180 degree curve. It is specified by the convection field

$$\text{(4.1)} \qquad \textbf{problem (a):} \quad a(x,y) := \begin{cases} a_0 \cdot \begin{pmatrix} (2y-1)(1-\bar{x}^2) \\ 2\bar{x}y(y-1) \end{pmatrix} & \text{for} \quad \bar{x} > 0 \\ a_0 \cdot \begin{pmatrix} 2y-1 \\ 0 \end{pmatrix} & \text{for} \quad \bar{x} \le 0\,, \end{cases}$$

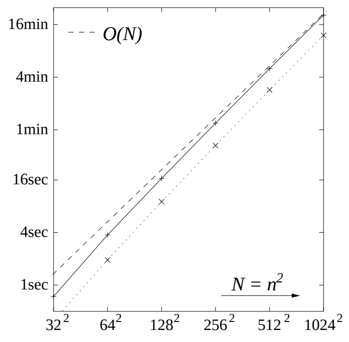
where $\bar{x} := 1.2x - 0.2$, and by the Dirichlet boundary conditions

$$\text{(4.2)} \qquad g(x,y) := \sin(\pi x) + \sin(13\pi x) + \sin(\pi y) + \sin(13\pi y) \qquad \text{on } \partial\Omega.$$

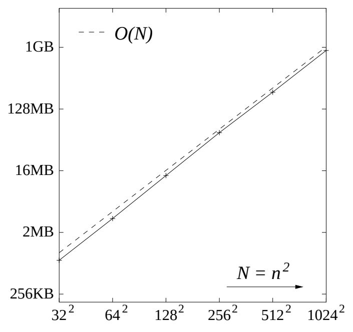
Problem (b) is an example of a circular flow problem. Its convection field is

$$\text{(4.3)} \qquad \textbf{problem (b):} \quad a(x,y) := a_0 \cdot \begin{pmatrix} 4x(x-1)(1-2y) \\ -4y(y-1)(1-2x) \end{pmatrix},$$

and it has the same boundary conditions as problem (a). For both problems, the right hand side was chosen to be $f = 0$. Figure 4.1 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.



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



Memory requirement

FIG. 4.2. *Computing time and memory requirement on an* SGI-ORIGIN *workstation*

As the coarse grid unknowns and eliminated couplings are chosen independently of the convection field, it is obvious that computing time (for setup and per iteration) and memory requirement are independent of the type of the test problem. Figure 4.2 shows that both computing time and memory requirement grow linearly with the number of unknowns. This can also be deduced by theoretical means. In fact, figure 4.2 already shows that the computing time for the complete solution grows linearly with the number of unknowns, but, as we will see, this only holds for a modest strength of convection.

Figure 4.3 shows the convergence rates for the two benchmark problems for problems of different size and for different strength a_0 of convection. It also shows the convergence rates when the algorithm is applied as a preconditioner for the Bi-CGSTAB method [10]. We can see that the speed of convergence is independent of the type of the 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 Péclet number should not exceed a certain value on the finest grid. This upper bound for the Péclet number coincides with the applicability of the central differencing scheme in the discretization.

For stronger convection, the heuristics of our elimination strategy no longer holds, and convergence begins to break down. The strong couplings then no longer occur between the hierarchically coarse unknowns but between unknowns that are placed on the same streamlines. In that case, a different elimination strategy would be necessary (see sec. 5).

A parallel implementation of the presented iterative substructuring algorithm was done using MPI as the message passing protocol. The subdomains were distributed to the processors like demonstrated in figure 2.4. We ran the different benchmark problems with varying problem size and increasing number of processors on a cluster of 32 SUN Ultra 60 workstations. Figure 4.4 illustrates the observed speedup and parallel efficiency. The diagrams show that the parallelization is efficient even on architectures that are not dedicated parallel computers.

5. Conclusion and Future Work.

We presented an inherently parallel multigrid method for the solution of convection diffusion equations which is of optimal complexity with respect to storage and computing time. The speed of convergence is largely independent of the strength and geometry of the convection field. First numerical results also indicate that fast convergence can be achieved independent of the geometry of the computational domain. For a general treatment of arbitrary shaped computational domains, including interior boundaries

ETNA
Kent State University
etna@mcs.kent.edu

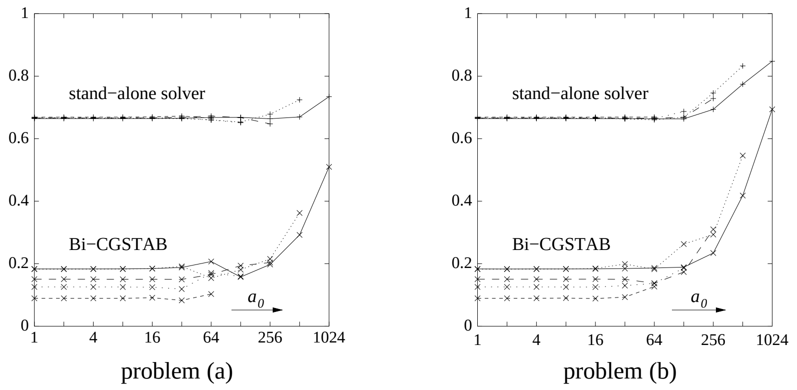


FIG. 4.3. *Average convergence rates in dependence of the strength of convection. Results are given for computational grids with* 64×64 *(- -),* 128×128 *(· · ·),* 256×256 *(– –),* 512×512 *(···), and* 1024×1024 *(—) unknowns.*

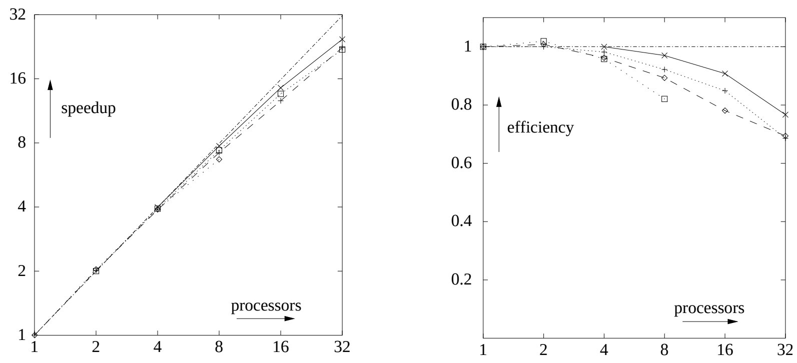


FIG. 4.4. *Speedup and parallel efficiency on a cluster of* SUN Ultra 60 *workstations for problems with* 128×128 *(· · ·),* 256×256 *(– –),* 512×512 *(···), and* 1024×1024 *(—) unknowns.*

and obstacles, we plan to use a quadtree or octree representation of the geometry and exploit the generated tree structure for our substructuring approach [1].

Future work will include the efficient treatment of strongly convection dominated flow and, of course, the extension of the method to 3D problems. In the case of very strong convection, our elimination strategy is no longer appropriate, as it depends on the existence of a certain amount of physical diffusion. The distance h between the coarse grid unknowns (see figure 3.2) then might have to be the mesh size of the finest grid. This would lead to a complete elimination between all coarse grid unknowns and would result in a direct nested dissection solver which would certainly be too expensive.

The extension to 3D raises a similar problem. While the direct nested dissection has an operation count of $\mathcal{O}(N^{3/2})$ in 2D, it requires $\mathcal{O}(N^2)$ operations in 3D. Likewise, a straightforward translation of our approach to the 3D case would result in a method that requires at least $\mathcal{O}(N \log N)$ operations.

Therefore, the elimination strategy has to be modified for both cases. For example, we could restrict the partial elimination to couplings that are "aligned" with the direction of the flow. Another possibility is to choose the eliminated couplings in a purely algebraic manner (i.e. choosing the actually strongest couplings). The resulting algorithms would be very similar to algebraic multigrid methods. However, they would retain the fixed coarse grid selection and, therefore, the inherently parallel structure of the method presented in this paper.

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 Euro-Par 2000, Parallel Processing, A. Bode, T. Ludwig, W. Karl, and R. Wismüller, eds., 2000, pp. 795–805.

[3] A. BRANDT AND I. YAVNEH, *On multigrid solution of high-reynolds incompressible flows*, Journal of Computational Physics, 101 (1992), pp. 151–164.

[4] R. EBNER AND C. ZENGER, *A distributed functional framework for recursive finite element simulation*, Parallel Computing, 25 (1999), pp. 813–826.

[5] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973).

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

[7] R. HÜTTL AND M. SCHNEIDER, *Parallel adaptive numerical simulation*, SFB-Bericht 342/01/94 A, Institut für Informatik, TU München, 1994.

[8] J. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid Methods, S. McCormick, ed., Frontiers in Applied Mathematics, SIAM, 1987, pp. 73–130.

[9] B. SMITH AND O. WIDLUND, *A domain decomposition algorithm using a hierarchical basis*, SIAM Journal of Scientific and Statistical Computing, 11 (1990), pp. 1212–1220.

[10] 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 (1992), pp. 631–644.