PDE SOFTWARE: Modules, Interfaces and Systems
B. Engquist and T. Smedsaas (eds.)
Elsevier Science Publishers B.V. (North-Holland)

A PORTABLE VECTOR-CODE FOR AUTONOMOUS MULTIGRID MODULES

P.W. Hemker, P. Wesseling & P.M. de Zeeuw
Centre for Mathematics and Computer Science, Amsterdam,
and University of Technology, Delft,
The Netherlands

The implementation is described of two multigrid algorithms for use as standard subroutines for the efficient solution of linear systems that arise from 7-point discretizations of elliptic PDEs on a rectangle. For both algorithms a tuned scalar-version and a tuned vector-version have been constructed and run on a CYBER 170, a CRAY 1 and a CYBER 205. The CPU-times are given and compared. The implementation is available in portable ANSI-FORTRAN.

INTRODUCTION

In this paper we describe software for the solution of discretized 2^{nd} order linear elliptic PDEs in two space dimensions. The domain of definition is assumed to be a rectangle and the discretization is assumed to result in a regular 7-diagonal matrix. The algorithms, based on multigrid cycling, are selected for efficiency. The aim was to obtain software that is perceived and can be used just like any standard subroutine for solving systems of linear equations. The user has to specify only the matrix and the right-hand-side, and remains unaware of the underlying multigrid method. Such a subroutine, that operates without outside interference, will be called autonomous. We find that a large class of equations can be solved efficiently by use of our autonomous multigrid subroutines. The equation may be non-self-adjoint, and its coefficients are arbitrary.
The two algorithms use saw-tooth multigrid cycles [8,9]. One algorithm is based on ILU-relaxation, the other on ZEBRA- relaxation [7]. The discretization on coarse grids is provided automatically by means of a built-in Galerkin approximation.
Various scalar- and vector-versions of the code have been constructed and run on a CYBER 170, a CRAY 1 and a CYBER 205. It appeared that a code written for automatic vectorization in portable ANSI FORTRAN runs efficiently in all cases. Specially tuned versions are only a small fraction more efficient.
In section 2 we describe the class of problems that can be solved. In section 3 we describe the general algorithm for multigrid cycling. In the sections 4, 5, 6 we specialize the general algorithm and come to the various versions of the codes. In sections 7, 8 we compare the various programs. In the last sections we formulate some conclusion.

2. THE PROBLEM

We consider the linear 2^{nd} order elliptic PDE in two dimensions

$$(2.1.a)\qquad \sum_{i,j=1,2} a_{ij}\left(\frac{\partial}{\partial x_i}\right)\left(\frac{\partial}{\partial x_j}\right) u + \sum_{i=1,2} a_i\left(\frac{\partial}{\partial x_i}\right) u + a_0 u = f \text{ on } \Omega \subset \mathbb{R}^2,$$

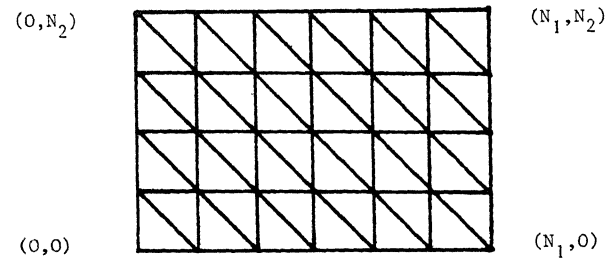
with variable coefficients and with boundary conditions on $\delta\Omega = \Gamma_N \cup \Gamma_D$

$$(2.1.b)\qquad \left(\frac{\partial}{\partial n}\right) u + \alpha\left(\frac{\partial}{\partial s}\right) u + \beta u = \gamma \text{ on } \Gamma_N,$$

$$u = g \text{ on } \Gamma_D.$$

The coefficients are arbitrary but should satisfy the ellipticity condition. If

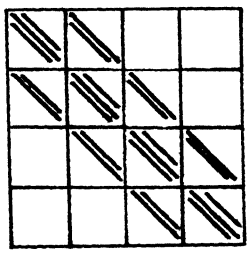
this equation on a rectangle Ω is discretized on a regular triangularization of the form



then the resulting discretization

$$A_h u_h = f_h \tag{2.2}$$

can be a linear system with a regular 7-diagonal structure.



The shape of a matrix A_h.

We consider codes for the solution of linear systems with a structure corresponding to this kind of 7-point discretization. On the rectangle Ω equidistant computational grids Ω^k, $k = 1,2,\ldots,\ell$, are defined

$$\Omega^k = \{(x_1,x_2) \mid x_i = m_i 2^{1-k},\ m_i = 0(1)N_i 2^{k-1}\}.$$

The user has to provide the discrete operator A_h and the data f_h only on the finest grid Ω^ℓ.
To solve the linear system efficiently, multigrid methods are used [2,7,8]. These methods make also use of Ω^k, $k = 1,2,\ldots,\ell-1$, but when using the autonomous subroutines the user remains unaware of this fact. Much effort has been spent in the search for efficient variants of the MG-method [3,4,5,6,7,8]. In this paper we onsider only two variants that are found to belong to the more promising ones.

. THE GENERAL MULTIGRID ALGORITHM

.he general multigrid cycling algorithm for the solution of (2.2) is an iterative process, which makes use of a sequence of discretizations on the grids Ω^k, $k = 1,2,\ldots,\ell$.
Each multigrid iteration cycle consists of
(1) p relaxation sweeps, followed by
(2) a coarse grid correction, followed by
(3) q more relaxation sweeps.
The coarse grid correction consists of
a) the computation of the current residual $r_h = f_h - A_h\tilde{u}_h$;
b) the restriction of the residual to the next coarser grid $r_H = R_{Hh}r_h$;
c) the computation of $\tilde{c}_H$, an approximation to the solution of the correction

equation

(3.1) $$A_H c_H = r_H.$$

This approximation is obtained by application of s multigrid iteration cycles to this equation, and

(d) updating the current solution $\tilde{u}_h$ by addition of the prolongated correction

$$\tilde{u}_h := \tilde{u}_h + P_{hH}\tilde{c}_H.$$

On the coarsest grid the correction equation (3.1) has to be (approximately) solved by another method (at choice). The coarse grid discrete operators A_H can be constructed by analogy to (2.2) or by Galerkin approximation:

(3.2) $$A_H = R_{Hh}A_h P_{hH}$$

(cf. [8]).

4. THE ALGORITHMS

In this paper we consider the implementation of two particular instances of the multigrid algorithms: MGD1 and MGEOZ. Based on various comparisons [3,6,9], the parameters p, q and s are chosen to be 0,1 and 1 respectively. The resulting strategy is called a saw-tooth cycle [9]. For the prolongation and the restriction 7-point operators are chosen, that correspond to linear interpolation on coarse-grid triangles in the triangulation of Ω (for P_{hH}) and to its adjoint operator (for R_{Hh}). The Galerkin approximation (3.2) is chosen for the construction of the coarse grid operators A_H. For an approximate solution on the coarsest grid a single relaxation sweep is used. The two algorithms differ only with respect to he relaxation method.
In MGD1 the Incomplete LU (ILU-) relaxation is used [9]. For this relaxation the 7-diagonal matrix A_h is decomposed as

$$A_h = LU - C$$

where L is a lower-triangular matrix (with 1 for all main-diagonal elements) and U is an upper triangular matrix. The requirement that L and U have non-zero diagonals only where A_h has, determines L and U. The rest-matrix C has only two non-zero diagonals, of which the elements are easily derived from L and U. One relaxation sweep of the Incomplete LU-relaxation is now the solution of the system

$$LU\, u^{(i+1)} = f + C\, u^{(i)}.$$

After such a relaxation sweep a residual is efficiently computed by

$$r^{(i+1)} := f_h - A_h u_h^{(i+1)} = C(u_h^{(i+1)} - u_h^{(i)})$$

In MGLOZ the ZEBRA-relaxation is used [7]. This is a line-Gauss-Seidel relaxation in which first all points on even lines (lines that appear in the coarser grid) are simultaneously relaxed and secondly all points on the odd lines. An important advantage of this relaxation is the fact that many points can be relaxed simultaneously and that the residual computation simplifies, because the residual vanishes at all odd lines after a relaxation sweep. For ZEBRA relaxation tridiagonal systems have to be solved. The solution of these systems can be accelerated by storage of the decomposition of the tridiagonal matrices. We have chosen to do this at an extra storage cost of 2 reals per grid-point.

5. THE STRUCTURE OF MGD1 AND MGEOZ

The general structure of both MGD1 and MGEOZ is the same. First, in the preparational phase, the sequence of coarse grid discrete operators is constructed by

a subroutine RAP, according to (3.2). Then the decomposition is performed (in DECOMP or DECOMPZ) and the initial estimate of the solution is set to zero. Finally, in the cycling phase, at most MAXIT iterations of the cycling process are performed. On the basis of intermediate results the iteration can be stopped earlier; this necessitates the computation of a vectornorm (in VL2NOR).
In the following the structure of the cycling process of MGD1 is described in quasi-FORTRAN.

```
      DO 10 k=ℓ-1(-1)1
      CALL RESTRICTION (f,f,k).                 f^k = R_Hh f^(k+1)
   10 CONTINUE
C     START OF maxit MULTIGRID ITERATIONS
      DO 50 n=1, maxit
      IF (n.EQ.1) GO TO 30
      CALL CTUMV (C,u,v)                        v^ℓ = C^ℓ(u^ℓ-v^ℓ)
C     v^ℓ IS THE NEW RESIDUE f^ℓ-A^ℓu^ℓ
      CALL RESTRICTION (f,v,ℓ-1)                f^(ℓ-1) = R_Hh v^ℓ
      DO 20 k=ℓ-2(-1)1
      CALL RESTRICTION (f,f,k)                  f^k = R_Hh f^(k+1)
   20 CONTINUE
   30 CALL SOLVE (u,f,1)                        u^1 = (L^1U^1)^-1 f^1
      DO 40 k=2 (1)ℓ-1
      CALL PROLONGATION (u,u,k)                 u^k = P_hH u^(k-1)
      CALL CTUPF (v,u,f,k)                      v^k = C^k u^k+f^k
      CALL SOLVE (u,v,k)                        u^k = (L^kU^k)^-1 v^k
   40 CONTINUE
      CALL PROLONGATION (v,u,ℓ)                 v^ℓ = P_hH u^(k-1)
      v^ℓ=v^ℓ+u^ℓ
      CALL CTUPF (u,v,f,ℓ)                      u^ℓ = C^ℓ v^ℓ+f^ℓ
      CALL SOLVE (u,u,ℓ)                        u^ℓ = (L^ℓU^ℓ)^-1 u^ℓ
   50 CONTINUE
```

In the actual implementation of MGD1, the matrix A_h is not kept in storage, but is overwritten by L and U. At minimal costs, the rest-matrix $C = LU - A_h$ is recomputed each time from L and U (in the subroutines CTUMV and CTUPF). All subroutines mentioned have their own particular features that make them more or less feasible for vectorization. This can be seen in table (8.2).
The structure of MGEOZ is more straightforward and follows directly from the MG-algorithm in section 3. Here the original matrix is not overwritten by the decomposition.
Two main alternatives exist for the implementation of ZEBRA relaxation. The lines in the grid can be relaxed successively and vectorization can be applied to speed up the solution of each tridiagonal system. However, because the solution of tridiagonal systems is not very suitable for vectorization, and all tridiagonal systems have the same size, we have chosen the other possibility to exploit vectorization, namely we solve all linear systems in each half relaxation sweep simultaneously by treating the even (odd) systems in parallel. Because of the rectangular shape of Ω the data-structure both in MGD1 and MGEOZ is simple. The grid-values of

u_h and f_h are stored sequentially in one-dimensional arrays. They are ordered by grid and in each grid they are ordered by meshline. The diagonals of A_h are stored similarly in a two-dimensional array; the columns of the array corresponding to the diagonals of the matrix.

6. VERSIONS OF THE PROGRAM

To investigate the advantages of vectorization, different versions of the programs have been constructed. One version is written in portable FORTRAN and is tuned for execution on sequential hardware (MGD1S and MGEOZS). Another version, also written in portable FORTRAN was tuned for vectorization (MGD1V and MGEOZV). To keep the FORTRAN portable, we had to rely on the automatic vectorization capabilities of the compilers at hand. All loops for which vectorization was required could indeed be expressed in standard ANSI FORTRAN.
Other versions of the programs were considered as well. An interesting variant was (not portable) MGD1D. This version is the same as MGD1V except for two statements, containing a call to a STACKLIB routine for the recursive parts of the routine SOLVE in MGD1. The STACKLIB library, supplied for the CYBER 200 series, contains particularly efficient routines for vector operations that are not vectorizable because of recursion. For the comparison of MGD1D and MGD1V see table (8.2) and (8.3). For details about the implementation of the vectorized versions cf. [10].

7. THE TEST PROBLEM

In this study we are not interested in the *numerical* behaviour of the algorithms for different problems. We consider here only the efficiency of their implementation. Therefore, we may restrict ourselves to a single testproblem: the solution of Poisson's equation on the unit square with Dirichlet boundary conditions. Of course, for this problem other alternatives exist for its efficient solution. A program implementing one cycle in a Full Multigrid (FMG) algorithm, specially tuned for this problem on a CYBER 205, is described in [1]. Such a program may run much faster than our general purpose code. They report the solution on a 129 × 129 grid (with the usual 5-point discretization) in 0.006 sec. However, we did *not* adapt our codes in any way to this particular problem. For our codes the cost of one iteration is the same for any 7-point discretization of a problem (2.1) on grids of a given size.
In all cases reported here, the problem was solved on a mesh with

$$(2^{\ell+1}+1)^2$$

meshpoints. The length of most vectors in the program is $(2^{n+1}+1)^j$, n = 1(1)ℓ. j = 1 and j = 2 (i.e. they represent lines or complete grids on level n). We performed experiments for ℓ = 4,5,6,7. For problems with ℓ > 5 the size of the problem was too large to run on the available CYBER 170; for ℓ > 6 The problem was too large for the available CRAY 1 (Daresbury 1983).

8. THE EFFECT OF VECTORIZATION

In the tables (8.1)-(8.5) we give CPU-times for the programs mentioned in section 6. On the CYBER 205 and the CRAY 1 the vector-tuned versions ran with the vector option, the CPU-times mentioned for the scalar-tuned versions ran without. If we run the portable vector-code in scalar mode we sacrify about 5% CPU-time on the CYBER 205 (CRAY 1: about 9%) when compared with the tuned scalar-code in scalar mode.
In the tables we give the total CPU-time in seconds spent in runs with 10 iteration cycles, including the preparational work. Also the time spent in the various subroutines is presented. From these numbers we derive the time spent in a single cycle. Additionally, we give the average convergence factor in the iterative cycling (CONV).

MGD1S	CYBER 170	CRAY 1		CYBER 205		
LEVEL	5	5	6	5	6	7
GRID	65 × 65	65 × 65	129 × 129	65 × 65	129 × 129	257 × 257
CONV	4.7E-2	4.7E-2	4.3E-2	4.7E-2	4.3E-2	4.3E-2
RAP	0.186	0.088	0.313	0.084	0.317	1.232
DECOMP	0.061	0.031	0.120	0.050	0.195	0.764
SOLVE	0.311	0.150	0.582	0.153	0.594	2.265
CTUMV	0.098	0.066	0.261	0.079	0.315	1.134
CTUPF	0.143	0.088	0.347	0.099	0.390	1.498
PROLON	0.041	0.030	0.113	0.024	0.093	0.360
RESTRI	0.066	0.017	0.062	0.014	0.054	0.202
VL2NOR	0.030	0.022	0.087	0.015	0.059	0.233
TOTAL	1.030	0.522	1.994	0.565	2.141	8.101
CYCLE	0.066	0.035	0.137	0.037	0.145	0.546

Table 8.1. CPU-times (in seconds) of the program MGD1S for problems with different sizes, run in scalar mode on different machines.

MGD1V	CRAY 1		CYBER 205		
LEVEL	5	6	5	6	7
GRID	65 × 65	129 × 129	65 × 65	129 × 129	257 × 257
CONV	4.7E-2	4.3E-2	4.7E-2	4.3E-2	4.3E-2
RAP	0.033 (2.7)	0.085 (3.7)	0.022 (3.8)	0.053 (6.0)	0.151 (8.2)
DECOMP	0.010 (3.1)	0.037 (3.2)	0.012 (4.2)	0.043 (4.5)	0.162 (4.7)
SOLVE	0.086 (1.7)	0.324 (1.8)	0.091 (1.7)	0.325 (1.8)	1.251 (1.8)
CTUMV	0.008 (8.2)	0.032 (8.2)	0.003 (26!)	0.010 (31!)	0.042 (27!)
CTUPF	0.011 (8.0)	0.043 (8.1)	0.004 (25!)	0.014 (28!)	0.059 (25!)
PROLON	0.007 (4.3)	0.018 (6.3)	0.009 (2.7)	0.022 (4.2)	0.061 (5.9)
RESTRI	0.004 (4.2)	0.011 (5.6)	0.012 (1.2)	0.031 (1.7)	0.092 (2.2)
VL2NOR	0.003 (7.3)	0.010 (8.7)	0.001 (15)	0.004 (15)	0.015 (16)
TOTAL	0.169 (3.1)	0.575 (3.5)	0.177 (3.2)	0.533 (4.0)	1.882 (4.3)
CYCLE	0.012 (2.9)	0.043 (3.2)	0.012 (3.1)	0.040 (3.6)	0.151 (3.6)

Table 8.2. CPU-times (in seconds) of the program MGD1V run in vector-mode. Between brackets: the acceleration factor by vectorization (compared with the tuned scalar version).

MGD1D	CYBER 205		
GRID	65 × 65	129 × 129	257 × 257
SOLVE	0.094 (1.6)	0.263 (2.3)	0.831 (2.7)
TOTAL	0.181 (3.1)	0.469 (4.6)	1.442 (5.6)
CYCLE	0.012 (3.1)	0.034 (4.3)	0.108 (5.1)

Table 8.3. CPU-times in seconds of the program MGD1D run in vector-mode

MGEOZS	CYBER 170	CRAY 1		CYBER 205		
LEVEL	6	6	7	6	7	8
GRID	65 × 65	65 × 65	129 × 129	65 × 65	129 × 129	257 × 257
CONV	2.3E-1	2.3E-1	2.2E-1	2.3E-1	2.2E-1	2.02E-1
RAP	0.188	0.089	0.315	0.085	0.319	1.240
DECOMPZ	0.012	0.006	0.023	0.011	0.044	0.175
ZEBRA	0.333	0.135	0.511	0.148	0.585	2.309
RESIDU	0.106	0.049	0.191	0.045	0.180	0.733
PROLON	0.058	0.035	0.131	0.027	0.100	0.390
RESTRI	0.030	0.013	0.049	0.010	0.037	0.142
VL2NOR	0.018	0.013	0.048	0.008	0.033	0.128
TOTAL	0.797	0.348	1.286	0.353	1.333	5.216
CYCLE	0.053	0.023	0.088	0.023	0.090	0.357

Table 8.4. CPU-times (in seconds) of the program MGEOZS for problems with different sizes, run in scalar-mode on different machines.

MGEOZV	CRAY 1		CYBER 205		
LEVEL	6	7	6	7	8
GRID	65 × 65	129 × 129	65 × 65	129 × 129	257 × 257
CONV	2.3E-1	2.2E-1	2.3E-1	2.2E-1	2.0E-1
RAP	0.033 (2.7)	0.085 (3.7)	0.022 (3.9)	0.054 (5.9)	0.151 (8.2)
DECOMPZ	0.006 (1.0)	0.023 (1.0)	0.010 (1.1)	0.040 (1.1)	0.158 (1.1)
ZEBRA	0.034 (4.0)	0.103 (5.0)	0.084 (1.8)	0.210 (2.8)	0.623 (3.7)
RESIDU	0.010 (4.9)	0.034 (5.6)	0.007 (6.4)	0.020 (9.0)	0.067 (10.9)
PROLON	0.008 (4.4)	0.022 (6.0)	0.013 (2.1)	0.032 (3.1)	0.093 (4.2)
RESTRI	0.004 (3.2)	0.009 (5.4)	0.009 (1.1)	0.022 (1.7)	0.059 (2.4)
VL2NOR	0.003 (4.3)	0.009 (5.3)	0.002 (4.0)	0.004 (8.3)	0.012 (10.7)
TOTAL	0.102 (3.4)	0.293 (4.4)	0.162 (2.2)	0.400 (3.3)	1.190 (4.4)
CYCLE	0.006 (3.8)	0.017 (5.2)	0.011 (2.1)	0.028 (3.2)	0.084 (4.3)

Table 8.5. CPU-times (in seconds) of the program MGEOZV run in vector-mode. Between brackets the acceleration by vectorization.

We see that certain parts of the algorithms benefit greatly from vectorization viz. CTUMV and CTUPF (a factor 25-30 on CYBER 205, a factor 8-9 on CRAY 1). Other parts vectorize also well: VL2NOR, RAP, DECOMP, PROLON, RESIDU (on CRAY 1 also RESTRI and ZEBRA, in which vectoroperations with stride 2 occur). Other parts hardly benefit because of their recursive structure (DECOMPZ and SOLVE). If we give up portability, SOLVE can be speeded up on the CYBER 205 by use of the STACKLIB library.

9. CONCLUSIONS

Implementation of general-purpose multigrid solvers on vectorcomputers is feasible. Efficient programs in portable FORTRAN are now available for variable coefficient elliptic problems in a rectangular domain, discretized by a 7-point difference molecule. For the implementation implicit vectorization (auto-vectorization) can be used. The effect of vectorization (the factor by which the program accelerates) depends strongly on the size of the problem and, of course, on the algorithm used.
Our implementation with ILU-relaxation vectorized well on the CYBER 205 (factor 3.2-4.3) but slightly worse on a CRAY 1 (factor 3.1-3.5). The implementation with ZEBRA relaxation vectorized better on a CRAY 1 (3.4-4.4) and less well on a CYBER 205 (factor 2.2-4.4). The reduced vectorizability of MGEOZ on the CYBER 205 is due to the frequent occurance of vectors with stride unequal 1. For MGEOZV the CRAY 1 is faster than the CYBER 205; for MGD1V the CYBER 205 is faster for large problems.
We notice that for the determination of the efficiency of an algorithm on a vector-machine the usual measure of complexity - the operations count - appears to be completely irrelevant. Many more aspects have to be taken into account such as: are the computations arranged in small or large do-loops, are they recursive, vectorizable, how are the data stored etc..
The relative efficiency of the various algorithms depends - of course - on the complexity of the algorithms and on the rate of convergence. The complexity of MGEOZ is less, but generally MGD1 has a better convergence rate. Based on the present experiments we see that roughly one iteration with MGD1 takes twice the CPU-time of a MGEOZ iteration. On the other hand, in our Poisson testproblem, the empirical convergence rate of MGD1 is twice the rate of MGEOZ, so that both algorithms appear

equally efficient in this case. In general, the relative efficiency of MGD1 and MGEOZ depends on the difference problem to solve, the size of the system of equations and on the machine used.

ACKNOWLEDGEMENT

We are indebted to Mr. W. Lioen who constructed different versions of MGEOZ.

NOTE

The codes discussed in the paper can be obtained by sending a tape to Mr. A. van Deursen, Dept. of Mathematics and Informations, Delft University of Technology, Julianalaan 132, 2628 Delft, The Netherlands.

REFERENCES

[1] Barkai, D. and Brandt, A., Vectorized Multigrid Poisson Solver for the *CDC CYBER 205*, In: Procs. Int. MG-Conference, Copper Mountain, Colorado, April 6-8, 1983.

[2] Brandt, A., Multi-level adaptive solutions to boundary-value problems, Math. Comp. 31, 333-390, 1977.

[3] Hemker, P.W., On the comparison of line-Gauss-Seidel and ILU relaxation in multigrid algorithms, In: J.J.H. Miller (ed.), Computational and asymptotic methods for boundary and interior layers, pp. 269-277. Boole Press, Dublin, 1982.

[4] Hemker, P.W., Multigrid methods for problems with a small parameter, To appear in: Procs. Dundee. Conf. 1983, LNM, Springer-Verlag.

[5] Hemker, P.W., Wesseling, P. and De Zeeuw, P.M., Multigrid methods: development of fast solvers. In: Procs. Int. MG-Conference, Copper Mountain, Colorado, April 6-8, 1983.

[6] Kettler, R., Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. In: W. Hackbusch and U. Trottenberg (eds.), Multigrid methods. Proceedings, Köln-Porz, 1981. Lect. Notes in Math. 960, pp. 502-534, Springer-Verlag, Berlin etc., 1982.

[7] Stüben, K. and Trottenberg U., Multigrid methods: fundamental algorithms, model problem analysis and applications. In: W. Hackbusch and U. Trottenberg (eds.), Multigrid methods. Proceedings, Köln-Porz, 1981. Lect. Notes in Math. 960, pp. 1-176, Springer-Verlag, Berlin etc. 1982.

[8] Wesseling, P., Theoretical and practical aspects of a multigrid method. Siam J. Sci. Stat. Comp. 3, 387-407, 1982.

[9] Wesseling, P., A robust and efficient multigrid method. In: W. Hackbusch and U. Trottenberg (eds.), Multigrid methods. Proceedings, Köln-Porz, 1981. Lect. Notes in Math. 960, pp. 614-630, Springer-Verlag, Berlin etc., 1982.

[10] De Zeeuw, P.M., Lioen, W. and Hemker, P.W., Vectorized Multigrid Codes, To appear as Mathematical Centre report, Amsterdam, 1983.

DISCUSSION

Speaker: P. Hemker

Gorenflo: You intend your method to be robust. Then it should also work when in your equation convection is large compared to diffusion, that is in the nearly singular case. But I do not see how. Can you comment on this?

Hemker: We have to distinguish between nearly singular and singularly perturbed problems. In the first case (an eigenvalue almost equal zero) any iterative solver will run into problems. In the second case, for instance for the convection-diffusion equation, if properly discretized (e.g. by an upwind discretization), the Incomplete LU relaxation and even more the Incomplete Line LU relaxation have execellent smoothing properties.
References:
P. W. Hemker: On the comparision of line Gauss-Seidel and ILU relaxation in multigrid algorithms. In: J. J. H. Miller (Ed.). Computational and asymptotic methods for boundary and interior layers, pp. 269-277, Boole Press, Dublin, 1982.
P. W. Hemker. Multigrid methods for problems with a small parameter in the highest derivative. To appear in: Numerical Analysis, Proc. of the Dundee 1983 Conference, D. F. Griffiths (Ed). Springer-Verlag.

Brandt: You gave examples of computing times for your software. For some of these problems there are vectorized multi-grid algorithms which are ~ 100 times as fast. Considering relaxation methods, I should like to emphasise that when one wishes to solve only to truncation error level then simple direction free, high vectorizable point relaxation (e.g. red-black ordering) is quite sufficient and adequate when the equations are highly anisotropic, unless there is one dominant (i.e. throughout most of the domain) alignment between one particular characteristic direction and a specific grid direction. Such dominant alignment must be known to the user and can be communicated to the system, in which case the latter would use simple, highly vectorized line relaxation.

Hemker: As far as I know there is one program, which has been referenced in my paper (Barkai and Brandt), which is specially designed for the CYBER 205 and solves the Poisson equation up to truncation error with a speed of 0.36 μsec/grid point. This is about 30 times as fast as our program. When we take into account that our program is designed for variable coefficient problems (and hence all the matrix entries have to be used explicitly) and that it is written in portable FORTRAN (and hence is not bound to one particular machine), we see that its existence is sufficiently justified. Possibly/probably you are right. However, we want to make available a routine which yields the solution of the linear system up to an arbitrary small residual, that can be specified by the user.

Hockney: It would appear that quite different strategies are best for different machines. For example, on the CYBER 205 it may pay to iterate more in the fine mesh because of the greater length of vectors involved. Have you any theory for estimating the run time on vector processors a priori?

Hemker: Indeed we find that different variants of the MG-method are best for different machines. So we find that the variant with ZEBRA-relaxation runs better on a CRAY 1, whereas the variant with ILU-relaxation runs better on the CYBER 205. We have no general theory for estimating the run time on vector processors, neither do we search for the optimal variants (or strategies) for the various different machines.

Duff: Your package is designed for linear problems but unfortunately life is nonlinear. In the solution of a nonlinear problem would you advocate using your package on the linearized problem obtained, for example, through a Newton-type

solution scheme? How would this compare with solving the nonlinear problem on each grid as recommended by Achi Brandt.

Hemker: Both approaches are possible. It is claimed that the FAS multigrid approach can be more efficient. On the other hand there are arguments in favour of the Newton approach. In the latter case, if the iterative process does not converge - as sometimes happens for nonlinear problems - one may obtain better insight of the reason for the divergence.

Young: You talked about achieving 50% vectorization using the LU algorithm. Since the solution of a triangular system is not inherently vectorizable how were you able to achieve this?

Hemker: In the solution process for the LU-relaxation forward and backward substitution are performed with L (or U) being 4-diagonal triangular matrices. If we execute the substitution blockwise, the two diagonals in the off-main-diagonal blocks can be eliminated by vector-operations. Only the (non-vectorizable) solution of simple bidiagonal systems is left.

Reid: I am on the X3J3 Fortran committee and have a special interest on the incorporation of array facilities in the future Fortran (8X) standard. You seemed to have some trouble with differences between the scalar and vector versions of your routines. Do you think that facilities such as array or array section assignments could help you to avoid some of these problems?

Hemker: Yes, I think they may do so. But there is more to say about this. In my opinion a real programming language should in the first place provide a level of abstraction (abstraction from the machine architecture), in such a way that we can conveniently express our mathematical, algorithmical and "software engineering" thoughts in it. In this sense FORTRAN is a poor language. Therefore, for research work we prefer ALGOL 68. We use FORTRAN only because it is - unfortunately - the only language that is widely available for numerical computation. We would not be helped at all by FORTRAN extensions that would not be widely used and certainly not by extensions that would be implemented only on vector machines. In the present work we could write our program in a most elementary but portable FORTRAN and on the vector machines the compilers were clever enough to vectorize where it was necessary.

Reid: We are not designing extensions for a manufacturer or set of manufacturers, but for the standard itself and our hope and belief is that by the 1990s Fortran 8X will be as widely available as Fortran 66 was in the 1970s. Brian Smith, Laurie Schonfelder and I are the only representatives of the mathematical software community on the committee and our aim is to add to Fortran those facilities that have proved valuable in more powerful languages. One cannot "wipe the slate clean" because of the vast number of large working Fortran programs, whose owners are desperately anxious to continue to use them for ten or even twenty years. We need support and constructive criticism from those who are daily using other languages.

Rice: My experience of vectorizing compilers is that they are not very clever and many loops are not vectorized automatically. One must then rewrite the non-vectorized loops so that they can be recognized as vectorizable. This is rather inelegant. Have you encountered this problem?

Hemker: Yes, we have indeed. However, in our case, we were able to rewrite the programs in a portable FORTRAN in such a way that all the necessary vectorization was recognized by the compiler. In this sense the compiler was clever enough: we did not have to resort to special machine/compiler dependent vector language.

Brandt: It is at the algorithmic design stage that one thinks about vectorizability. It would be very good to have facilities to communicate these

design features directly to the machine. I would welcome features in extensions to Fortran to accommodate this.

Hockney: It is not only vectorization which is important but also how long the vectors are. This effect is characterized by the $n_{1/2}$ parameter of the computer, about which I will talk more later this week.

Hemker: This is perfectly true. In the given examples this also explains the behaviour of the speed-up factors on the CYBER 205, in particular for the prolongation and restriction routines.
Further, for the CYBER 205, it is important whether the vectors are contiguously stored or not.