44 Contributions November 1984 SUPERCOMPUTER

Multigrid algorithms run on supercomputers‡

Piet Hemker

CWI
Kruislaan 413
1098 SJ Amsterdam

At the Center for Mathematics and Computer Science (CWI), multigrid algorithms are used to solve elliptic partial differential equations. Emphasis is laid on portability of the implementation, allowing programs to run fast on a broad class of (super)computers.

The aim of research at the CWI on multigrid methods in elliptic partial differential equations is the construction of algorithms that efficiently yield a numerical solution of these problems. This kind of research is motivated by numerous applications, mainly in physics and in the engineering sciences. Except for a few very simple cases, it is impossible to find explicit mathematical expressions for the solutions, so that one has to rely on a numerical approximation to the solution.

By the very nature of partial differential equations, their solutions are continuous functions of several variables. In the numerical approach, these functions are approximated by only a finite set of numbers. Usually, these numbers represent the function values of the solution at an evenly spaced set of *gridpoints* in the domain of definition of the equation.

In practice, many problems appear in the form of an equation for the function $u(x,y)$, with (x,y) in a rectangle Ω. The form of this equation is

$$\left(\frac{\partial}{\partial x}\left(a_{11}\frac{\partial}{\partial x}+a_{12}\frac{\partial}{\partial y}\right)+\frac{\partial}{\partial y}\left(a_{21}\frac{\partial}{\partial x}+a_{22}\frac{\partial}{\partial y}\right)\right)u+b_1\frac{\partial u}{\partial x}+b_2\frac{\partial u}{\partial y}+cu=f, \qquad (1)$$

with additional conditions for $u(x,y)$ on the boundary of Ω. The coefficients a_{ij}, b_i, c and f are given functions of x and y. Much of the research on multigrid methods is restricted to this equation. The computer programs that have been recently con-

‡Adapted from *CWI Newsletter* 3 with kind permission.

structed at the CWI are almost all intended for equations of this type.

For those who are not familiar with elliptic partial differential equations a simple example is given by the Poisson equation:

$$(\frac{\partial}{\partial x})^2 u + (\frac{\partial}{\partial y})^2 u = 0 \quad \text{on } \Omega,$$

with $u(x,y)$ prescribed on the boundary of the 2-dimensional domain Ω. For this equation we can imagine the solution $z = u(x,y)$ as a surface in 3-dimensional space. On the boundary of Ω its position (x,y,z) is given and in the interior of Ω the surface behaves like a soap-film between the prescribed boundary values.

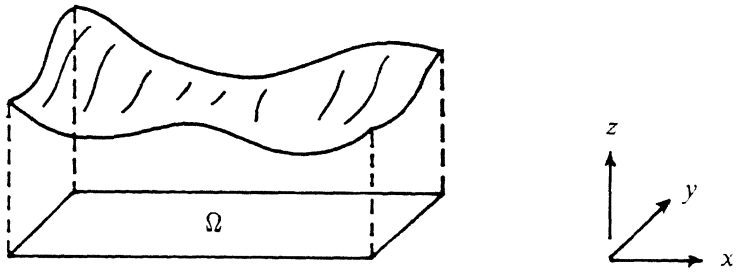

Figure 1.

This example illustrates some essential properties of elliptic partial diffential equations: boundary conditions are to be given all along the boundary and the solution in the interior is a smooth function.

The usual technique for finding an approximation to $u(x,y)$ is to replace equation

(1) by a set of N linear equations for N unknown values u_{ij} which are meant to represent the function values $u(x,y)$ where (x_i,y_j) is a gridpoint. All these gridpoints form a *grid* (or *net*) for the *discretization* of (1). The approximation u, for $u(x_i,y_j)$ becomes more accurate as the number of gridpoints N becomes larger. Therefore, it is often necessary to solve linear systems with N very large. N may be so large that – with conventional solution methods such as Gaussian elimination – it can take many days to solve these systems on a computer.

Therefore, the usual way to solve the large systems is by relaxation methods, i.e. iterative methods in which an initial guess of the solution is improved step by step. Well-known relaxation methods are Gauss–Seidel relaxation, SOR, zebra-relaxation, and Incomplete (Line) LU-decomposition-relaxation. Successful research in recent years has resulted in other, much faster converging, iteration methods such as ICCG (preconditional conjugate gradient methods, cf. [6]). A disadvantage of all these methods is that the rate of convergence of the iterations decreases for larger N.

A significant improvement in solving these (very large) systems of equations is found in the multigrid method. This is a technique which accelerates the convergence of the relaxation methods so that the rate is independent of N. This is done by introducing coarser grid discretizations (linear systems with $N := N/4, N/16$ etc.) and by combining relaxation for the large system with the (less laborious and faster converging) relaxation on the coarser grids. A good account of the multigrid method is found in [2].

For those to whom the basic idea of the multigrid method is new, a very short explanation is given. The principle of this method is based on three facts:

1
The simple relaxation methods such as Gauss–Seidel damp the rapidly varying components in the error much faster than the slowly varying components. In other words: they can be considered as efficient smoothers for the error rather than as reducers of the overall error.

2
The remaining (smooth) error components can be represented on coarser grids, where the number of gridpoints is much smaller. Consequently, the remaining error components can be reduced there much more efficiently.

3
On the coarse grid the solution is most efficiently reduced by a simple relaxation method and, again, by coarser grid corrections. Thus, a recursive procedure can be defined where on the coarsest grid the linear system to be solved has a very small number of unknowns.

It will be clear that the above principle is rather general and that many variants are possible. The idea can also be applied to other equations in which the original problem has continuous solutions. The idea can, for instance, be used for integral equations [4]. Attempts are even made to use the multigrid idea in cases where the linear systems do not originally stem from a continuous equation [5].

The convergence of the multigrid method for equation (1) depends on the coefficients in the equation, on the operators that take care of the interaction between the various grids, and on the relaxation method used. On the one hand the efficiency of a multigrid algorithm depends on this convergence and on the other hand on the amount of arithmetic operations in each iteration. In recent years, some research at the CWI was devoted to the selection of optimal efficient multigrid strategies for different equations (1), cf.[3,7].

It appears that for different classes of (1), different relaxation methods give optimal efficiency. For problems like the Poisson problem zebra- and ILU-relaxation are successful (zebra is slightly more efficient, but ILU performs better for a larger class of equations), while ILLU-relaxation is particulary suited for problems where the coefficients b_i dominate the coefficients a_{ij}.

Several implementations of multigrid algorithms have been constructed at the CWI. Besides a comprehensive program for experimental purposes written in Algol 68, a number of programs was written in Fortran with efficient execution in practical applications in mind. Two particular programs, developed in cooperation with the

Numerical Group of the Delft University of Technology, are called MGD1 (ILU relaxation) and MGD5 (ILLU).

In practice, the speed of these programs depends not only on the convergence rates or the number of arithmetic operations per iteration cycle, but also on the architecture of the computer used.

Here, the programmer has to decide whether his aim is to develop a program for a particular machine or to pursue an efficient program for a general class of computers. We decided in favor of the latter and wrote two versions, one aimed at the usual sequential (=scalar) computer and one at vector computers (Cray-1 or Cyber 205). In both cases we did not use features that are available only on one particular machine and we wrote the programs in a most elementary and portable Fortran. For the vector computers it meant that we used the auto-vectorization capabilities of the Fortran compilers.

Thus for ILU- and ILLU-relaxation we constructed a scalar version (MGD1S, MGD5S) and a vector version (MGD1V, MGD5V). For the scalar architecture the computing time for an iteration cycle is proportional to the number of gridpoints in the finest grid. For different machines the execution times are given in Table 1. From this, we see that for an equation like Poisson's equation (for which 3 iterations and a preparational phase corresponding to 3 iteration cycles are necessary) a linear system with $N = 257 \times 257 \approx 66000$ equations can be solved in less than a second.

Relaxation	MGD1 ILU	MGD5 ILLU
IBM 3081K	16.7	25.7
Cyber 170	15.4	24.9
Cray-1S	9.1	12.7
Cyber 205	8.1	11.1

Table 1. CPU-times for the scalar versions on scalar architecture in μsec/(cycle × meshpoint).

It is interesting to see to what extent the arithmetic operations in MGD1 and MGD5 can be arranged so as to make effective use of the vector architecture of the Cray-1 or the Cyber 205 (i.e. to what extent the algorithms are vectorizable). The acceleration factors of the vector programs run (if possible) in vector mode over the scalar programs (run in scalar mode) are given in Table 2.

	N	MGD1	MGD5
Cyber 170 (scalar mode)	65×65	0.86	0.95
Cray-1S (vector mode)	65×65	3.2	2.7
	129×129	3.6	2.9
	257×257	3.8	3.1
Cyber 205 (vector mode; two pipes)	65×65	3.2	2.2
	129×129	4.2	2.5
	257×257	4.8	2.6

Table 2. Acceleration factor of the vector version over the scalar version for the algorithms MGD1 and MGD5.

We see that vectorization of the MGD5 algorithm has more effect on the Cray than on the Cyber. The other algorithm, MGD1, is better vectorizable, especially on the Cyber 205. Now a Poisson type problem with $N = 257 \times 257$ is solved in 0.2 sec.

Other programs were made for zebra-relaxation. By its nature, this relaxation method seems better suited for vectorization than ILU- or ILLU-relaxation, and on the Cray-1 better acceleration factors were indeed found. However, to make this method efficient on the Cyber the data structures in the program had to be changed drastically. In the MGD-programs the data (u_{ij}) are stored in a natural way in a rectangular array, corresponding to the location of the gridpoints (x_i, y_j) in the rectangle Ω. In order to prevent the frequent use of strides > 1 in the zebra program (which is necessary for efficient vectorization on the Cyber), the data u_{ij} had to be

re-ordered by even and odd lines. In this way the program could be accelerated by a factor 7.3 on the Cyber. The same program runs without problems on the Cray.

For efficient implementation of an algorithm, the structure of the program has to be tuned to a great extent to the computer architecture. We are willing to do this as long as our programs remain portable.

A program can generally be made faster if one tunes the programming really to one particular machine and even more if one restricts its use to only one particular case of equation (1). Such a program has been constructed by Barkai and Brandt [1]. It solves (only) the Poisson equation on a Cyber 205. In this program, a checker-board relaxation is used and the data structures have been specially adapted for this relaxation on this particular computer. The result is a non-portable program which is extremely fast. In [1] it is mentioned that the Poisson equation with $N = 129 \times 129$ can be solved in 0.006 seconds.

At the CWI we do not plan to proceed in the direction of non-portable programs. At the moment, we are more interested in efficient algorithms for solving wider classes of equations. Besides our special interest in the solution of equations (1) of singular pertubation type, we are considering the implementation of an algorithm for (1) with (strongly) discontinuous coefficients.

Acknowledgements

I would like to thank Paul de Zeeuw who did most of the programming, and Walter Lioen who implemented the Fortran programs with zebra-relaxation. Furthermore, I would like to thank Drs. I.P. Jones and C.P. Thompson from AERE, Harwell (England), for their kind cooperation in running the programs on the Cray-1 and the IBM 3081K.

References

1
D. Barkai & A. Brandt, *Vectorized Multigrid Poisson Solver for the CDC Cyber 205,* J. Appl. Math. & Computat. **13**, 1983, 215-227.

2
W. Hackbusch & U. Trottenberg (Eds), *Multigrid Methods.* LNM 960, Springer Verlag, 1982.

3
P.W. Hemker, R. Kettler, P. Wesseling & P.M. de Zeeuw, *Multigrid methods: Development of Fast Solvers,* J. Appl. Math. & Computat. **13**, 1983, 311-326.

4
H. Schippers, *Multiple Grid Methods for Equations of the Second Kind with Applications in Fluid Mechanics,* Mathematical Centre Tract **163**, CWI, Amsterdam, 1983.

5
K. Stüben, *Algebraic Multigrid (AMG): Experiences and Comparisons,* J. Appl. Math. & Computat. **13**, 1983, 419-451.

6
H.A. van der Vorst, *Preconditioning by Incomplete Decompositions,* Ph. D. Thesis, University of Utrecht, 1982.

7
P.W. Hemker & and P.M. de Zeeuw, *Some implementations of multigrid linear system solvers,* CWI Report NM-R8401, 1984; to appear in *Proceedings Multigrid Conference,* Bristol, September 1983, IMA Publication, Academic Press, London, 1984.