INT. J. NUM. METH. HEAT FLUID FLOW, VOL. 4, 159–172 (1994)

VECTORIZED STRONGLY IMPLICIT SOLVING PROCEDURE FOR A SEVEN-DIAGONAL COEFFICIENT MATRIX

H.-J. LEISTER AND M. PERIĆ

Lehrstuhl für Strömungsmechanik, Universität Erlangen-Nürnberg, Cauerstr. 4, D-8520 Erlangen, Germany

ABSTRACT

The paper presents an extension of Stone's[1] strongly implicit procedure for solving linear equation systems resulting from the discretization of partial differential equations to three-dimensional problems. The solver is applicable to seven-diagonal coefficient matrices, as are obtained when central-difference approximations are used for discretization. The algorithm is implemented in a way which allows vector processing on modern supercomputers, in spite of its recursive structure. Other solvers, using incomplete lower-upper decomposition (ILU), can be vectorized in the same way. Test calculations show solver performance of about 150 Mflops on CRAY-YMP and over 200 Mflops on FUJITSU-VP200 computers. A listing of the FORTRAN code is provided.

KEY WORDS Implicit matrix solver Vectorization ILU decomposition

INTRODUCTION

Solution of large linear equation systems uses most of computing time in computational fluid dynamics (CFD) codes. Due to the non-linearity and coupling of partial differential equations, outer iterations must be used to update the coefficients and source terms in the linear equation systems. These are also solved iteratively, and iterations within the solver are called *inner iterations*. Most CFD codes solve for the velocity components and pressure or pressure correction in a sequential way. In order to enforce the continuity requirement, the pressure or pressure correction equation has to be solved to a tighter tolerance than other equations for each outer iteration. This equation is of Poisson type in case of incompressible flows and usually has Neumann boundary conditions on all boundaries. The efficiency of the CFD code strongly depends on the efficiency of the solver for the linear equation systems.

One of the most popular iterative solvers for CFD is the Stone's strongly implicit (SIP) solver[1]. Stone derived it for two-dimensional partial differential equations, but it can easily be extended to three-dimensional (3-D) problems. It has proven significantly more efficient than successive overrelaxation (SOR), alternate direction implicit (ADI) or incomplete lower-upper decomposition (ILU) solvers, and it requires substantially less computing operations per inner iteration than conjugate gradient (CG) methods, especially for non-symmetric matrices. SIP solver has also been used as a preconditioner in a CG solver, where it showed improved performance compared with standard incomplete Cholesky (IC) or ILU preconditioners[2]. However, the ILU-based solvers (and therefore also SIP solver and CG solvers which use ILU

9061–5539/94/020159–14$2.00
© 1994 Pineridge Press Ltd

Received February 1993
Revised September 1993

or similar methods as preconditioners) have the disadvantage that they use a recursive algorithm and are, therefore, not vectorizable or parallelizable in a straightforward manner.

In this paper we present a SIP solver for 3-D problems which is vectorized to a large degree, and which can easily be implemented in any 3-D, CFD code. The performance of the solver is demonstrated for several test problems and vector computers. A FORTRAN code with a description of the main variables is given in the appendix. Other solvers based on ILU can be vectorized in the same way.

DESCRIPTION OF ALGORITHM

Linear algebraic equation systems resulting from the discretisation of 3-D transport equations have the general form:

$$[A]\{\Phi\} = \{Q\} \tag{1}$$

where $[A]$ is an $N \times N$ square coefficient matrix (with N being the number of grid nodes), $\{\Phi\}$ is the vector matrix of the nodal variable values ordered in a certain way and $\{Q\}$ is the corresponding vector matrix of source terms. We consider here the case of central difference approximations and a structured grid with N_i nodes in ith coordinate direction ($N = N_x \cdot N_y \cdot N_z$). The ordering of nodes is such that surfaces $z =$ const. (constant index K) are stacked one above another, and within one surface index J increases first (y-direction), then index I (x-direction). The one-dimensional storage index of the variable Φ, ijk, is calculated from the three-dimensional grid indices i, j and k as follows:

$$ijk = (k-1)\cdot NIJ + (i-1)\cdot NJ + j$$

with

$$i = 1, \dots, NI; \qquad j = 1, \dots, NJ; \qquad k = 1, \dots, NK \tag{2}$$
$$NIJ = NI \cdot NJ$$

where NI, NJ and NK denote N_x, N_y and N_z, respectively. Other ordering of nodes is also possible. For this choice the matrix $[A]$ has the structure shown in *Figure 1*. It has non-zero elements only on seven diagonals. The iterative solution of (1) requires that an iteration matrix $[M]$ be chosen, and the process of solution proceeds as follows:

$$[M]\{\Phi^{n+1}\} = \{Q\} - [A - M]\{\Phi^n\}$$

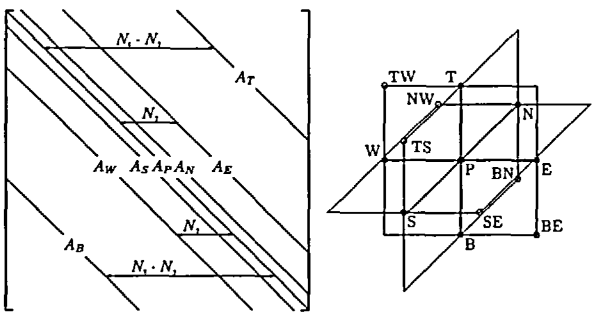

Figure 1 Structure of the coefficient matrix $[A]$ (left), and computational molecule (full symbols, right)

or

$$[M]\{\delta^m\} = \{R^m\} \tag{3}$$

where $\{\delta^m\}$ is the change in variable Φ from iteration m to $m+1$ and $\{R^m\}$ is the residual after the mth iteration, $\{R^m\} = \{Q\} - [A]\{\Phi^m\}$. For an iterative solution method to be efficient, the matrix $[M]$ should be as close to $[A]$ as possible. The SIP method of Stone[1] uses a product of two triangular matrices $[L]$ and $[U]$ (for lower and upper) as the iteration matrix, i.e.:

$$[M] = [L]\cdot[U] = [A] + [N] \tag{4}$$

The standard ILU-decomposition choses patrices $[L]$ and $[U]$ such that they have non-zero elements only on those diagonals where matrix $[A]$ has non-zero elements, and requires that the elements of $[M]$ are equal to the corresponding elements of $[A]$. This method, however, does not converge rapidly, since the resulting matrix $[N]$ is not small, i.e.:

$$[N]\{\Phi\} \neq \{0\} \tag{5}$$

Stone has realized that the solutions of partial differential equations are usually smooth functions of space and suggested a method of selecting the elements of $[L]$ and $[U]$ with the same sparsity as in ILU, but minimizing (5). The matrix $[M]$ has six additional diagonals with non-zero elepents, compared with matrix $[A]$, cf. *Figure 2*. If we set the elements on the main diagonal of $[U]$ arbitrarily to unity, the elements of $[M]$ at node P, denoted by subscript ijk, can be expressed in terms of elements of $[L]$ and $[U]$ as follows:

$$\begin{aligned}
M_{B,ijk} &= L_{B,ijk} \\
\mathbf{M}_{\mathbf{BN},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{B},\mathbf{ijk}}\mathbf{U}_{\mathbf{N},\mathbf{ijk}-\mathbf{NIJ}} \\
\mathbf{M}_{\mathbf{BE},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{B},\mathbf{ijk}}\mathbf{U}_{\mathbf{E},\mathbf{ijk}-\mathbf{NIJ}} \\
M_{W,ijk} &= L_{W,ijk} \\
\mathbf{M}_{\mathbf{NW},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{W},\mathbf{ijk}-\mathbf{NJ}} \\
M_{S,ijk} &= L_{S,ijk} \\
M_{P,ijk} &= L_{B,ijk}U_{T,ijk-NIJ} + L_{W,ijk}U_{E,ijk} + L_{S,ijk}U_{N,ijk} + L_{P,ijk} \\
M_{N,ijk} &= L_{P,ijk}U_{N,ijk} \\
\mathbf{M}_{\mathbf{SE},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{S},\mathbf{ijk}}\mathbf{U}_{\mathbf{E},\mathbf{ijk}-\mathbf{1}} \\
M_{E,ijk} &= L_{P,ijk}U_{E,ijk} \\
\mathbf{M}_{\mathbf{TW},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{W},\mathbf{ijk}}\mathbf{U}_{\mathbf{T},\mathbf{ijk}-\mathbf{NJ}} \\
\mathbf{M}_{\mathbf{TS},\mathbf{ijk}} &= \mathbf{L}_{\mathbf{S},\mathbf{ijk}}\mathbf{U}_{\mathbf{T},\mathbf{ijk}-\mathbf{1}} \\
M_{T,ijk} &= L_{P,ijk}U_{T,ijk}
\end{aligned} \tag{6}$$

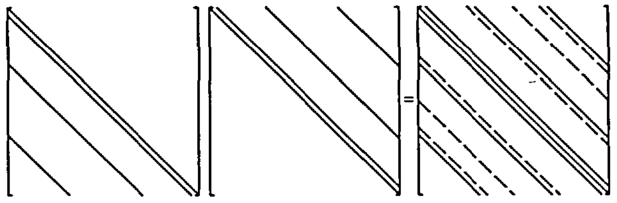

Figure 2 Structure of the iteration matrix $[L][U] = [M]$; dashed lines denote diagonals corresponding to nodes (from left to right) BN, BE, NW, SE, TW and TS, cf. *Figure 1*

The bold face printed elements correspond to nodes marked with open symbols in *Figure 1.* In the standard ILU method, the matrix $[N]$ has the six diagonals corresponding to the bold face printed elements in (6), while the other elements of $[M]$ are set equal to the corresponding elements of $[A]$. From these relations, the elements of $[L]$ and $[U]$ can be uniquely calculated. However, as noted before, $[N]\{\Phi\}$ is not zero and convergence is only slightly better than in ADI methods.

Stone[1] recognized that, by allowing $[N]$ to have, in addition, non-zero elements on the diagonals of $[A]$, and by using the smoothness property of the solutions of partial differential equations, faster convergence can be achieved. The idea was to provide a contribution in $[N]\{\Phi\}$ from the elements of the diagonals found in $[A]$ such that the contribution of the six diagonals not present in $[A]$ is partially cancelled, leading to:

$$[N]\{\Phi\} \approx \{0\} \tag{7}$$

The above equation can be written for a single node as follows:

$$\begin{aligned} &N_B\Phi_B + N_T\Phi_T + N_S\Phi_S + N_N\Phi_N + N_W\Phi_W + N_E\Phi_E + N_P\Phi_P \\ &\quad + M_{BN}\Phi_{BN} + M_{BE}\Phi_{BE} + M_{NW}\Phi_{NW} + M_{SE}\Phi_{SE} + M_{TW}\Phi_{TW} + M_{TS}\Phi_{TS} \approx 0 \end{aligned} \tag{8}$$

where the identity of elements of $[N]$ and $[M]$ at the diagonals not present in $[A]$ is assumed. The problem is now to define the elements N_B, N_T, N_S, N_N, N_E, N_W and N_P, such that the above equation is satisfied, but without introducing any additional unknowns. The Stone's idea was to write approximations for Φ_{BN}, Φ_{BE}, Φ_{NW}, Φ_{ES}, Φ_{TW}, Φ_{TS} in terms of 'principal' nodal values Φ_P, Φ_E, Φ_W, Φ_N, Φ_S, Φ_T and Φ_B of the form:

$$\begin{aligned} &\Phi_{BN} \approx \alpha(\Phi_B + \Phi_N - \Phi_P) \\ &\Phi_{NW} \approx \alpha(\Phi_N + \Phi_W + \Phi_P) \quad \text{etc.} \end{aligned} \tag{9}$$

where $0 \leqslant \alpha \leqslant 1$. From the assumption that (cf. (9))

$$\begin{aligned} M_{BN}[\Phi_{BN} - \alpha(\Phi_B + \Phi_N - \Phi_P)] &\approx 0 \\ M_{NW}[\Phi_{NW} - \alpha(\Phi_N + \Phi_W - \Phi_P)] &\approx 0 \quad \text{etc.} \end{aligned} \tag{10}$$

the coefficients N_B, N_T, N_S, N_N, N_W, N_E and N_P can be expressed through the bold face printed elements of $[M]$ of (6):

$$\begin{aligned} N_B &= -\alpha(M_{BN} + M_{BE}) \\ N_T &= -\alpha(M_{TW} + M_{TS}) \\ N_S &= -\alpha(M_{SE} + M_{TS}) \\ N_N &= -\alpha(M_{BN} + M_{NW}) \\ N_W &= -\alpha(M_{NW} + M_{TW}) \\ N_E &= -\alpha(M_{SE} + M_{BE}) \\ N_P &= \alpha(M_{BN} + M_{BE} + M_{NW} + M_{SE} + M_{TW} + M_{TS}) \end{aligned} \tag{11}$$

If approximations (9) are accurate, then (8) will hold when above expressions are used. By requiring now that $[M] = [A] + [N]$ and using expressions (6) and (11), the following equations for the calculation of the elements of [*L*] and [*U*] at *ijk*th node are obtained:

$$\begin{aligned}
L_{B,ijk} &= \frac{A_{B,ijk}}{[1+\alpha(U_{N,ijk-NIJ}+U_{E,ijk-NIJ})]}\\
L_{W,ijk} &= \frac{A_{W,ijk}}{[1+\alpha(U_{N,ijk-NJ}+U_{T,ijk-NJ})]}\\
L_{S,ijk} &= \frac{A_{S,ijk}}{[1+\alpha(U_{E,ijk-1}+U_{T,ijk-1})]}\\
H_1 &= \alpha(L_{B,ijk}U_{N,ijk-NIJ}+L_{W,ijk}U_{N,ijk-NJ})\\
H_2 &= \alpha(L_{B,ijk}U_{E,ijk-NIJ}+L_{S,ijk}U_{E,ijk-1})\\
H_3 &= \alpha(L_{W,ijk}U_{T,ijk-NJ}+L_{S,ijk}U_{T,ijk-1})\\
L_{P,ijk} &= A_{P,ijk}+H_1+H_2+H_3-L_{B,ijk}U_{T,ijk-NIJ}-L_{W,ijk}U_{E,ijk-NJ}-L_{S,ijk}U_{N,ijk-1}\\
U_{N,ijk} &= (A_{N,ijk}-H_1)/L_{P,ijk}\\
U_{E,ijk} &= (A_{E,ijk}-H_2)/L_{P,ijk}\\
U_{T,ijk} &= (A_{T,ijk}-H_3)/L_{P,ijk}
\end{aligned} \tag{12}$$

The above equations must be solved in the order written. The elements of [*L*] and [*U*] corresponding to the boundary nodes are assumed to be zero.

Since the iteration matrix [*M*] is made of the product of triangular matrices [*L*] and [*U*], solution of (3) is very cost effective. We first calculate an auxiliary vector matrix $\{V\}$ as:

$$\{V^m\} = [L^{-1}]\{R^m\} \tag{13}$$

and then the increment vector matrix $\{\delta\}$:

$$\{\delta^m\} = [U^{-1}]\{V^m\} \tag{14}$$

These two equations are solved easily by forward and backward substitution as follows:

$$V^m_{ijk} = (R^m_{ijk} - L_{B,ijk}V^m_{ijk-NIJ} - L_{W,ijk}V^m_{ijk-NJ} - L_{S,ijk}V^m_{ijk-1})/L_{P,ijk} \tag{15}$$

$$\delta^m_{ijk} = (V^m_{ijk} - U_{N,ijk}\delta^m_{ijk+1} - U_{E,ijk}\delta^m_{ijk+NJ} - U_{T,ijk}\delta^m_{ijk+NIJ}) \tag{16}$$

where

$$\begin{aligned}
R^m_{ijk} = Q_{ijk} &- A_{P,ijk}\Phi^m_{ijk} - A_{E,ijk}\Phi^m_{ijk+NJ} - A_{W,ijk}\Phi^m_{ijk-NJ} - A_{N,ijk}\Phi^m_{ijk+1}\\
&- A_{S,ijk}\Phi^m_{ijk-1} - A_{T,ijk}\Phi_{ijk+NIJ} - A_{B,ijk}\Phi^m_{ijk-NIJ}
\end{aligned} \tag{17}$$

is the residual at *m*th iteration at node *ijk*. For fixed α the calculation of elements of [*L*] and [*U*] matrices, (12), needs to be performed only once. For subsequent iterations, only (17), (15) and (16) need to be solved. Recalculation of (12) with different α reduces the number of iterations when an accurate solution is required, cf. Stone[1]. However, in CFD applications the linear equation systems are not solved exactly, since the matrix elements need to be updated due to non-linearity and coupling of equations. Since the convergence of SIP solver is rather rapid at the beginning, constant values of α are usually used.

PERFORMANCE OF THE SOLVER

When the equations for the velocity components, pressure or pressure correction, temperature, etc., are solved sequentially as is the case in SIMPLE-like algorithms[3], the linear equation systems need not be solved accurately for given coefficient and source matrices, $[A]$ and $\{Q\}$. Reduction of residual levels by one order of magnitude is usually sufficient before the coefficients and source terms are updated. For convection/diffusion equations, with Dirichlet boundary conditions on at least one boundary, one or two iterations are usually sufficient. The pressure or pressure correction equation for incompressible flows usually has Neumann boundary conditions on all boundaries and converges slowly; it usually requires 5–10 SIP iterations. The performance of the SIP-solver in a typical fluid flow and heat transfer problem is demonstrated for a test case presented in *Figure 3*. It shows a circular cylinder lying on its side, with left base cold and right base hot. Calculations were performed at Rayleigh numbers of 1000 and 10000 on a grid with $32 \times 24 \times 48$ control volumes. Velocity vectors for the middle circular cross section and both velocity vectors and isotherms for the vertical axial cross section and $Ra = 10000$ are also shown in *Figure 3*. The vertical axial cross-section shows flow and isotherms patterns similar to those of two-dimensional buoyancy-driven cavities, cf. Hortmann *et al.*[4]. However the circular cross-section reveals a very strong secondary flow with four counter-rotating eddies. In *Figure 4* the change of the sum of absolute residuals with iterations is shown for the U- and p'-equation in the test flow problem. The level of residuals is reduced by more than two orders of magnitude in 3 iterations for U and in 8 iterations for p', for both Rayleigh numbers and a

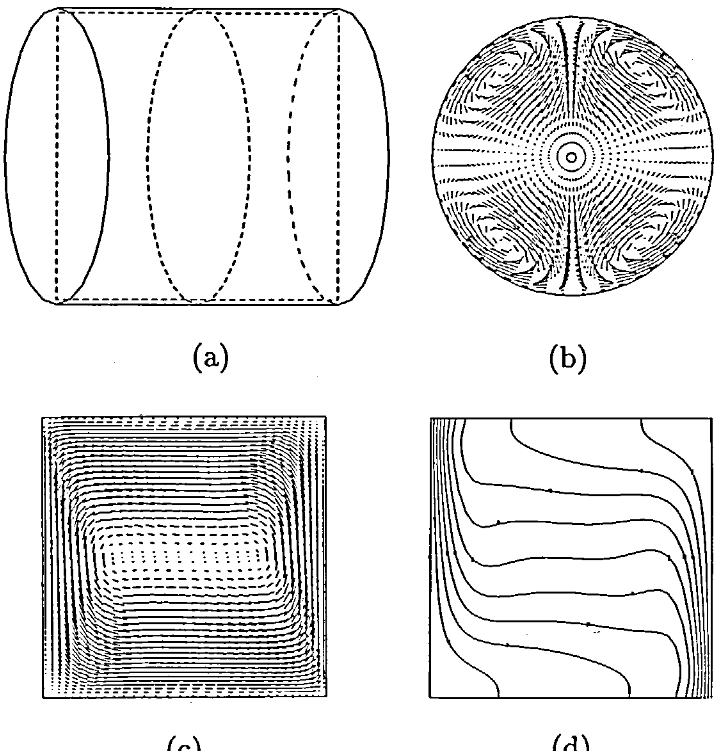

Figure 3 Flow problem: geometry (a), velocity vectors in the radial (b) and axial (c) cross-section and isotherms in the axial cross-section (d)

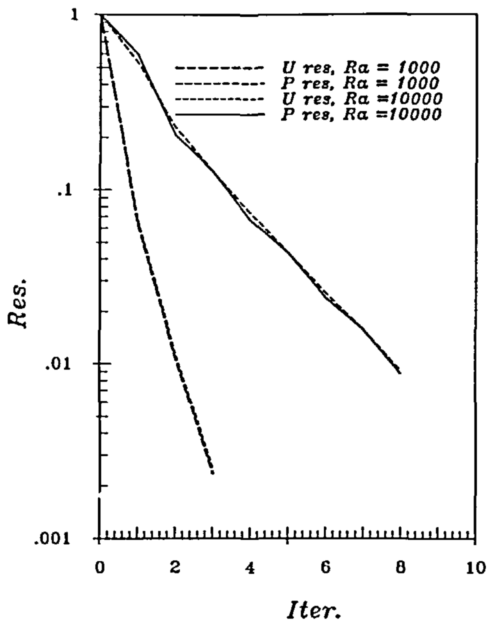

Figure 4 Residuals vs. iterations for the flow problem

given grid. The number of required iterations increases with grid refinement. However, linear equation systems – as noted above – need not be accurately solved, so reduction of residual levels by one order of magnitude usually suffices; in the present case one inner iteration for U, V and T and 4 for p' were found to be optimum.

In order to investigate the dependence of the convergence behaviour on the parameter α, a test problem with a well known analytical solution were chosen. Laplace equation:

$$\frac{\partial^2\Phi}{\partial x^2}+\frac{\partial^2\Phi}{\partial y^2}+\frac{\partial^2\Phi}{\partial z^2}=0 \tag{18}$$

was solved using finite difference discretization on uniform grids and central difference approximation of the second derivatives. The boundary conditions are of Dirichlet type:

$$\Phi(0,x,z)=\Phi(x,0,z)=\Phi(x,y,0)=0$$

$$\Phi(1,y,z)=yz, \qquad \Phi(x,1,z)=xz, \qquad \Phi(x,y,1)=xy$$

The analytical solution of this problem is $\Phi(x,y,z)=xyz$. Calculations were performed with different values of α in the range $0\leqslant\alpha<1$ and for various grids. The iterative procedure was stopped when the sum of absolute residuals had fallen by four orders of magnitude. The performance of SIP solver and its dependence on α is presented in *Figure 5*. The finer the grid is, the more pronounced is the dependence of convergence rate on α. For a grid with $11\times11\times11$ nodes, the solver needs about 2.5 times more iterations for $\alpha=0$ (which corresponds to the standard ILU) than with $\alpha=0.98$. For a $21\times21\times21$ nodes grid the difference is already more than factor 6, and for $41\times41\times41$ nodes the factor is over 20. This indicates the effectiveness of Stone's approximation and the superiority of SIP over standard ILU. The facts that the optimum value of α is problem dependent and that the efficiency strongly depends on the value of α are the undesirable features of the SIP solver. However, values of $\alpha\approx(0.92\text{–}0.94)$ were found to give results close to the optimum ones for a wide range of problems; these values are suggested for general use.

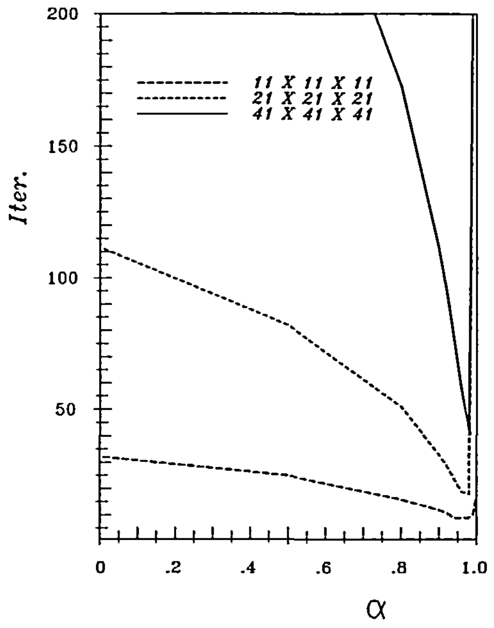

Figure 5 Dependence of convergence rate on α, Laplace problem

For the same test problem and a 41 × 41 × 41 nodes grid, ADI solver[5] is about 10 times slower than SIP, while the symmetric CG method with IC preconditioning[6] needs about the same time (but less iterations) to reach a solution of the same accuracy as the SIP. However, SIP is applicable to non-symmetric matrices as well, while the above CG method is not. A non-symmetric version of the CG method requires twice as many computing operations per iteration than the symmetric version, and about three times as many as the SIP solver. Since the convection-diffusion problems from CFD result in non-symmetric matrices, SIP is obviously a good choice when structured grids are used. The CG solvers, on the other hand, can be used on unstructured meshes as well. As already noted, SIP solver can also be used as a preconditioner in a CG solver instead of IC and ILU methods.

VECTORIZATION OF THE SOLVER

The algorithm as described by (12), (15) and (16), is strongly recursive. A recursive structure corresponds to data dependencies which usually prevent a programmer from taking advantage of vector computers. Analysing the data dependencies in (12), however, shows that the recursive structure of SIP can partially be resolved. Let us define field variable sets:

$$\mathscr{P}^l_\Phi = \{\Phi_{ijk} \text{ with } i+j+k=l\} \tag{19}$$

Superscript l describes plane cuts through the computational space as indicated in *Figure 6*. From (12) and (15) it can be seen that for the calculation of the elements of $[L]$, $[U]$ and $\{V\}$ corresponding to $\mathscr{P}^l_\Phi$, field variables with index values $ijk-1$, $ijk-NJ$ and $ijk-NIJ$ are necessary. For these field variables it follows from (2) that $i+j+k=l-1$ is valid and hence they are members of $\mathscr{P}^{l-1}_\Phi$. This provides for an effective vectorization, because all elements of $\mathscr{P}^l_\Phi$ can be treated as a vector. The ILU decomposition and forward substitution ((12) and (15))

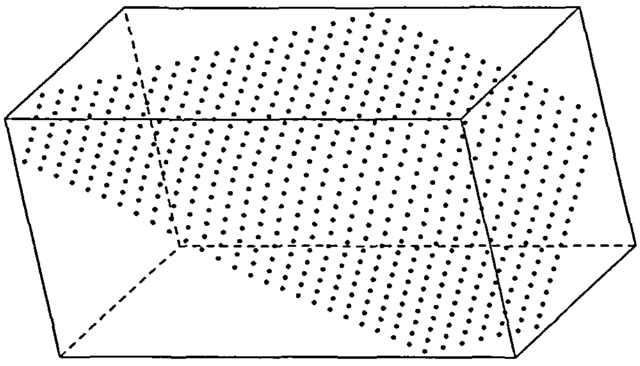

Figure 6 Planes of $i + j + k = \text{const.}$

start with the inner point having the smallest l, i.e. $i = 2, j = 2, k = 2$ or $l_{\min} = 6$, and end with the inner point having the largest value of l, i.e. $l_{\max} = NI + NJ + NK - 3$, i.e.:

$$\mathscr{P}_\Phi^6 \Rightarrow \mathscr{P}_\Phi^7 \Rightarrow \mathscr{P}_\Phi^8 \Rightarrow \cdots \Rightarrow \mathscr{P}_\Phi^{NI+NJ+NK-4} \Rightarrow \mathscr{P}_\Phi^{NI+NJ+NK-3} \tag{20}$$

In the backward substitution (16) the opposite structure is necessary:

$$\mathscr{P}_\Phi^{NI+NJ+NK-3} \Rightarrow \mathscr{P}_\Phi^{NI+NJ+NK-4} \Rightarrow \cdots \Rightarrow \mathscr{P}_\Phi^8 \Rightarrow \mathscr{P}_\Phi^7 \Rightarrow \mathscr{P}_\Phi^6 \tag{21}$$

In order to be able to perform operations from (12), (15) and (16) in vector mode, we have to create $l_{\max} - l_{\min} + 1$ vectors $\{\Phi_l\}$ and process them in a vector loop which runs from $l_{\min}$ to $l_{\max}$ or in the reverse order. One could store the field variables in this fashion and have the whole code structured in the above sense, as done by Yoon and Kwak[7] in their LU-solver. However, these vectors are of variable length and are usually not processed as efficiently as when the vectors contain data from one coordinate surface, e.g. $k = \text{const}$. We therefore prefer to store variable values as described by (2), and process all non-recursive loops (such as (17)) for the calculation of residuals, or calculating elements of $[A]$ and $\{Q\}$) by simply cutting the vector matrices $\{\Phi\}$ in vectors $\{\Phi_k\}$, $k = 1, \ldots, N_k$. Processing the recursive loops in two integer arrays have to be predefined before calling SIP for the first time. One – $mip(l)$ – defines the number of nodes belonging to surfaces $\mathscr{P}_\Phi^l$ where l runs from $l_{\min}$ to $l_{\max}$. The other – $ijksip(n)$ – identifies the ijk storage indices of nodes within one SIP-vector, where n runs from $mip(l) + 1$ to $mip(l + 1)$. The basic idea is that for all $l = l_{\min}, \ldots, l_{\max}$ the possible k-index range $\mathscr{K}_l$ is determined. In the next loop variable k runs over $\mathscr{K}_l$. Now the dependence of the i-index range $\mathscr{I}_l$ on l and k is analysed. The third loop runs then over $\mathscr{I}_l$ range of i. Since the index j is given by $l - k - i$, the correlation between the surface indices and the ijk storage indices can easily be calculated. Further details are given in the appendix and subroutine *VECIND*.

The performance of the vectorized version of SIP is analysed on both the flow test problem of *Figure 3* and on the Laplace equation model presented above. First the dependence of performance (measured in millions of floating point operation per second within the SIP-solver, Mflops) on grid fineness is studied. *Figure 7* shows performance of the vectorized SIP solver on CRAY-YMP for the flow and heat transfer problem of *Figure 3* for five levels of systematic grid refinement, starting from $4 \times 3 \times 6$ CV up to $64 \times 48 \times 96$ CV. Scalar performance is about 20 Mflops. The lower vector performance for coarser grids is due to short vectors; once the grid is fine enough so that the optimum vector length is reached, further grid refinement does not lead to improvement of performance. It should also be noted that the most unfavourable grid is the one with the same number of nodes in each direction, since the vector length of $\{\Phi_l\}$ varies all the time. If the number of nodes in one direction is significantly higher than in the other two

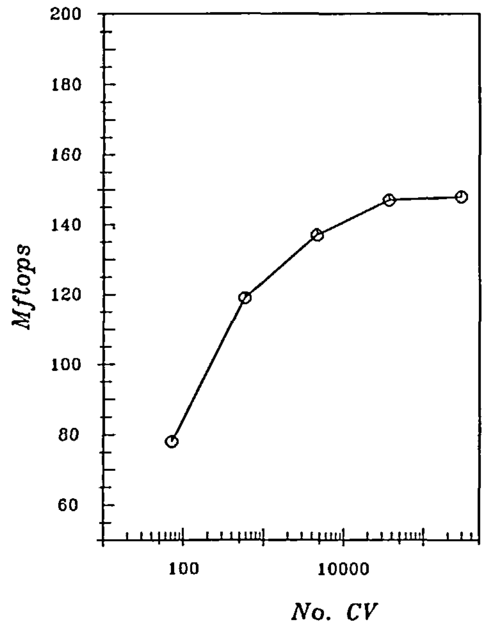

Figure 7 Performance of the vectorized SIP solver in Mflops vs. grid fineness, flow problem (scalar performance ≈20 Mflops)

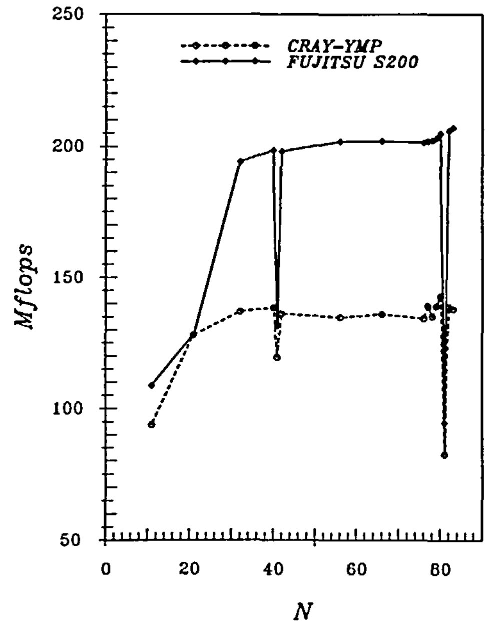

Figure 8 Performance of the vectorized SIP solver in Mflops vs. number of nodes in one direction, Laplace problem (scalar performance: on CRAY YMP ≈20 Mflops, on FUJITSU VP200 ≈15 Mflops)

directions, then there is a range of indices l where the length of vector $\{\Phi_l\}$ is constant, which is desirable for optimum vector processing.

Finally, performance of the vectorized SIP-solver for the Laplace problem is studied using uniform grids with the same number of nodes in each direction. Results of calculations on CRAY-YMP and FUJITSU VP200 are shown in *Figure 8*, where the speed in Mflops is plotted against the number of nodes in one direction N_i. Scalar performance on FUJITSU VP200 is about 15 Mflops. The grid is refined gradually, so that many more points are obtained on a curve which is otherwise similar to that of *Figure 7*. Surprisingly, on both computers there is a sudden drop of performance for $N_i = 41$ and $N_i = 81$. It is especially severe for $N_i = 81$, when the FUJITSU performance drops from over 200 Mflops to well below 100. This phenomenon has to do with internal memory access, which appears to be unfavourable only for certain numbers of grid nodes.

CONCLUSIONS

A strongly implicit solution method, after Stone[1], has been extended to 3-D problems and implemented in a CFD code using finite volume and finite difference discredization. In sequential solution methods for coupled equations like SIMPLE, one inner iteration in the SIP-solver per outer iteration (coefficient update) is sufficient for all equations except the pressure-correction equation, which needs 4–10 inner iterations to reduce the residual level by approximately one order of magnitude. The choice of the parameter α influences the rate of convergence, the effect becoming more pronounced as the grid is refined. The superiority of SIP over standard ILU solution method also increases with grid refinement. Values of α between 0.92 and 0.94 were found to produce nearly optimum results for a wide variety of flow problems.

In spite of the recursive structure of the solution algorithm, vectorization in SIP solver was achieved by avoiding data dependencies through indirect addressing and sweeping through the domain along diagonal planes. Vectorized CFD code reaches speeds of over 200 Mflops on CRAY-YMP computer in most routines, with SIP-solver performance of about 150 Mflops. The computing speed is thus increased through vectorization by a factor of 10 (on CRAY YMP) to 20 (on FUJITSU VP200). Due to indirect addressing and variable vector length, in some cases the performance of the SIP solver may drop suddenly for a specific number of grid nodes in a particular direction. The remedy in such a case is to add or to subtract one grid point in one coordinate direction. Other solvers, like ILU and many variants of CG solvers with preconditioners based on ILU or IC, can be vectorized using the same strategy. The usual data structure and vectorization used in the rest of a CFD code can be retained. Due to its numerical efficiency, fast convergence and low number of computing operations, vectorized SIP solver is recommended as an effective tool for solving linear equation systems resulting from discretization of partial differential equations.

REFERENCES

1 Stone, H. L. Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Num. Anal.*, **5**, 530–558 (1968)
2 Ferziger, J. H. Private communication, Stanford University (1992)
3 Perić, M. A finite volume method for the prediction of three-dimensional fluid flow in complex ducts, *PhD Thesis*, University of London (1985)
4 Hortmann, M., Perić, M. and Scheuerer, G. Finite volume multigrid prediction of laminar natural convection: bench-mark solutions, *Int. J. for Num. Meth. Fluids*, **11**, 189–207 (1990)
5 Peaceman, D. W. and Rachford, H. H. The numerical solution of parabolic and elliptic differential equations, *J. Soc. Ind. Appl. Math.*, **3**, 28–41 (1955)
6 Meijerink, J. A. and van der Vorst, H. A. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems, *J. Comp. Physics*, **44**, 134–155 (1981)

7 Yoon, S. and Kwak, D. Three-dimensional incompressible Navier–Stokes solver using lower-upper symmetric-Gauss-Seidel algorithm, *AIAA J.*, **29**, 874–875 (1991)

APPENDIX

In order to implement the present solver in a CFD code which uses one-dimensional storage described in (2), one needs to integrate the following two routines into the code: *VECIND* and *SIPVEC*. The following text describes the meaning of the main variables from the two routines.

The blank common block contains *NI*, *NJ* and *NK*, whose meaning was described in the text above. *NIM*, *NJM* and *NKM* correspond to $NI - 1$, $NJ - 1$ and $NK - 1$, respectively. *MAXIT* is the maximum allowed number of inner iterations. $LK(K)$ and $LI(I)$ are defined as (cf. (2)):

$$LK(K) = (K - 1) \cdot NIJ, \qquad LI(I) = (I - 1) \cdot NJ \tag{22}$$

Arrays $MIP(M)$ and $IJKSIP(IJK)$ were described above and are defined in the routine *VECIND*. *RESMAX* stands for the sum of absolute residuals normalized with its initial value which serves as convergence criterion. *ALFA* is the parameter α, cf. (9) and (12).

The common block *COEF* contains elements of the coefficient matrix $[A]$. The elements of the triangular matrix $[L]$ are named *BB*, *BS*, *BW* and *BP* and correspond to L_B, L_S, L_W and L_P from (12). *P*1, *P*2 and *P*3 correspond to H_1, H_2 and H_3 of (12). The elements of $[U]$ are named *BN*, *BE* and *BT* and correspond to U_N, U_E and U_T from (12). The elements of $[L]$ and $[U]$ corresponding to the boundary modes ($i = 1$ or NI; $j = 1$ or NJ; $k = 1$ or NK) are assumed equal to zero (default in the code).

The variable $RES(IJK)$ is used to store the residual value at the grid nodes ((17); loop no. 30). The residuals are then overwritten with the values of $\{V\}$, cf. (15) (loop no. 50), and $\{V\}$ is finally overwritten by $\{\delta\}$, cf. (15) (loop no. 60). *FI* stands for the dependent variable Φ; it is passed as an argument by the calling routine and may represent velocity components, temperature, pressure correction etc., $RESH(J)$ is used to store the sum of absolute residuals along the line $ij =$ const. (z-direction), in order to avoid a long scalar loop like loop no. 40, which calculates the total sum.

The instruction *CDIR$ IVDEP* is the CRAY-YMP directive to ignore vector dependencies, since otherwise apparently recursive loops would be processed in scalar mode. The instructions * *VOCL LOOP, SCALAR* and * *VOCL LOOP, VECTOR* are the directives of FUJITSU VP200 computer forcing scalar and vector processing of the following loop, respectively. * *VOCL LOOP, NOVREC* means on FUJITSU VP200 'no vector recurrences', which is analog to the above CRAY-YMP directive. In the code, however, all FUJITSU directives start with a 'C' in the first column and are therefore ignored by other computers (CRAY ignores FUJITSU directives and vice versa).

The routine *VECIND* needs to be called only once, before the first execution of the routine *VECSIP*. It runs in scalar mode, but due to the low number of operations and only one execution, it does not reduce the efficiency of the code. Alternatively, the arrays $MIP(M)$ and $IJKSIP(IJK)$ may be calculated while generating the grid or elsewhere and provided as input data.

FORTRAN CODE

```
C=================================================================
      SUBROUTINE VECIND
C=================================================================
C     Hans-Joerg Leister, LSTM Erlangen, 1991
      PARAMETER (NX=41,NY=41,NZ=41,NXY=NX*NY,NXYZ=NXY*NZ,NNN=NX+NY+NZ)
      COMMON NI,NJ,NK,NIM,NJM,NKM,MAXIT,LK(NZ),LI(NX),MIP(NNN),
     *       IJKSIP(NXYZ),NIJ,NIJK,N3D,RESMAX,ALFA
C
      N3D=NIM+NJM+NKM
      NIJ=NI*NJ
      NIJK=NIJ*NK
      MIP(6)=0
C
      DO 51 M=6,N3D
      IC1=0
      LM=MIP(M)
      NKMIN=MAX0(M-NIM-NJM,2)
      NKMAX=MIN0(M-4,NKM)
C
      DO 53 K=NKMIN,NKMAX
      NIMIN=MAX0(M-NJM-K,2)
      NIMAX=MIN0(M-K-2,NIM)
      IJKBE=(K-1)*NIJ+(NIMIN-1)*NJ+M-K-NIMIN
C
      DO 57 I=1,1+NIMAX-NIMIN
      IJKSIP(LM+IC1+I)= IJKBE+(I-1)*NJM
      WRITE(6,*) LM+IC1+I, IJKSIP(LM+IC1+I)
   57 CONTINUE
C
      IC1=IC1+1+NIMAX-NIMIN
   53 CONTINUE
C
      MIP(M+1)=MIP(M)+IC1
   51 CONTINUE
C
      RETURN
      END

C=================================================================
      SUBROUTINE SIPVEC(FI)
C=================================================================
C     Hans-Joerg Leister, LSTM Erlangen, 1991
      PARAMETER (NX=41,NY=41,NZ=41,NXY=NX*NY,NXYZ=NXY*NZ,NNN=NX+NY+NZ)
      COMMON NI,NJ,NK,NIM,NJM,NKM,MAXIT,LK(NZ),LI(NX),MIP(NNN),
     *       IJKSIP(NXYZ),NIJ,NIJK,N3D,RESMAX,ALFA
      COMMON /COEF/ AE(NXYZ),AW(NXYZ),AN(NXYZ),AS(NXYZ),AT(NXYZ),
     *       AB(NXYZ),AP(NXYZ),Q(NXYZ),T(NXYZ)
      DIMENSION FI(NXYZ),BB(NXYZ),BS(NXYZ),BW(NXYZ),BP(NXYZ),
     *       BE(NXYZ),BN(NXYZ),BT(NXYZ),RES(NXYZ),RESH(NXY)
C
C.....Calculation of Elements of [L] and [U]
C
CVOCL LOOP,SCALAR
      DO 10 M=6,N3D
      LM=MIP(M)
      IJEN= MIP(M+1)-MIP(M)
CVOCL LOOP,VECTOR
CVOCL LOOP,NOVREC
CDIR$    IVDEP
      DO 10 IJ=1,IJEN
      IJK=IJKSIP(LM+IJ)
      BB(IJK)=AB(IJK)/(1.+ALFA*(BN(IJK-NIJ)+BE(IJK-NIJ)))
      BW(IJK)=AW(IJK)/(1.+ALFA*(BN(IJK-NJ) +BT(IJK-NJ)))
      BS(IJK)=AS(IJK)/(1.+ALFA*(BE(IJK-1) + BT(IJK-1)))
       P1 =ALFA*(BB(IJK)*BN(IJK-NIJ)+BW(IJK)*BN(IJK-NJ))
       P2 =ALFA*(BB(IJK)*BE(IJK-NIJ)+BS(IJK)*BE(IJK-1))
       P3 =ALFA*(BW(IJK)*BT(IJK-NJ) +BS(IJK)*BT(IJK-1))
      BP(IJK)=1./(AP(IJK)+P1+P2+P3-BB(IJK)*BT(IJK-NIJ)-
     *        BW(IJK)*BE(IJK-NJ)-BS(IJK)*BN(IJK-1)+1.E-30)
```

```
      BN(IJK)=(AN(IJK)-P1)*BP(IJK)
      BE(IJK)=(AE(IJK)-P2)*BP(IJK)
      BT(IJK)=(AT(IJK)-P3)*BP(IJK)
   10 CONTINUE
C
C.....Calculation of Residuals
C
      DO 200 L=1,MAXIT
      RES1=0.0
      DO 20 IJ=NJ+1,NIJ-NJ
      RESH(IJ)=0.
   20 CONTINUE
C
      DO 30 K=2,NKM
      KK=LK(K)
      DO 30 IJ=NJ+1,NIJ-NJ
      IJK=KK+IJ
      RES(IJK)=Q(IJK)-AE(IJK)*FI(IJK+NJ)-AW(IJK)*FI(IJK-NJ)-AN(IJK)*
     *         FI(IJK+1)-AS(IJK)*FI(IJK-1)-AT(IJK)*FI(IJK+NIJ)-
     *         AB(IJK)*FI(IJK-NIJ)-AP(IJK)*FI(IJK)
      RESH(IJ)= RESH(IJ)+ABS(RES(IJK))
   30 CONTINUE
C
      DO 40 IJ=NJ+1,NIJ-NJ
      RES1=RES1+RESH(IJ)
   40 CONTINUE
C
      IF(L.EQ.1) RESNOR=RES1
      RSM=RES1/(RESNOR+1.E-30)
C
C.....Forward Substitution
C
CVOCL LOOP,SCALAR
      DO 50 M=6,N3D
      LM=MIP(M)
      IJEN= MIP(M+1)-MIP(M)
CVOCL LOOP,VECTOR
CVOCL LOOP,NOVREC
CDIR$    IVDEP
      DO 50 IJ=1,IJEN
      IJK=IJKSIP(LM+IJ)
      RES(IJK)=(RES(IJK)-BB(IJK)*RES(IJK-NIJ)-BW(IJK)*RES(IJK-NJ)-
     *          BS(IJK)*RES(IJK-1))*BP(IJK)
   50 CONTINUE
C
C.....Backward Substitution
C
      DO 60 M=N3D,6,-1
      LM=MIP(M)
      IJEN= MIP(M+1)-MIP(M)
CVOCL LOOP,VECTOR
CVOCL LOOP,NOVREC
CDIR$    IVDEP
      DO 60 IJ=1,IJEN
      IJK=IJKSIP(LM+IJ)
      RES(IJK)=RES(IJK)-BN(IJK)*RES(IJK+1)-BE(IJK)*RES(IJK+NJ)-
     *         BT(IJK)*RES(IJK+NIJ)
   60 CONTINUE
C
C.....Variable Update
C
      DO 70 IJK=NIJ+NJ+1,NIJK-NIJ-NJ
      FI(IJK) = FI(IJK) + RES(IJK)
   70 CONTINUE
C
C.....Convergence Check
C
      WRITE(6,*) '  ',L,' SWEEP, RES = ',RSM
      IF(RSM.LT.RESMAX) RETURN
  200 CONTINUE
      RETURN
      END
```