AIAA JOURNAL
Vol. 36, No. 4, April 1998

Fast Newton–Krylov Method for Unstructured Grids

Max Blanco* and David W. Zingg†
University of Toronto, Downsview, Ontario M3H 5T6, Canada

Three algorithms are presented and compared for the solution of the steady Euler equations on unstructured triangular grids. All are variations on Newton's method—one quasi- and two full-Newton schemes—and employ the BILU (n)-preconditioned generalized minimum residual method (GMRES) algorithm to solve the Jacobian matrix problem that arises at each iteration. The quasi-Newton scheme uses a first-order approximation to the Jacobian matrix with the standard GMRES implementation, in which matrix-vector products are formed in the usual explicit manner. The full-Newton schemes are distinguished by the implementation of GMRES: One employs the standard GMRES algorithm, and the other is matrix free using Fréchet derivatives. The matrix-free, full-Newton algorithm is shown to be the fastest of the three algorithms. Optimal preconditioning, reordering, and storage strategies for the matrix-free, full-Newton algorithm are presented. Register and cache performance issues are briefly discussed.

Introduction

COMPRESSIBLE flow solvers on unstructured grids have proven to be very effective for steady inviscid flows. The most promising approaches generally use either the multigrid technique or some approximate form of Newton's method to converge to a steady-state solution. The appeal of the latter is based on the potential for quadratic, or superlinear, asymptotic convergence. At each iteration of a Newton-like algorithm, a nonsymmetric system of linear equations must be solved. Iterative methods tend to be used for the solution of this system because direct solution is impractical for large problems, as shown by Venkatakrishnan.[1] Krylov subspace methods, such as the generalized minimum residual method (GMRES),[2] are often employed due to the nonsymmetric nature of the system. Because Krylov methods tend to fare poorly with ill-conditioned systems, a suitable preconditioner must be employed. Fortunately, the approximate solution obtained using iterative methods is sufficient to maintain superlinear convergence.[3]

The linearization of the residual function must be virtually exact to obtain the desired superlinear convergence. However, formation of the Jacobian matrix resulting from the exact linearization can require excessive amounts of storage. This requirement can be reduced by forming the matrix-vector products required by GMRES indirectly via the Fréchet derivative; the matrix-free procedure has no effect on the superlinear convergence of the resulting full-Newton algorithm. If a preconditioner based on an incomplete lower-upper factorization (ILU) is employed, a lower-order approximation to the Jacobian matrix can be used in its formation to reduce storage needs. Full-Newton algorithms using Krylov methods, implemented with either standard explicit matrix-vector products or indirectly formed matrix-free products, are described in Refs. 4–10. Another approach is to use a quasi-Newton method in which the exact Jacobian matrix is replaced by a lower-order approximation.[11–17] Although the number of Newton iterations to convergence may increase, the cost per GMRES iteration is decreased due to the improved conditioning of the matrix and the reduced operation count. Venkatakrishnan and Mavriplis[17] found this quasi-Newton strategy to be competitive with explicit multigrid solvers, albeit with increased storage requirements.

Other factors that can influence the efficiency of a Newton–Krylov algorithm include the type of preconditioner used and the ordering of the unknowns.[18] The choice of a preconditioner is dependent on the tradeoff between speed and memory. Many authors have used the block ILU factorization with no additional fill, BILU(0), using the natural four-by-four block size for the two-dimensional Euler equations. Other ILU variants include ILU(n) and BILU(n), using scalar- and block-based storage schemes, respectively, which allow fill entries based on a level-of-fill parameter n, and ILUT(p, t), which employs a threshold drop tolerance t to introduce a maximum of p fill entries in each factor.[19] Possible reordering algorithms include reverse Cuthill–McKee, one-way dissection, nested dissection, and quotient minimum degree.[20]

The goal of the present paper is to examine the tradeoffs inherent in the various Newton–GMRES implementations and to develop a fast, robust algorithm. In particular, we compare two full-Newton algorithms, using matrix-free and standard implementations of GMRES, and a quasi-Newton algorithm. The primary criterion is CPU usage. Results from an extensive parameter study of each solver are presented. Optimal strategies for the choice of GMRES Krylov subspace dimension and exit tolerance, BILU(n) fill level, reordering technique, and storage model are presented.

Euler Equations

The Euler equations, which model compressible inviscid fluid flows, can be derived by writing expressions for the conservation of mass, momentum, and energy. In integral form, for an arbitrary two-dimensional control volume Ω bounded by surface $\partial\Omega$, they are given by

$$\oint_{\partial\Omega} (\boldsymbol{F}\,\mathrm{d}y - \boldsymbol{G}\,\mathrm{d}x) = 0 \tag{1}$$

where

$$\boldsymbol{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u[e + (p/\rho)] \end{bmatrix}, \qquad \boldsymbol{G} = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho v[e + (p/\rho)] \end{bmatrix} \tag{2}$$

$$e = \frac{p}{(\gamma - 1)\rho} + \frac{u^2 + v^2}{2} \tag{3}$$

Equation (3), the equation of state for a perfect gas, permits closure of the system by establishing a relation between the thermodynamic variables.

Algorithm Components

The three algorithms compared in this paper share a number of common components. The differences in implementation of a subset of these components result in algorithms that vary significantly in storage requirements and performance. This section presents a discussion of the essential components of a Newton-like algorithm and some related issues.

Received Oct. 19, 1996; presented as Paper 97-0331 at the AIAA 35th Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan. 6–9, 1997; revision received Sept. 22, 1997; accepted for publication Nov. 17, 1997.

*Graduate Student, Institute for Aerospace Studies, 4925 Dufferin Street. E-mail: max@oddjob.utias.utoronto.ca. Student Member AIAA.

†Associate Professor, Institute for Aerospace Studies, 4925 Dufferin Street. E-mail: dwz@oddjob.utias.utoronto.ca. Member AIAA.

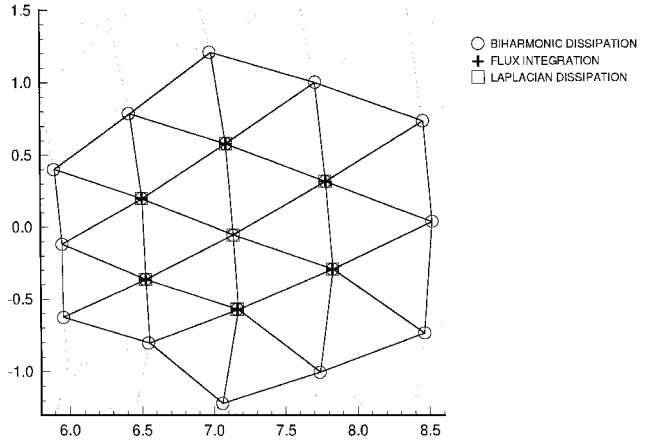

Fig. 1 Depiction of dissipation and flux stencils.

Discretization

Discretization of Eq. (1) over an unstructured triangular grid using a finite volume operator results in a system of nonlinear equations. This system is written as

$$\boldsymbol{R}(\boldsymbol{Q}) = 0 \tag{4}$$

where $\boldsymbol{Q} = [\rho, \rho u, \rho v, \rho e]^T$ and $\boldsymbol{R}$ is referred to as the residual. The residual operator consists of a centered flux integration and a nonlinear scalar artificial dissipation, which is a blend of Laplacian and biharmonic operators controlled by a pressure switch and is similar to that used by Mavriplis and Jameson.[21] Details of this particular implementation, including the boundary conditions, can be found in Refs. 22 and 23. The stencils of the three component operators are depicted in Fig. 1.

Newton's Method

Newton's method for the iterative solution of a nonlinear system of equations can be written as follows:

$$\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\right)^{(k)} \Delta \boldsymbol{Q}^{(k)} = -\boldsymbol{R}\left[\boldsymbol{Q}^{(k)}\right] \tag{5}$$

where the superscript k is the iteration index. The term $\partial \boldsymbol{R}/\partial \boldsymbol{Q}$ is the Jacobian matrix. Solution of this matrix equation for the unknown, $\Delta \boldsymbol{Q}^{(k)}$, and the update of the solution vector, $\boldsymbol{Q}^{(k+1)} = \boldsymbol{Q}^{(k)} + \Delta \boldsymbol{Q}^{(k)}$, is continued until the process is deemed to have converged. All cases presented later are initiated with uniform freestream flow conditions. The L_2 norm of the full residual vector is used to measure convergence:

$$L_2(\boldsymbol{R}) = \sqrt{\sum_{i=1}^{N} R_i^2} \tag{6}$$

where summation proceeds over all N unknowns.

GMRES

The nonsymmetric nature of the Jacobian matrix suggests the use of the GMRES algorithm of Saad and Schultz[2] in the solution of Eq. (5). Because the matrix problem is solved with an iterative method, a distinction must be drawn between these inner iterations and the outer Newton iterations. The inner iterations are terminated when the residual norm reaches an exit tolerance $\beta^{(k)}$ that depends on the outer residual $\boldsymbol{R}$ at each outer iteration:

$$\beta^{(k)} = \sigma \cdot L_2\left[\boldsymbol{R}^{(k)}\right] \tag{7}$$

where the parameter σ remains fixed. The maximum number of inner iterations is also fixed, causing GMRES to terminate even if the exit tolerance is not satisfied for a particular outer iteration. The choice of this parameter m can affect convergence, memory use, and robustness. The restarted GMRES algorithm is not used because the dimension of the Krylov subspace is kept small by forcing early exit to avoid the expense of solving the Jacobian problem with unnecessary precision.

The kernel of the GMRES procedure is the matrix-vector product. The standard way of forming this product involves the explicit multiplication of the sparse matrix and vector, both of which are stored in memory. The matrix-vector product required by GMRES can also be formed without explicitly performing the multiplication. This is done by making use of the Fréchet derivative of the residual function $\boldsymbol{R}$, which can be defined as

$$\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\right)^{(k)} \cdot \boldsymbol{v} = \frac{\boldsymbol{R}\left[\boldsymbol{Q}^{(k)} + \epsilon \boldsymbol{v}\right] - \boldsymbol{R}\left[\boldsymbol{Q}^{(k)}\right]}{\epsilon} \tag{8}$$

where $\boldsymbol{v}$ is the vector multiplicand. The parameter ϵ is calculated at every inner iteration as a function of machine zero ϵ_{mz} and the magnitude of $\boldsymbol{v}$, as described in Ref. 9:

$$\epsilon = \frac{\sqrt{\epsilon_{\text{mz}}}}{L_2(\boldsymbol{v})} \tag{9}$$

The resulting implementation of GMRES is termed matrix free.

Jacobian Matrix

The linearization employed in forming the Jacobian matrix of Eq. (5) has a significant impact on the performance of Newton-like methods. The most significant choice is whether to linearize the biharmonic artificial dissipation operator, shown in Fig. 1, because its stencil includes next-to-nearest neighbors. For implementations requiring explicit matrix-vector products, the biharmonic operator must be linearized to obtain the full-Newton method. Linearization of the biharmonic operator causes high computational and memory overheads due to the nonzeros in its stencil associated with next-to-nearest neighbors. In linearizing the artificial dissipation, we include neither the spectral radius nor the pressure switch, so that the artificial dissipation contributes nonzero entries to the diagonals of the blocks only. The same matrix is used to form the preconditioner for this standard full-Newton method.

Alternatively, the biharmonic operator can be truncated so as to leave Jacobian entries in positions corresponding to nearest-neighbor nodes only, as described in Ref. 22. The Laplacian dissipation added to the Jacobian matrix is then modified to become

$$\epsilon_2^J = \epsilon_2^R + \kappa \cdot \epsilon_4^R \tag{10}$$

where ϵ_2^R and ϵ_4^R are the coefficients of the Laplacian and biharmonic dissipation operators used in the residual and κ is a free parameter. A value of $\kappa = 5.0$ has been found to work well for a range of cases. The storage needs of the resulting first-order Jacobian matrix are significantly reduced vis-à-vis a complete linearization of the residual; however, the departure from Newton's method due to the incomplete linearization of the residual results in an increase in the number of outer iterations required to achieve convergence. We refer to this type of scheme as standard quasi-Newton.

The matrix-vector product from the matrix-free implementation is effectively the same as that obtained using a full linearization of the residual operator. It therefore yields a full-Newton scheme, which we term matrix-free, full-Newton. The chief advantage of this algorithm is that the full matrix need not be stored. However, some form of matrix is still required to form the BILU(n) preconditioner. The preconditioner used in the matrix-free scheme is based on the first-order matrix for two reasons: First, memory requirements are reduced with respect to the use of the full matrix for preconditioning; and second, results reported for structured grids by Pueyo and Zingg[10] indicate that their matrix-free, full-Newton scheme is significantly more efficient when using the first-order matrix to form the preconditioner.

Preconditioning

To date, most researchers have employed the BILU(0) preconditioner, which results from the adoption of a block-based matrix storage scheme, when using GMRES in implicit solvers. The choice of zero additional fill is due to the increased memory requirements associated with nonzero fill and the ease of its implementation.

We present results in which the use of the BILU(n) preconditioner is explored. Increasing the amount of fill can result in a reduction in the number of inner and outer iterations but leads to a higher

cost per application of the preconditioner and increased storage requirements. The optimal level of fill must be determined through numerical experiment. The preconditioner is frozen once the outer residual falls below 10^{-5} in all calculations.

Reordering of Unknowns

The ordering of unknowns can affect the number of nonzeros in the preconditioner, the effectiveness of the preconditioner, and cache performance. Four common reordering algorithms—reverse Cuthill–McKee (RCM), one-way dissection (1WD), nested dissection (ND), and quotient minimum degree (QMD)—are tested. The natural ordering of unknowns produced by the grid-generation process is also included in the comparisons.

Storage Model

The choice of storage model for the Jacobian matrix can also have a significant impact on solver performance and storage requirements. The data structure is such that the Jacobian matrix is sparsely populated by four-by-four blocks of nonzero entries. This fact can be exploited by the use of a block-based compressed sparse row (CSR) model in which the matrix entries are grouped into four-by-four blocks. This type of storage has been found to improve performance significantly over the scalar-based CSR model presented in Ref. 6. The difference in performance is due to the improved caching and register use of block-based operations.

The ILU(n) preconditioner that results from a scalar storage model differs from the BILU(n) preconditioner due to the extra nonzeros in the four-by-four blocks of the latter. This occurs because the four-by-four blocks are not fully populated, as shown in Table 1. Although the flexibility of the scalar storage model allows the matrix nonzero structure to be tailored exactly to its needs, whereas the block-based model must allocate space for all 16 block entries even if they are zeros, its total size is penalized by the need to store an integer column-index pointer for each nonzero entry. The block-based scheme, in contrast, needs to store only a block-column-index pointer for each block, thus reducing the number of such column pointers by approximately a factor of 16.

Note that the extra nonzeros inherent to the block-based storage model may well change the character and quality of an incomplete factorization. However, the BILU(n) preconditioner resulting from the block-based storage can effectively be duplicated by the scalar model by adding the extra nonzeros of the block matrix to the scalar matrix before calculation of the preconditioner. We term the resulting preconditioner block/scalar ILU(n) [B/SILU(n)].

Initial Iterations of Transonic Flows

The convergence of the quasi-Newton scheme is roughly the same for subsonic and transonic flows, except that instabilities may develop during the initial iterations of transonic cases. This problem is encountered with transonic cases due to shock formation and motion as the solution evolves toward steady state. When solving transonic flows, the Jacobian is modified by the addition of entries of the form $1/\Delta t$ to its diagonal, which results in the much more stable implicit Euler pseudo-time-marching scheme. This diagonal conditioning is decreased as the solution progresses and is turned off after 25 outer iterations.

The matrix-free, full-Newton algorithm tends to converge in fewer than 10 iterations once the solution lies within the radius of convergence of Newton's method. Outside this region, it can fail to converge. This is a problem with transonic flows only, whereas subsonic flows allow the use of the full-Newton algorithm from the first iteration. The implicit Euler approach discussed in preceding sections might also be used with the matrix-free implementation, but we prefer a hybrid approach because of the high cost per iteration of the matrix-free scheme. The matrix-free algorithm therefore makes use of the lower cost per outer iteration of the quasi-Newton solver to provide it with an initial solution lying within its radius of convergence. The switch from quasi- to full-Newton solver occurs when the outer residual falls below a preselected level. Although it is desirable to switch to the full-Newton mode as early as possible, instability can result from premature changeover. The solver converges reliably for transonic cases if the switching occurs when the norm of the outer residual is equal to 10^{-3}.

Table 1 Nonzero structure resulting from linearization of dissipation and flux operators[a]

Next-to-nearest neighbor				Nearest neighbor				Diagonal			
x	0	0	0	x	x	x	0	x	0	0	0
0	x	0	0	x	x	x	x	0	x	0	0
0	0	x	0	x	x	x	x	0	0	x	0
0	0	0	x	x	x	x	x	0	0	0	x

[a]x, nonzero, and 0, zero.

Algorithms

The three algorithms that are compared in this paper differ most significantly in their treatment of the Jacobian matrix. Parameter studies were performed for GMRES(m, σ) and BILU(n) over the ranges of values $m = \{5, 10, 15, 20, 25\}$, $\sigma = \{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$, and $n = \{0, 1, 2, 3, 4, 5, 6\}$ for each algorithm. The range of values that provides near-optimal performance is given for each algorithm. All results shown were obtained using the RCM ordering and block CSR storage. A summary of the algorithms' salient features is presented next.

Standard Quasi-Newton (SQN)

1) Standard GMRES implementation
2) GMRES($m = \{10, 15, 20, 25\}$, $\sigma = \{0.1, 0.05, 0.01\}$)
3) First-order Jacobian matrix
4) BILU($n = \{1, 2, 3\}$) preconditioner

Standard Full-Newton (SFN)

1) Standard GMRES implementation
2) GMRES($m = \{15, 20, 25\}$, $\sigma = \{0.05, 0.01, 0.005\}$)
3) Second-order Jacobian matrix
4) BILU($n = \{1, 2, 3\}$) preconditioner

Matrix-Free, Full-Newton (MFFN)

1) Matrix-free GMRES implementation
2) GMRES($m = \{15, 20, 25\}$, $\sigma = \{0.1, 0.05, 0.01\}$)
3) Second-order Jacobian matrix via Fréchet derivative
4) BILU($n = \{3, 4, 5, 6\}$) preconditioner

Test Cases

Results are presented for subsonic ($M_\infty = 0.63$ and $\alpha = 2.0$ deg) and transonic ($M_\infty = 0.85$ and $\alpha = 1.0$ deg) flows over the NACA 0012 airfoil. Calculations were performed on two grids: one grid consisted of 14,193 nodes, 350 of which were located on the airfoil surface and 120 of which were located on the far-field boundary; the other consisted of 3631 nodes, 150 of which were located on the airfoil surface and 60 of which were located on the far-field boundary. The latter was employed only for results in which comparison was made with the full-Newton scheme using standard GMRES because of the high memory requirements of this approach. Both grids were generated using an algorithm due to Predovic[23] that employs a combination of advancing front and Delaunay triangulation techniques.

All convergence histories were obtained using an SGI Indigo2 workstation (75 MHz IP26 R8000 CPU with 16 kbytes of primary cache and 2 Mbytes of secondary cache) and the same compiler optimizations. Comparison was made on the basis of CPU time because iteration or work unit counts do not fully reflect the relative performance.

Results

Optimum performance results are presented for each of the algorithms, followed by expositions of the effects of selection of preconditioner, reordering, and storage model on the performance of the matrix-free, full-Newton algorithm. The optimum parameters and memory requirements (in megabytes) for each algorithm are shown in Table 2. Memory use shown in Table 2 includes integer (4 bytes) row- and block-column-index pointers and double-precision (8 bytes) values necessary for storing matrix and preconditioner.

Table 2 Optimal parameter sets and storage costs for various algorithms

Solver	Case	Grid	Prec.	GMRES (m, σ)	Matrix storage	Prec. storage
SFN	Subsonic	Coarse	BILU(1)	20, 0.005	8.5	13.3
MFFN	Subsonic	Coarse	BILU(4)	20, 0.01	3.1	8.7
SQN	Subsonic	Fine	BILU(2)	15, 0.05	12.5	22.4
MFFN	Subsonic	Fine	BILU(4)	20, 0.01	12.5	35.5
SQN	Transonic	Fine	BILU(2)	15, 0.01	12.5	22.4
MFFN	Transonic	Fine	BILU(4)	20, 0.01	12.5	35.5

Table 3 Convergence data for optimal runs of various algorithms

Solver	Case	Grid	t_{prec}	t_{GMRES}	t_{total}	N_{in}	N_{out}
SFN	Subsonic	Coarse	7.1	31.2	01:35	13.3	7
MFFN	Subsonic	Coarse	3.4	23.3	00:32	11.9	7
SQN	Subsonic	Fine	52.3	304.3	14:22	5.9	75
MFFN	Subsonic	Fine	18.5	189.8	03:57	18.2	9
SQN	Transonic	Fine	91.5	543.1	21:06	8.9	95
MFFN	Transonic	Fine	69.2	339.5	10:20	12.0	35

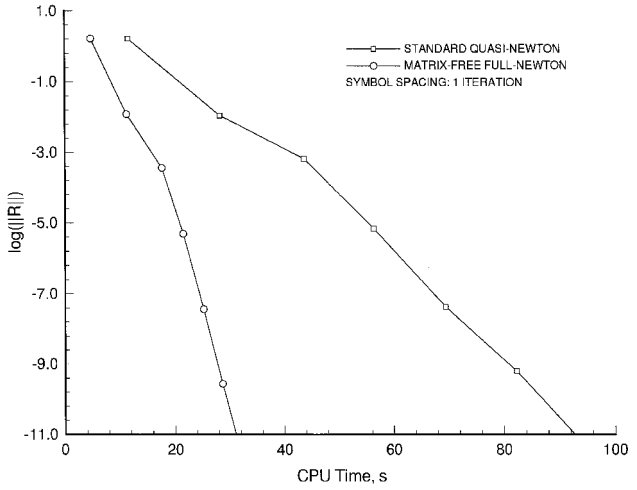

Fig. 2 Optimal convergence of the standard and matrix-free, full-Newton algorithms for the subsonic test case on the coarse grid.

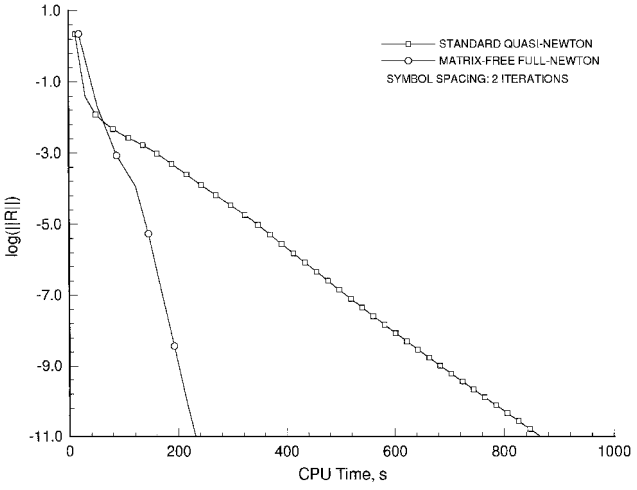

Fig. 3 Optimal convergence of the matrix-free, full-Newton and quasi-Newton algorithms for the subsonic test case on the fine grid.

Newton's Method and GMRES Implementation

The matrix-free and standard full-Newton algorithms are compared in Fig. 2. Figures 3 and 4 compare the standard quasi-Newton and matrix-free, full-Newton algorithms. All relevant CPU data are shown in Table 3, which includes the CPU time spent in calculating the BILU(n) factors, t_{prec}; in the GMRES algorithm, t_{GMRES}; and overall, t_{total}; as well as the average number of inner iterations N_{in} at each outer iteration and the total number of outer iterations N_{out} for a residual reduction of 12 orders of magnitude. The balance of CPU time not shown consists of setup time and calculation of the residual and Jacobian.

Table 4 Effect of the fill parameter n on the performance of the matrix-free solver for the subsonic case

Prec.	GMRES (m, σ)	NPNZ	Prec. storage	t_{prec}	t_{GMRES}	t_{total}	N_{in}	N_{out}
BILU(0)	20, 0.01	1,557,568	12.5	16.2	793.2	15:22	19.9	45
BILU(1)	20, 0.01	2,043,600	16.3	12.8	429.4	8:29	19.5	25
BILU(2)	20, 0.01	2,817,408	22.4	13.2	275.5	5:32	19.0	15
BILU(3)	20, 0.01	3,648,992	28.9	16.3	239.2	4:51	18.8	12
BILU(4)	20, 0.01	4,487,920	35.5	18.5	189.8	3:57	18.2	9
BILU(5)	20, 0.01	5,331,568	42.2	24.8	180.4	3:53	18.0	8
BILU(6)	20, 0.01	6,177,680	48.8	32.2	189.1	4:10	17.8	8

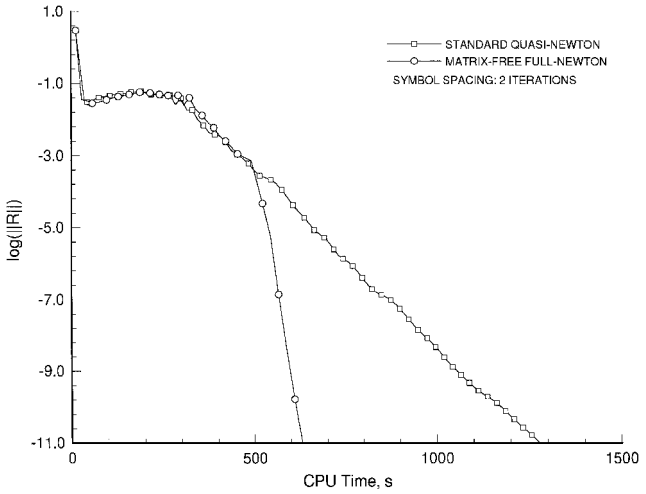

Fig. 4 Optimal convergence of the matrix-free, full-Newton and quasi-Newton algorithms for the transonic test case on the fine grid.

Comparison of the matrix-free and standard full-Newton algorithms is done on the coarse grid for the subsonic case due to the high storage needs of the latter algorithm. The matrix-free algorithm is faster by a factor of 3. Its superior performance is primarily due to the use of the smaller first-order Jacobian, which also leads to substantially lower memory requirements. The standard full-Newton scheme has therefore been discarded from further consideration.

For the subsonic calculation on the fine grid, the matrix-free algorithm is more than three times faster than the quasi-Newton algorithm, as shown in Fig. 3. The superior performance of the matrix-free algorithm is due to the reduction in the number of outer iterations, as both inner and outer iterations are more costly. The storage requirements of the matrix-free algorithm are the same as those of the quasi-Newton algorithm for a given level of preconditioner fill because both solvers use the first-order Jacobian matrix and the corresponding BILU(n) preconditioner. However, the optimum preconditioner is larger for the matrix-free, full-Newton scheme than for the quasi-Newton scheme.

Figure 4 compares the matrix-free, full-Newton algorithm with the quasi-Newton algorithm for the transonic case on the fine grid. These results show the matrix-free algorithm to be almost twice as fast as the quasi-Newton algorithm. The start-up phase before switch over to matrix-free mode accounts for 80% of the total CPU expense for this case.

BILU(n) Preconditioner

Measured strictly in terms of CPU expense, the BILU(5) preconditioner is optimal for the matrix-free, full-Newton algorithm, as shown in Table 4 and Fig. 5. However, the slight increase in speed over BILU(4) does not justify the increased storage. Therefore we consider BILU(4) to be optimal. It is clear that increasing the amount of fill greatly improves the quality of the preconditioner. Up to $n = 5$, this outweighs the increased cost of application due to the additional fill. There is a penalty in memory use, as the number of preconditionernonzeros NPNZ increases with fill level n. Table 4

Table 5 Effect of reorderings on the performance of the matrix-free solver for the subsonic case; BILU(4) and GMRES(20, 0.01)

Reordering	NPNZ	Prec. storage	t_{total}	N_{in}	N_{out}	t_{GMRES}	$\bar{t}_{GMRES}$
None	4,528,944	35.9	4:31	18.3	10	212.9	1.69
ND	5,437,552	43.0	10:46	19.7	21	524.2	1.88
1WD	5,174,784	40.9	6:20	19.4	13	311.8	1.77
RCM	4,487,920	35.5	3:57	18.2	9	192.7	1.71
QMD	4,994,368	39.5	10:08	19.7	20	490.0	1.88

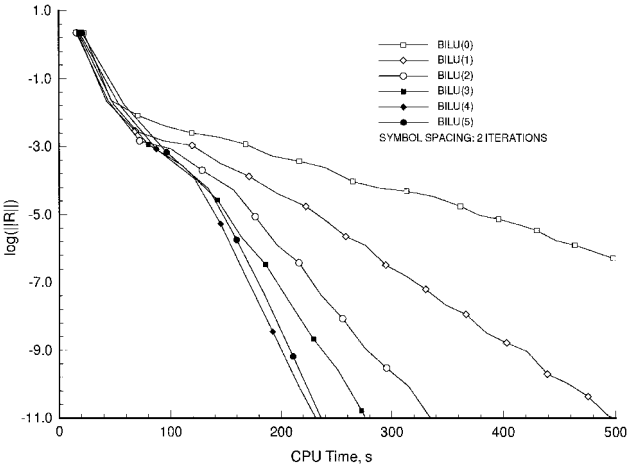

Fig. 5 Effect of fill parameter n on the performance of the matrix-free algorithm for the subsonic test case on the fine grid.

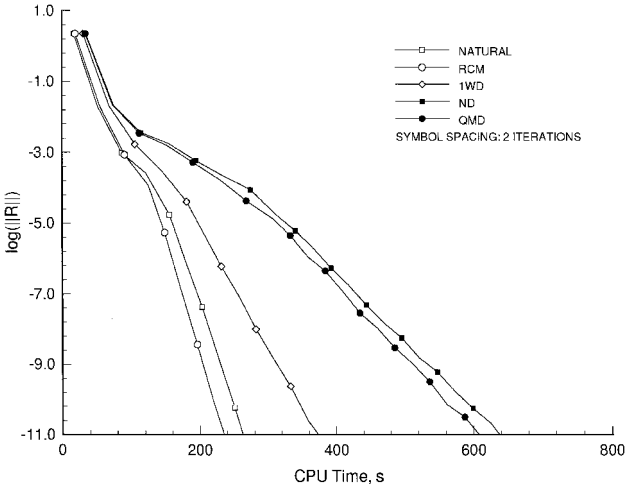

Fig. 6 Effect of reorderings on the performance of the matrix-free, full-Newton algorithm for the subsonic test case on the fine grid.

shows that the memory requirements of the matrix-free solver can be reduced by decreasing the amount of fill in the preconditioner, with a corresponding increase in CPU time. However, the matrix-free, full-Newton solver remains considerably faster than the quasi-Newton solver even with BILU(1) preconditioning.

Reordering

Table 5 and Fig. 6 compare the reorderings using results for the subsonic flow obtained with the matrix-free, full-Newton algorithm and the BILU(4) preconditioner. These results show that the reorderings can affect the quality of the BILU(n) preconditioner. When using the natural, 1WD, ND, and QMD orderings, the Jacobian matrix problem of Eq. (4) is not solved with sufficient precision because of the forced exit from GMRES after 20 inner iterations. This results in an increase in the number of outer iterations required. Furthermore, RCM leads to the fewest nonzero entries in the preconditioner. As a result, RCM is the preferred ordering for this application. It is interesting to note that the natural ordering is nearly as effective as the RCM ordering despite producing a much larger bandwidth, as shown in Figs. 7 and 8.

Table 6 Comparison of ILU, B/SILU, and BILU preconditioners for the matrix-free, full-Newton solver with and without I-node (I) and loop unrolling (U) optimizations; subsonic case, fine grid, GMRES(20, 0.01)

Storage model	Opts	Prec.	Matrix storage	Prec. storage	t_{prec}	t_{GMRES}	t_{total}
Scalar	None	ILU(4)	15.1	51.9	51.4	292.1	6:40
Scalar	I	ILU(4)	15.1	51.9	48.1	310.3	6:56
Scalar	None	B/SILU(4)	18.0	51.6	49.1	291.6	6:30
Scalar	I	B/SILU(4)	18.0	51.6	39.4	235.0	5:24
Block	None	BILU(4)	12.5	35.5	92.6	672.7	13:16
Block	U	BILU(4)	12.5	35.5	18.5	189.8	3:56

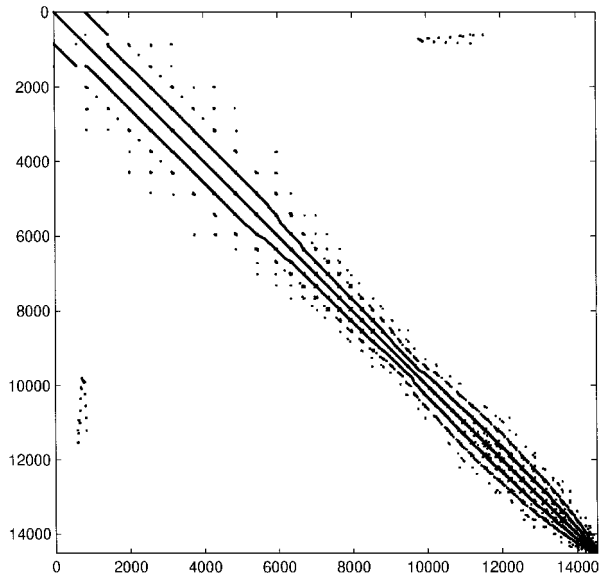

Fig. 7 Jacobian matrix structure on coarse grid with natural ordering.

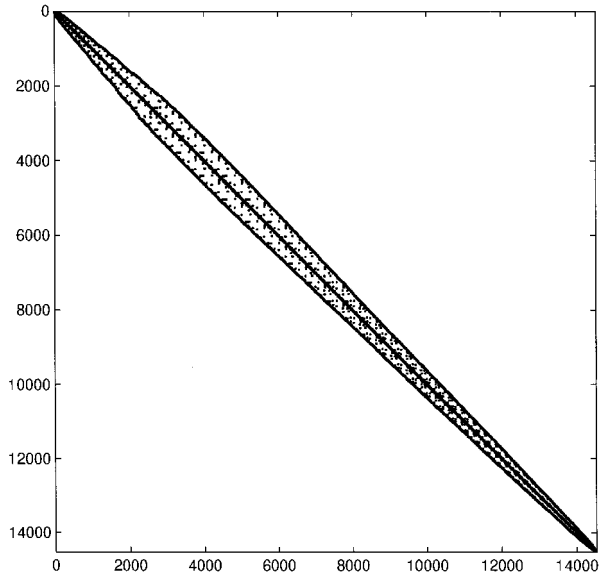

Fig. 8 Jacobian matrix structure on coarse grid after RCM reordering.

Block vs Scalar CSR Storage Models

Results are shown in Table 6 and Fig. 9 for the subsonic calculation using the matrix-free, full-Newton algorithm, both with and without coding optimizations. Loop unrolling (U) can be applied to block-based operations, whereas I nodes (I), a technique by which identical column-index arrays are loaded only once and then reused, can be applied when using the scalar-based storage model. I-node optimizations perform better for the B/SILU(4) algorithm because all four rows corresponding to the degrees of freedom of a mesh node are identical and can be represented by one I node.

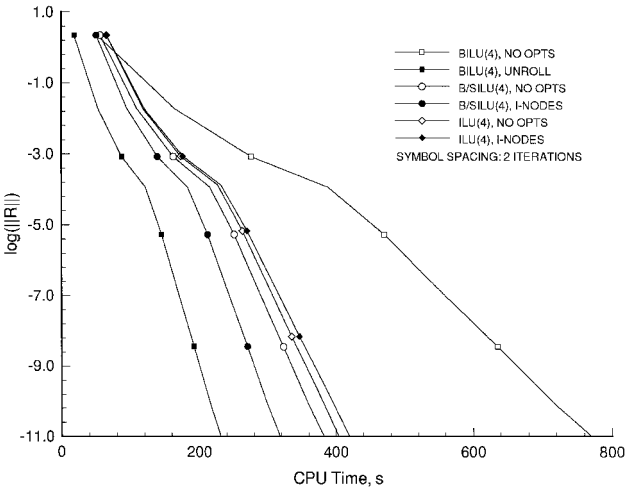

Fig. 9 Comparison of ILU, B/SILU, and BILU preconditioners for the matrix-free, full-Newton algorithm for the subsonic test case on the fine grid.

Comparison of the performance of ILU(4) and B/SILU(4) reveals that the addition of extra nonzeros into the B/SILU(4) preconditioner has little effect on the quality of the preconditioner, as no change is seen in the number of inner or outer iterations. The difference in quality is seen only when using zero fill, as B/SILU(0) proves to be much superior to ILU(0).

Comparison of the B/SILU(4) and BILU(4) preconditioners demonstrates the savings that can be obtained by doing matrix calculations on a block basis. These savings are due to both loop unrolling, which encourages efficient use of registers, and improved data caching, as the data necessary for the block-based operations are located in contiguous physical memory. Detailed examination of the number of cache misses shows the superiority of the block storage model in this regard.

Conclusions

Three Newton-like algorithms for computing steady aerodynamic flows have been presented, optimized, and compared. The results show that the reverse Cuthill–McKee ordering is best for these algorithms and that BILU(n) is the preferred preconditioner, with n equal to 4 for the matrix-free, full-Newton algorithm. The matrix-free, full-Newton algorithm produces the fastest convergence and is considerably faster than the quasi-Newton algorithm. Given the previous observation that the quasi-Newton approach is comparable in speed to a fast multigrid solver, it is clear that a matrix-free, full-Newton algorithm is an extremely competitive option. Future work will involve extension to turbulent flows and improvements to the startup phase of transonic cases.

References

[1]Venkatakrishnan, V., "Newton Solution of Inviscid and Viscous Problems," AIAA Paper 88-0413, Jan. 1988.

[2]Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[3]Coleman, T. F., *Large Sparse Numerical Optimization*, Vol. 165, Lecture Notes in Computer Science, Springer–Verlag, New York, 1984, Chap. 4.

[4]Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids," AIAA Paper 95-1740, June 1995.

[5]Barth, T. J., and Linton, S. W., "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation," AIAA Paper 95-0221, Jan. 1995.

[6]Blanco, M., and Zingg, D. W., "A Fast Solver for the Euler Equations on Unstructured Grids Using a Newton-GMRES Method," AIAA Paper 97-0331, Jan. 1997.

[7]Forsyth, P. A., and Jiang, H., "Iterative Methods for Full Newton Solution of the Euler Equations," *Sixth International Symposium on Computational Fluid Dynamics*, Vol. 1, Elsevier, New York, 1995, pp. 318–323.

[8]Johan, Z., Hughes, T. J. R., and Shakib, F., "A Globally Convergent Matrix-Free Algorithm for Implicit Time-Marching Schemes Arising in Finite Element Analysis in Fluids," *Computer Methods in Applied Mechanics and Engineering*, Vol. 87, Nos. 2, 3, 1991, pp. 281–304.

[9]Nielsen, E. J., Anderson, W. K., Walters, R. W., and Keyes, D. E., "Application of Newton–Krylov Methodology to a Three-Dimensional Unstructured Euler Code," AIAA Paper 95-1733, June 1995.

[10]Pueyo, A., and Zingg, D. W., "Progress in Newton-Krylov Methods for Aerodynamic Calculations," AIAA Paper 97-0877, Jan. 1997.

[11]Anderson, W. K., "A Grid Generation and Flow Solution Method for the Euler Equations on Unstructured Grids," *Journal of Computational Physics*, Vol. 110, No. 1, 1994, pp. 23–38.

[12]Barth, T. J., "Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes," AIAA Paper 91-0721, Jan. 1991.

[13]Batina, J. T., "A Fast Implicit Upwind Solution Algorithm for Three-Dimensional Unstructured Dynamic Meshes," AIAA Paper 92-0447, Jan. 1992.

[14]Cabello, J., Morgan, K., Lohner, R., Luo, H., and Baum, J., "An Implicit Solver for Laminar Compressible Flows on Unstructured Grids," AIAA Paper 95-0344, Jan. 1995.

[15]Luo, H., Baum, J., Lohner, R., and Cabello, J., "Implicit Schemes and Boundary Conditions for Compressible Flows on Unstructured Meshes," AIAA Paper 94-0816, Jan. 1994.

[16]Rogers, S. E., "A Comparison of Implicit Schemes for the Incompressible Navier–Stokes Equations with Artificial Compressibility," AIAA Paper 95-0567, Jan. 1995.

[17]Venkatakrishnan, V., and Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," *Journal of Computational Physics*, Vol. 105, No. 1, 1993, pp. 83–91.

[18]Dutto, L. C., "The Effect of Ordering on Preconditioned GMRES Algorithm, for Solving the Compressible Navier–Stokes Equations," *International Journal for Numerical Methods in Engineering*, Vol. 36, No. 3, 1993, pp. 457–497.

[19]Saad, Y., "Krylov Subspace Techniques, Conjugate Gradients, Preconditioning and Sparse Matrix Solvers," *Computational Fluid Dynamics*, edited by H. Deconinck, Vol. 5, Lecture Series, von Kármán Inst. for Fluid Dynamics, Rhode-Sainte-Genèse, Belgium, 1994, Chap. 6.

[20]George, A., and Liu, J. W., *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall Series in Computational Mathematics, Prentice–Hall, Reading, MA, 1981, Chaps. 4, 5, 7, and 8.

[21]Mavriplis, D. J., and Jameson, A., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," AIAA Paper 87-0353, Jan. 1987.

[22]Blanco, M., "An Implicit Solution Method for the Euler Equations on Unstructured Triangular Grids," M.A.Sc. Thesis, Inst. for Aerospace Studies, Univ. of Toronto, Downsview, ON, Canada, Sept. 1994.

[23]Predovic, D. T., "Multigrid Solution of the Euler Equations Using Unstructured Adaptive Meshes," M.A.Sc. Thesis, Inst. for Aerospace Studies, Univ. of Toronto, Downsview, ON, Canada, April 1994.

J. Kallinderis
Associate Editor