JOURNAL OF COMPUTATIONAL PHYSICS **123,** 379–393 (1996)
ARTICLE NO. 31

Efficient Iterative Methods Applied to the Solution of Transonic Flows

ANDREW M. WISSINK*, ANASTASIOS S. LYRINTZIS†, AND ANTHONY T. CHRONOPOULOS‡

Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, Minnesota 55455, †School of Aeronautics and Astronautics, Purdue University, West Lafayette, Indiana 47907, and ‡Computer Science, Wayne State University, Detroit, Michigan 48202

Received January 12, 1994; revised July 11, 1995

We investigate the use of an inexact Newton's method to solve the potential equations in the transonic regime. As a test case, we solve the two-dimensional steady transonic small disturbance equation. Approximate factorization/ADI techniques have traditionally been employed for implicit solutions of this nonlinear equation. Instead, we apply Newton's method using an exact analytical determination of the Jacobian with preconditioned conjugate gradient-like iterative solvers for solution of the linear systems in each Newton iteration. Two iterative solvers are tested; a block s-step version of the classical Orthomin(k) algorithm called orthogonal s-step Orthomin (OSOmin) and the well-known GMRES method. The preconditioner is a vectorizable and parallelizable version of incomplete LU (ILU) factorization. Efficiency of the Newton–Iterative method on vector and parallel computer architectures is the main issue addressed. In vectorized tests on a single processor of the Cray C-90, the performance of Newton–OSOmin is superior to Newton–GMRES and a more traditional monotone AF/ADI method (MAF) for a variety of transonic Mach numbers and mesh sizes. Newton–GMRES is superior to MAF for some cases. The parallel performance of the Newton method is also found to be very good on multiple processors of the Cray C-90 and on the massively parallel thinking machine CM-5, where very fast execution rates (up to 9 Gflops) are found for large problems.

1. INTRODUCTION

It has always been an objective of researchers in computational fluid dynamics (CFD) to develop more efficient numerical solution procedures. This is particularly true today when many of the more interesting engineering simulations, namely 3D unsteady problems with meshes over 1 million gridpoints, push the limits of existing computer technology. In the past, numerical methods were developed for the purpose of approximating the solution to the problem with the least amount of computational work. This led to many efficient implicit algorithms, the most common being approximate factorization/alternating direction implicit (AF/ADI) and multi-grid methods. However, with the development of vector-parallel and, more recently, massively parallel computer architectures, the efficiency of the algorithm on these architectures is as dominant a factor as its convergence qualities.

When powerful machines such as the Cray-2 became available in the late 1980s, a number of researchers investigated use of exact Newton's method for solving steady state CFD problems. Direct sparse matrix solvers (e.g., Gaussian elimination) were used for exact solution of the linear systems in each Newton iteration. Results using this approach were obtained for transonic flows using the potential equations [1, 2], Euler equations [3, 4], and Navier Stokes equations [5, 6]. While the exact Newton's method was found to be robust and have quadratic convergence (with a good initial guess), the CPU time was not competitive with existing iterative implicit methods. This was due mainly to the time required for exact solution of the large linear systems.

An approach that has shown promising results recently is the inexact Newton's method. In this approach, an inexact solution using an iterative solver is performed in each Newton iteration. There are two advantages of using a conjugate gradient-like iterative solver over a direct solver for a problem with n unknowns. First, direct methods tend to be memory intensive and costly, requiring $O(n^2)$ operations to generate an exact solution for banded systems with bandwidth $\sqrt{n}$ (arising in two-dimensional problems). Iterative methods can generate a good approximation to the solution in far fewer than $O(n^2)$ iterations. The second advantage is that conjugate gradient-like iterative methods contain many long vector operations and are consequently well suited for parallel execution. Promising results using the Newton method coupled with linear iterative methods (Newton–iterative) have been obtained for Navier–Stokes calculations in two dimensions for a variety of problems. McHugh and Knoll [7] have computed low speed internal viscous flows and Ajmani *et al.* [8] have computed subsonic viscous flows over an airfoil. Venkatakrishnan [9] has used the inexact Newton methods to compute transonic viscous flows over an airfoil and Ajmani *et al.* [10] and Orkwis and McRae [11, 12] have computed high speed viscous internal flows. Other recent interesting applications of Newton's method to solve nonlinear CFD problems are [13–17].

0021-9991/96 $18.00

A popular iterative solver used in the Newton–iterative methods is the GMRES algorithm of Saad and Shultz [18]. GMRES has gained wide acceptance in the CFD community due to its ability to solve difficult nonsymmetric linear systems. It is the iterative solver used in [7–10, 13, 19]. Another method capable of solving nonsymmetric linear systems is the Orthomin algorithm of Vinsome [20]. Orthomin has not received much attention in the CFD community because it does not have the robustness qualities of GMRES. However, Chronopoulos [21] recently introduced an s-step version of Orthomin, which enhances its robustness qualities as well as its parallelization potential. Swanson and Chronopoulos [22] have modified the approach to make the s-steps orthogonal. The resulting orthogonal s-step Orthomin algorithm attains a level of robustness comparable to GMRES and has shown good efficiency on a parallel computer architecture (e.g., [22]). In a recent comparison study by McHugh and Knoll [7], it was determined that GMRES is superior to several other iterative methods for inexact Newton solution of two-dimensional Navier–Stokes problems. It is for this reason that we compare the s-step Orthomin method to GMRES in this work.

In this paper, we address the issue of efficiency of the Newton–iterative approach for solving the 2D steady transonic small disturbance (TSD) potential equation. Traditionally, this equation has been solved using an AF/ADI method for the implicit operator (e.g., [23, 26]). Instead, we apply Newton's method using an analytical exact computation of the Jacobian and solve the linear systems with conjugate gradient-like iterative methods. In earlier work with Newton's method [24], we used the Orthomin(k) [20] algorithm for the iterative solver with a vectorized incomplete LU (ILU) factorization preconditioner [25]. The results compared favorably with the MAF method of Goorjian [26], which uses an AF/ADI implicit operator. However, the method lacked robustness for more difficult high Mach number transonic flows. In this paper, we apply the more robust block s-step version of Orthomin(k), referred to as OSOmin(s,k) [22], along with the well-known GMRES(m) algorithm. The vectorized ILU preconditioner is modified to be more parallelizable using the technique of Di Brozolo and Robert [27]. An earlier version of this work was presented as Ref. [28].

The issue of parallelization is also addressed. The Newton–iterative method is compared to the MAF algorithm of Goorjian [26] for a test problem on a vector-parallel architecture (Cray C-90) with eight processors and a massively parallel architecture (thinking machine CM-5) with up to 512 processors. Both preconditioned and unpreconditioned solvers are tested using a variety of mesh sizes and Mach numbers.

2. THEORETICAL DEVELOPMENT

The transonic small disturbance (TSD) equation is a simplified version of the full potential equation, assuming small disturbances. It is an accurate prediction for flows around thin objects with relatively weak shocks, where the isentropic assumption of the potential equations is valid. For two-dimensional steady flow, the TSD equation can be written as

$$\left[\frac{1-M_\infty^2}{\tau^{2/3}(\gamma+1)M_\infty^2}-\Phi_x\right]\Phi_{xx}+\Phi_{yy}=0, \tag{1}$$

where Φ is the perturbation potential, subscripts x and y denote differentiation, M_∞ is the free stream Mach number, and τ is the maximum airfoil thickness nondimensionalized by the airfoil chord.

The first term in the bracket is defined as K, the transonic similarity parameter:

$$K=\frac{1-M_\infty^2}{\tau^{2/3}(\gamma+1)M_\infty^2}. \tag{2}$$

The $K-\Phi_x$ term is the nonlinear term in the TSD equation. If $K-\Phi_x$ is greater than zero (subsonic flow), the equation is elliptic, whereas if $K-\Phi_x$ is less than zero (supersonic flow), the equation is hyperbolic. Hence the system of equations describing the transonic flowfield is a mixed set of elliptic and hyperbolic equations. This mixture complicates the finite difference discretization of the flowfield. For points where the TSD equation is elliptic, central finite differences should be used and for the points where it is hyperbolic, upwind differences should be used.

A technique to perform this switching between upwind and central differences was introduced by Murman and Cole in 1971 [29]. Finite differences are used to find the value of $K-\Phi_x$, and switching is done depending upon whether $K-\Phi_x$ is positive or negative. Although the Murman switch satisfies conservation properly, it has the weakness of allowing numerical instabilities to develop in regions of flow around complicated geometries or blunt surfaces, such as flow near the leading edge of an airfoil. In 1985, Goorjian *et al.* [26] introduced the monotone switch, based on the ideas of Godunov [30]. As the name "monotone" implies, this switch eliminates oscillations, improving convergence and thereby making the method more robust. The differencing approximation of the TSD equation with the monotone switch incorporated is

$$F(\Phi^{n+1})=[(\tilde{G}\Delta_x^+ + \hat{G}\Delta_x^-)\delta_x^- + \delta_{yy}]\Phi^{n+1}\to 0, \tag{3}$$

where

$$\tilde{G} = \frac{(\gamma+1)M_\infty^2}{2}\left[(1-\varepsilon_{i,j})\{(1-\varepsilon_{i+1/2,j})\nabla u_{i+1,j} + \varepsilon_{i-1/2,j}\nabla u_{i,j}\} + (1-\varepsilon_{i-1/2,j})\nabla u_{i,j}\right], \quad (4)$$

$$\hat{G} = \frac{(\gamma+1)M_\infty^2}{2}\left[\varepsilon_{i-1,j}\{(1-\varepsilon_{i-1/2,j})\nabla u_{i,j} + \varepsilon_{i-3/2,j}\nabla u_{i-1,j}\} + (\varepsilon_{i-1/2,j})\nabla u_{i,j}\right], \quad (5)$$

and

$$\nabla u_{i,j} = K - \delta_x^- \Phi_{i,j}^n. \quad (6)$$

The difference operators are standard finite difference operators, defined by

$$\Delta_x^+ f_{i,j}^n = \frac{f_{i+1,j}^n - f_{i,j}^n}{\frac{1}{2}(x_{i+1} - x_{i-1})}$$

$$\Delta_x^- f_{i,j}^n = \frac{f_{i,j}^n - f_{i-1,j}^n}{\frac{1}{2}(x_{i+1} - x_{i-1})}$$

$$\delta_x^- f_{i,j}^n = \frac{f_{i,j}^n - f_{i-1,j}^n}{(x_i - x_{i-1})}$$

$$\delta_{yy} f_{i,j}^n = \Delta_y^+(\delta_y^- f_{i,j}^n)$$

and the switches are defined by

$$\varepsilon_{i,j} = 0 \quad \text{if } \tfrac{1}{2}(\nabla u_{i+1,j} + \nabla u_{i,j}) \geq 0$$
$$= 1 \quad \text{otherwise;}$$
$$\varepsilon_{i+1/2,j} = 0 \quad \text{if } \nabla u_{i+1,j} \geq 0$$
$$= 1 \quad \text{otherwise.}$$

The monotone approximate factorization (MAF) algorithm of Goorjian *et al.* [26] uses the above discretization technique, coupled with an AF/ADI solution technique, to solve the TSD equation. Algorithm details can be found in [26]. A geometric acceleration parameter sequence, originally proposed in the AF2 algorithm of Ballhaus and Jameson [23], is used to accelerate the convergence of MAF. The size of this acceleration parameter sequence, g, can be adjusted to give optimal convergence. Optimal values of g for the problems we tested are given in the results section, where the MAF algorithm is compared the Newton–iterative method.

In our approach, after discretization using the above approach, an inexact Newton algorithm is applied to solve the nonlinear system $F(\Phi^{n+1})$, as defined in Eq. (1).

ALGORITHM. Inexact Newton.

Choose Φ_0.

For $n = 0,1, \ldots$ **until** convergence **do**

1. Solve Iteratively: $F'(\Phi^n)C^n = -F(\Phi^n)$
2. $\Phi^{n+1} = \Phi^n + C^n$

EndFor

The Jacobian of the function, $F'(\Phi^n)$, is computed in its exact form analytically. For higher order accurate problems, exact computation of the Jacobian can be CPU intensive. However, with the first-order TSD equation, exact computation is relatively easy and requires only a small percentage of the CPU time (2% to 5%). Most of the CPU time is spent solving the linear systems with the preconditioned iterative solver. The large, nonsymmetric, sparse linear systems are banded with six diagonals, representing coefficients $\Phi_{i,j-1}$, $\Phi_{i-2,j}$, $\Phi_{i-1,j}$, $\Phi_{i,j}$, $\Phi_{i+1,j}$, and $\Phi_{i,j+1}$ (Fig. 1).

The diagonals $\Phi_{i-2,j}$ and $\Phi_{i+1,j}$ exist in different regions of the flowfield, depending upon whether the flow is subsonic or supersonic. In subsonic regions, central differencing is used and the $\Phi_{i-2,j}$ entry is 0. In supersonic regions, upwind differencing is used and the $\Phi_{i+1,j}$ entry is 0. In a completely subsonic flowfield, the system would be a slightly nonsymmetric pentadiagonal system. As a larger portion of the flow becomes supersonic (caused by increasing Mach number) the magnitude of the factors in the sixth diagonal become larger. For some difficult cases, the systems cannot be solved by the iterative methods and these systems could be indefinite.

An important consideration in the use of inexact Newton methods is the level of accuracy required of the linear solution in order to assure convergence of the Newton method. Superlinear convergence requires that the inner linear iteration (i.e., solving the linear system of equations) be solved more accurately than the outer (i.e., Newton) iteration. For quadratic convergence, the inner iteration must be solved to machine accuracy. Since it is usually impractical to use iterative methods to solve to machine accuracy, the typical criteria used for calculations of this type is the 2-norm of the linear residual must be less than or equal to the 2-norm of the nonlinear residual, as described in [31]. However, this criteria presupposes that the nonlinear residual decreases in each iteration. We found that during the shock formation, the nonlinear residual tends to oscillate unless the initial guess contains a shock. Use of the above criteria with an oscillating nonlinear residual will cause the linear solution accuracy to oscillate, promoting further nonlinear oscillations that can eventually lead to divergence. For this reason, we establish the

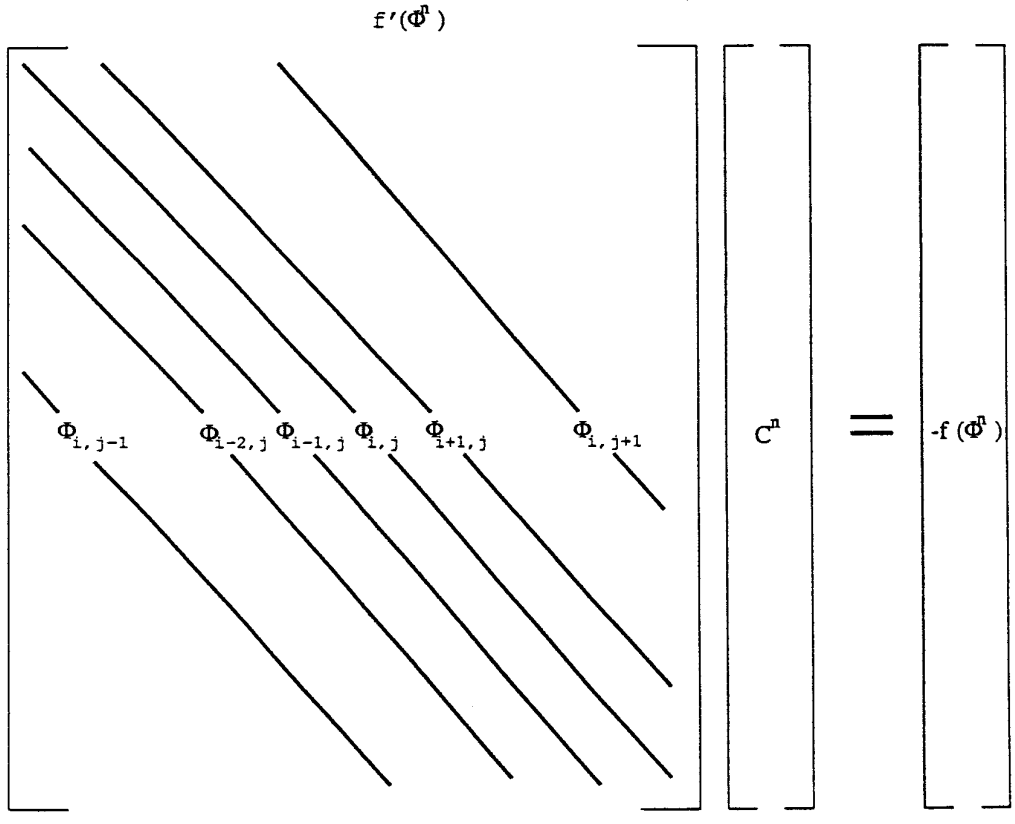

FIG. 1. Six diagonal linear system.

following criteria for the linear solution accuracy:

$$\|\text{Linear Residual}\|_2 \leq \min \begin{cases} \|\text{Nonlinear Residual}\|_2 \\ \text{Specified Value (e.g., } 10^{-1}\text{–}10^{-3}\text{).} \end{cases} \tag{7}$$

The *specified value* condition essentially puts a ceiling on the minimum linear accuracy until a good approximation of the shock is formed and the nonlinear residual stops oscillating. In general, the shock is well formed when the nonlinear residual drops by one to three orders of magnitude, so the specified value is usually set in this range. With this stopping criterion, the convergence of the Newton method should be somewhere between superlinear and quadratic.

3. PRECONDITIONED ITERATIVE METHODS

In solving large sparse linear systems arising in engineering problems, direct methods have traditionally been the method of choice because most engineering problems form nonsymmetric linear systems with high condition numbers which iterative methods were unable to solve. However, over the past two decades many new conjugate gradient-type algorithms have been developed that are capable of solving nonsymmetric linear systems. In addition, the size of many engineering problems has increased to the point that direct methods are memory-intensive and costly. For these reasons, attention has been refocused on the use of iterative methods.

The two iterative methods utilized in this work are the orthogonal s-step Orthomin(k) (OSOmin(s,k)) method of Swanson and Chronopoulos [22] and the GMRES(m) method of Saad and Shultz [18]. Both of these algorithms are Krylov subspace methods based on generalizations of the conjugate gradient method. Both GMRES(m) and OSOmin(s,k) can solve systems in which the coefficient matrix A is nonsymmetric with a symmetric part (i.e., $(A + A^T)/2$) positive definite (i.e., all positive eigenvalues). They also are able to solve some systems in which A is nonsymmetric with the symmetric part indefinite.

In deriving these conjugate gradient-like methods for nonsymmetric systems, different approaches are used. The approach used by the s-step methods, is to truncate the recursion, making the new s direction vectors orthogonal to the previous s direction vectors. Another approach, used by GMRES, uses periodic restarts of the algorithm. There

is a class of other nonsymmetric iterative methods based on a nonsymmetric Lanczos procedure. These include the biconjugate gradient algorithm (BCG) and its stabilized variants (Bi-CGSTAB) [32], the conjugate gradient squared algorithm (CGS) [33, 34], and algorithms based on the quasi-minimum residual idea (QMRCGS) [35]. These methods may be faster for some classes of linear systems. However, in a recent comparison study by McHugh and Knoll [7], it was determined that GMRES is superior to these Lanczos-based methods for inexact Newton solution of two-dimensional Navier–Stokes problems. It is for this reason that we compare the s-step approach to GMRES.

3.1. *OSOmin*

Orthogonal s-step Orthomin is a relatively new algorithm, introduced in 1992 by Swanson and Chronopoulos [22]. Its original basis is the Orthomin(k) method of Vinsome [20]. Orthomin(k) works well for diagonally dominant and slightly nonsymmetric systems, but it tends to break down when the diagonal dominance is lost, or when the degree of nonsymmetry becomes large. In 1991, Chronopoulos [21] introduced an s-step version of Orthomin(k). The s-step version forms, at each iteration, a block of s independent direction vectors using repeated matrix–vector products of the coefficient matrix with a single residual vector [36, 37, 21]. Then the solution is advanced simultaneously using the s direction vectors. The s-step method is more robust than the standard method, giving it the capability to solve more difficult linear systems. In addition, since each s direction can be advanced simultaneously, independently of one another, the method is well-suited for parallel execution. The original version of s-step Orthomin does not maintain orthogonality between the s direction vectors. As a result, the method becomes unstable unless s is small ($s \leq 5$). An alternative approach to the s-step methods is *block methods*.

The block methods use many independent initial residual vectors. In the orthogonal s-step Orthomin approach, a modified Gram–Schmidt method is used to orthonormalize the direction vectors within each block of s-step Orthomin. Although this reduces the data locality properties somewhat and requires slightly more operations, it allows the block size s to be increased up to $s = 16$ without affecting the numerical stability of the method. Using higher values of s makes the method more robust and increases the degree of parallelism. More details of the orthogonal s-step Orthomin (OSOmin(s,k)) algorithm can be found in [22]. The following notation facilitates the description of the algorithm:

- The matrix $P_i = [p_i^1, \ldots, p_i^s]$ (of dimension $n \times s$) is a block of s direction vectors
- The vectors $\alpha_i = [\alpha_i^1, \ldots, \alpha_i^s]^{\mathrm{T}}$ (of dimension s) are the steplengths that minimize $\|r_{i+1}\|_2$ over the affine Krylov subspace

$$\left\{ x_i + \sum_{j=1}^{s} \alpha_j A p_i^j : \alpha_j \text{ scalars} \right\}.$$

- For indices $l = 1, \ldots, s$ and $j_i \leq j \leq i$, $\beta_j^l = [\beta_j^{(l,1)}, \ldots, \beta_j^{(l,s)}]^{\mathrm{T}}$ are vectors (of dimension s) of parameters used in orthogonalizing AP_i against AP_j.

For integers k and i (such that $1 \leq i$), let $j_i = \max(1, i - k + 1)$ for OSOmin(s,k). The algorithm is given below.

ALGORITHM. OSOmin(s,k).

Compute $r_1 = f - Ax_1$.

For $i = 1, \ldots$ **until** convergence **do**

1. Compute $AP_i = [Ar_i, A^2 r_i, \ldots, A^s r_i]$ and set $P_i = [r_i, Ar_i, \ldots, A^{s-1} r_i]$
   If $(1 < i)$ **then**
2. compute $\beta_j^l = [(A^{(l+1)} r_i)^{\mathrm{T}} Ap_j^1, \ldots, (A^{(l+1)} r_i)^{\mathrm{T}} Ap_j^s)]^{\mathrm{T}}$, where $l = 1, \ldots, s$ and $j = j_{(i-1)}, \ldots, i-1$
3. $AP_i = AP_i - \sum_{j=j_{(i-1)}}^{i-1} AP_j [\beta_j^l]_{l=1}^s$
4. $P_i = P_i - \sum_{j=j_{(i-1)}}^{i-1} P_j [\beta_j^l]_{l=1}^s$
   EndIf
5. Apply MGS to the matrix AP_i to obtain final AP_i and P_i
6. Compute $\alpha_i = [r_i^{\mathrm{T}} Ap_i^1, \ldots, r_i^{\mathrm{T}} Ap_i^s]^{\mathrm{T}}$
7. $r_{i+1} = r_i - AP_i \alpha_i$
8. $x_{i+1} = x_i + P_i \alpha_i$

EndFor

3.2. *GMRES*

The second iterative solver used is the generalized minimum residual (GMRES) technique, introduced by Saad and Shultz in 1986 [18]. GMRES is an efficient and robust method that has gained wide acceptance in solving nonsymmetric systems generated in computational fluid dynamics problems. Because the storage grows linearly and computational work quadratically with the dimension of the Krylov subspace, the version most often used is the restarted version, GMRES(m). In this approach, the method is restarted every m steps, limiting the size of the Krylov subspace to m. GMRES(m) is more robust than standard Orthomin(k) but it is theoretically proven that the s-step version of Orthomin(k) has the same convergence properties as GMRES(m) with $s = m$ (see [21]). Therefore, OSOmin(s,k) and GMRES(m) maintain about the same level of robustness for our problem. The algorithm for GMRES(m) is given below.

ALGORITHM. Restarted GMRES(m).

For $k = 1, \ldots$ **until** convergence **do**
1. Compute $r_k = f - Ax_k$, $\beta = \|r_k\|_2$, $v_1 = r_k/\beta$
   For $j = 1, 2, \ldots, m$ **do**
   2. $w_j = Av_j$
   3. **For** $i = 1, \ldots, j - 1$ **do**
      $h_{i,j} = (w_j, v_i)$
      $w_j = w_j - h_{i,j}v_i$
      EndFor
   4. $h_{j+1,j} = \|w_j\|_2$
   5. $v_j = w_j/h_{j+1,j}$
   EndFor
   6. Compute y_m, the minimizer of $\|\beta e_1 - \overline{H}_y\|_2$
   7. $x_{k+1} = x_k + V_m y_m$, where $V_m = [v_1, v_2, \ldots, v_m]$

EndFor

3.3. *Preconditioning*

The convergence rate of iterative schemes is highly dependent on the condition number of the system. Therefore, most iterative solution algorithms implement a preconditioning strategy along with the iterative method. The purpose of the preconditioner is to form a preconditioning matrix P_r such that $AP_r^{-1} \approx I$, clustering the eigenvalues of the system around unity. The transformed system $(AP_r^{-1})P_r x = b$ (i.e., right preconditioned system) has a lower condition number and is solved more quickly by the iterative method. One further requirement is that P_r^{-1} must be easy to compute. An effective preconditioner can also increase the robustness of the iterative method because it can transform insolvable systems with high condition numbers into systems with lower condition numbers that are within the limits of the iterative solver.

We use a right preconditioning approach based on the ILU factorization method, developed by Meijerink and Van Der Vorst [38]. Zero fill-in is used (i.e., ILU(0)). Preconditioning reduces the number of iterations in the iterative method considerably (by a factor of about three in our tests) but the preconditioner itself can have very slow execution rates if not implemented properly.

An efficient vectorized implementation of ILU was introduced by Van der Vorst [25]. The approach uses a Von Neumann series expansion approximation for the L and U inversions, truncating the expansion at the first-order term. This allows the method to be vectorized in blocks of length $\sqrt{n}$, for an $n \times n$ system (n = no. of gridpoints). The convergence rate of the iterative method with the vectorized ILU preconditioner is slightly worse than with original ILU, but the execution rate on a vectorized processor is about three times faster.

While the vectorized ILU approach is efficient on vector processors, it does not exhibit good parallel performance. This is due to a recursion between vectorized blocks. That is, block i uses the results of block $i - 1$. One could parallelize the method by stripmining the vectorized blocks, but since the length of the blocks are only length $\sqrt{n}$, this approach can be inefficient. An efficient parallel implementation of ILU was proposed by Di Brozolo and Robert [27]. The original preconditioning matrix is broken up into *nblocks* submatrices, ignoring the original relation between the submatrices. Each submatrix is then executed independently on a different processor. The loss of connection of the submatrices is accounted for by introducing an overlapping region and taking the average of the computed values by the subsystems in the overlapping regions. We found an optimal overlapping region of length one. This technique is illustrated with *nblocks* = 4 in Fig. 2.

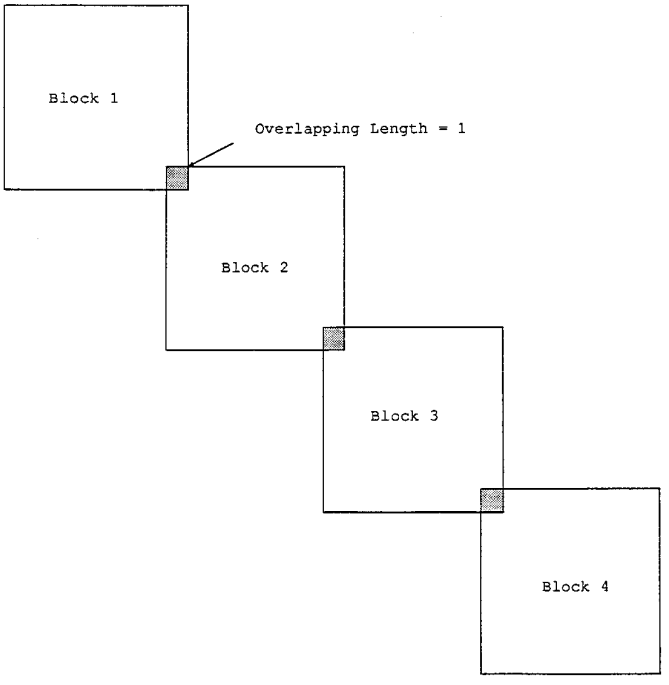

FIG. 2. DiBrozolo block ILU preconditioner.

If *nblocks* = 1, this preconditioner is exactly the same as the vectorized ILU preconditioned method. The effectiveness of ILU decreases as the size of *nblocks* increases. Thus, there is a trade-off between improved parallel performance and reduced convergence rate with use of this approach. Generally, the preconditioner begins to break down with large values of *nblocks* (i.e., *nblocks* ≥ 8) so the Di Brozolo parallelization is efficient only on a small number of processors.

4. RESULTS

The Newton–iterative method is compared to the more traditional approximate factorization/ADI MAF method of Goorjian [26] on two modern supercomputer architec-

TABLE I

Storage Requirements for MAF and Newton Algorithms

MAF	$6n$
Newton–OSOmin(s,k)	$[9 + 2(k + 1)s]n$
Newton–GMRES(m)	$[9 + (m + 2)]^n$

tures; a vector-parallel Cray YMP C-90 and a massively parallel Thinking Machine CM-5. We test on eight processors of the C-90 and each processor has a peak execution rate of 1 Gflop per processor, giving a peak rate of 8 Gflops. The CM-5 has 512 processors with a peak execution rate of 125 Mflops per processor, giving it an overall peak execution rate of 64 Gflops. Both methods (i.e., Newton and MAF) are implemented to solve a test problem, computing the transonic flowfield over a NACA 0012 airfoil. The methods are run to the same exit accuracy level, so their solutions are exactly the same. The execution time and rate are measured using the hardware performance monitor on the Cray.

The memory requirements of MAF and the Newton–iterative methods are significantly different, and must be taken into consideration for the implementation. With n gridpoints, MAF requires storage of the solution vector, the correction vector, and four vectors used to store the tridiagonal systems during vectorized ADI sweeps, totaling $6n$. The Newton method requires storage of the six diagonals of the linear system, the right-hand side, and the solution and correction vectors, totaling $9n$. In addition, the Newton–iterative method requires storage for the preconditioned iterative solvers. This amounts to $[2(k + 1)s]n$ for OSOmin(s,k)and $(m + 2)n$ for GMRES(m). Details on the derivation of these storage requirements are given in [22] for OSOmin and in [18] for GMRES. The overall storage requirements of MAF and the Newton–iterative methods are given in Table I. The values of s, k, and m vary for different cases tested, so the storage requirements for the Newton–iterative methods are problem dependent.

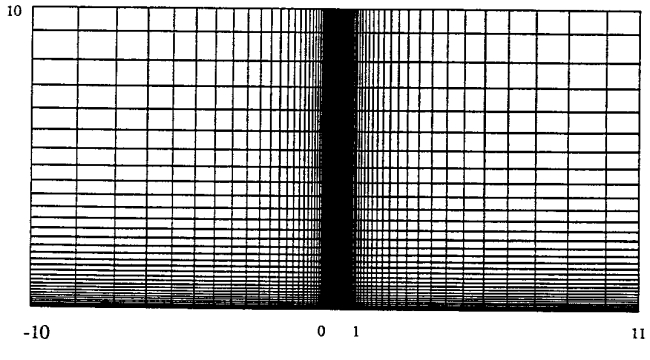

FIG. 3. Solution mesh (64 × 32 shown for illustration purposes, but actual meshes used are 256 × 128 and 512 × 512).

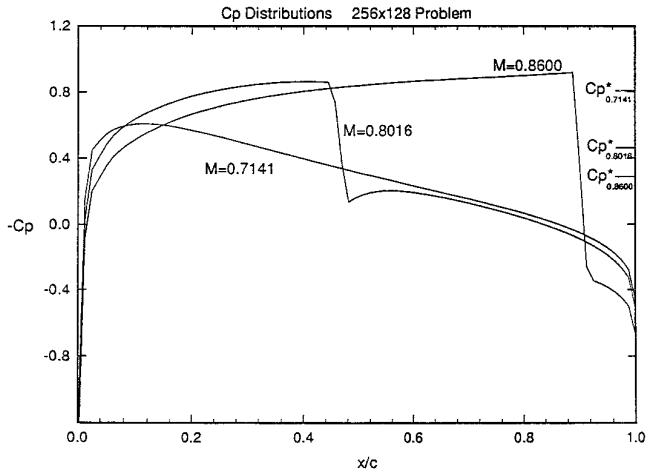

FIG. 4. C_p Distributions over NACA 0012 airfoil for cases tested on 256 × 128 grid.

A test problem is chosen which is representative of a typical transonic aerodynamic calculation. A NACA 0012 airfoil at zero angle of attack is placed in a computational domain extending 10 chordlengths in front of and behind the airfoil. A mesh is overlaid on the domain which is nonuniform in both the x and y directions, except on the airfoil where uniform spacing in x is used. Since the NACA 0012 is a symmetric airfoil, we only compute on the upper half of the computational domain. This simple symmetric test case was chosen arbitrarily and we expect the method would have no difficulties for a lifting case.

The solution mesh is shown in Fig. 3. The boundary conditions applied are no perturbations ($\Phi = 0$) along all edges of the mesh, except on the bottom edge, where the flow tangency condition is applied on the airfoil (i.e., $\partial\Phi/\partial y = \partial f/\partial x$, where $f(x)$ is the airfoil shape function for NACA 0012) and $\partial\Phi/\partial y = 0$ elsewhere on the x axis. Once the steady potential field is solved, the C_p distribution on the airfoil can be determined by

$$C_p = -2\tau^{2/3}\Phi_x, \tag{8}$$

where Φ_x is the derivative in the x direction along the airfoil and is approximated with finite differences. This definition for C_p comes from the small disturbance approximation.

Results from three different implementations of the Newton method and MAF to solve the above problem are presented in the remaining portions of this section. In the first set of tests, a 256 × 128 mesh is used and three different transonic Mach numbers are tested. The iterative solvers use ILU preconditioning. Results are given on single and

TABLE II

Single Processor Execution Times and Rates (on Cray C-90) with 256 × 128 Mesh

Method	$M_\infty = 0.7141$	$M_\infty = 0.8016$	$M_\infty = 0.8600$
MAF	6.1 s/143 Mflops	13.5 s/143 Mflops	67.5 s/143 Mflops
Newton–OSOmin	5.0 s/538 Mflops	8.4 s/549 Mflops	22.1 s/685 Mflops
Newton–GMRES	13.1 s/501 Mflops	18.7 s/504 Mflops	24.1 s/510 Mflops

Note. Iterative solvers use ILU preconditioning, nonlinear residual converged by eight orders of magnitude.

multiple processors of the Cray C-90. The second set of tests is similar to the first, except that a finer 512 × 512 mesh is used and lower Mach numbers are tested (the transonic cases become more difficult with the finer mesh due to better shock definition). The third set of tests are done using the same 512 × 512 mesh but with unpreconditioned versions of the iterative solvers. The purpose of this is to, first, show the improvement in convergence that preconditioning adds and, second, to study whether the better parallel performance of the unpreconditioned solvers outweighs their slowness on one processor. Results from this third set of tests are presented both from the Cray C-90 and the Thinking Machine CM-5.

4.1. 256 × 128 *Mesh*

Three different Mach numbers are tested with the 256 × 128 mesh. The first is $M_\infty = 0.7141$, corresponding to a similarity parameter of $K = 1.76$. This flow is completely subsonic, giving a smooth C_p distribution over the airfoil. The second is $M_\infty = 0.8016$ (K=1.10), which is moderately transonic for this airfoil, causing weak shocks to form and disrupt the smooth C_p distribution. A substantial portion of the flowfield is subsonic, though. The third Mach number tested is $M_\infty = 0.8600$ ($K = 0.73$). This flow forms strong shocks and is the most difficult of the three cases. The C_p distributions over the NACA 0012 airfoil for these three Mach numbers are shown in Fig. 4. The sonic condition C_p^* for the three cases is also indicated in Fig. 4.

There are several parameters in the iterative methods which can be adjusted to optimize their convergence. In MAF, the size of the geometric sequence g can be adjusted. In GMRES(m), the size of the Krylov subspace m is adjusted, and in OSOmin(s,k), the number of s directions along with the value of k is adjusted. For the single processor tests, these parameters are set to give the fastest single processor convergence time. The original vectorized form of ILU (i.e., *nblocks* = 1) is used for preconditioning with the iterative methods. The optimal parameters for the single processor tests are $g = 6$, $m = 10$, $s = 2$, and $k = 1$ for the $M_\infty = 0.7141$ case; $g = 9$, $m = 12$, $s = 3$, and $k = 2$ for the $M_\infty = 0.8016$ case; and $g = 9$, $m = 17$, $s = 15$, and $k = 2$ for the $M_\infty = 0.8600$ case. The methods are considered converged when the nonlinear residual 2-norm is reduced by eight orders of magnitude. Execution rates and times are shown in Table II, and convergence plots are shown in Figs. 5–7.

Newton–OSOmin outperforms both MAF and Newton–GMRES for all three Mach numbers. For the two lower Mach numbers, Newton–GMRES is slower than MAF while Newton–OSOmin is about 1.2 and 1.6 times faster, respectively. For the $M_\infty = 0.8600$ case, both Newton–OSOmin and Newton–GMRES are about 3 times faster than MAF. The execution rates of the Newton–iterative methods are between 3.5 and 5 times faster than MAF. Since the speedups in execution time are lower than this, the total amount of computational work performed by the Newton method is more than MAF. The speed of the Newton methods is due to their efficient vector performance. They would be slower on a serial machine.

The convergence plots show that the Newton methods converge smoothly for the subsonic case without shocks (Fig. 5), but they exhibit oscillations in the initial stages for the two transonic cases with shocks (Figs. 6 and 7). This oscillatory behavior is due to use of a freestream

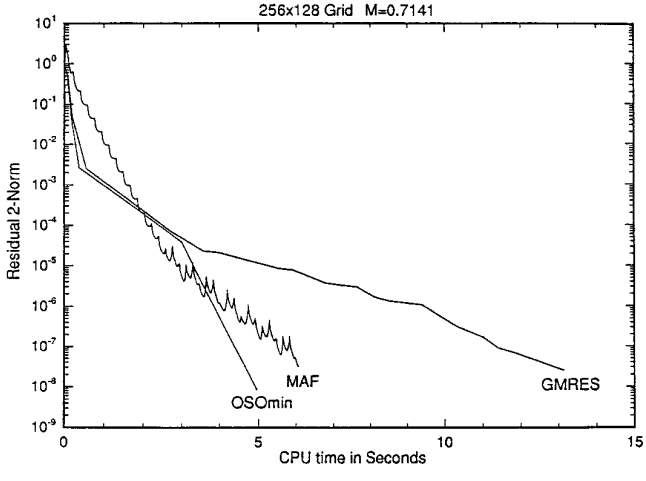

FIG. 5. Residual vs. CPU time measured on single processor of Cray C-90 for $M_\infty = 0.7141$ case on 256 × 128 mesh.

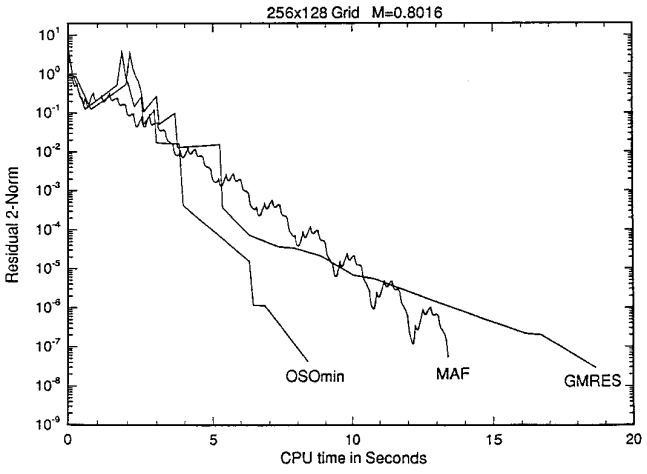

FIG. 6. Residual vs CPU time measured on single processor of Cray C-90 for $M_\infty = 0.8016$ case on 256×128 mesh.

condition (i.e., without a shock) for the initial condition in the Newton method. For cases with shocks, this initial guess is poor and oscillations persist until a good approximation of the shock is made. The oscillations are more abundant with a finer mesh. Venkatakrishnan [6, 9] developed a strategy to overcome this problem by using mesh sequencing, whereby an initial guess is determined on a coarse mesh and extrapolated to progressively finer meshes. This reduced the oscillations considerably. Since our work focuses on the efficiency of the iterative methods more than on the Newton method itself, we did not incorporate this approach. However, this is certainly an idea for future implementation of this work.

We next test the methods on multiple processors of the C-90. The Di Brozolo version of ILU [27] is used with *nblocks* = 4. It should be noted that larger values of *nblocks* were tried but this caused the preconditioner to become ineffective to the point that the convergence rate became very poor. With *nblocks* = 4, the robustness of ILU is decreased but not substantially. As a result of the decreased robustness in ILU, the iterative methods require more time to converge and the optimal values of the parameters m, s, and k in GMRES(m) and OSOmin(s,k) are different from the single processor tests. For the $M_\infty = 0.7141$ case, the optimal parameters are $m = 12$, $s = 4$, and $k = 1$. For $M_\infty = 0.8016$, they are $m = 11$, $s = 6$, and $k = 1$, and for $M_\infty = 0.8600$, they are $m = 17$, $s = 16$, and $k = 1$. The methods are again considered converged when the nonlinear residual drops by eight orders of magnitude. It should be noted that the version of MAF which we used (i.e., the original version proposed by Goorjian [26]) has a recursion in the ADI line sweeps, when sweeping in the mean flow direction, and this is the reason for the poor parallel performance. The parallel performance could be improved by eliminating this recursion, but the convergence rate then becomes slower. The execution time and speedup for the three methods on 1, 4, and 8 processors of the C-90 are shown in Table III.

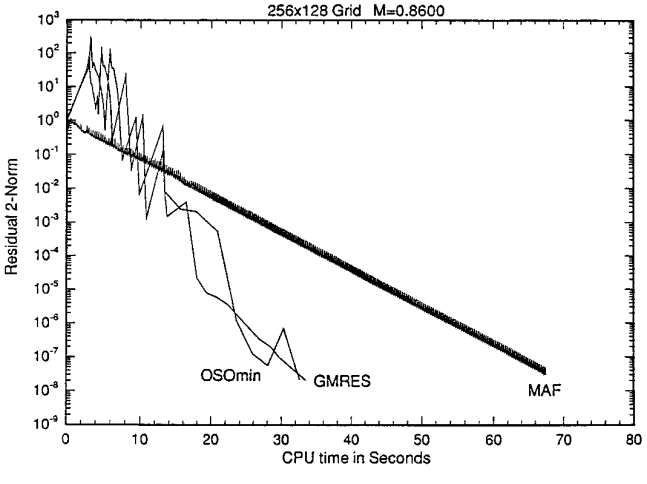

FIG. 7. Residual vs CPU time measured on single processor of Cray C-90 for $M_\infty = 0.8600$ case on 256×128 mesh.

The Newton–iterative methods achieve relatively good parallel speedups while MAF achieves little speedup. A plot of the parallel speedups is shown in Fig. 8. The speedup of Newton–OSOmin is greatly affected by the value of s. In the subsonic case, where $s = 4$, the speedups are small in comparison to the most difficult transonic case, where $s = 16$. The speedups of Newton–GMRES remain relatively constant for all three cases, but are slightly better for lower m. This is probably due to the minimization step in GMRES, where a least-squares solution of the small $m \times m$ system is performed. Since this step is performed serially, larger values of m reduce the parallelism slightly.

The iterative methods by themselves achieve better parallel speedups. What slows the method is the preconditioning. The setup of the L and U factors in ILU is completely serial and must be done for each new linear system in each Newton iteration. On a single processor, this step requires only 5–7% of the CPU time and is relatively insignificant, but on eight processors it requires 15–20% and causes a significant reduction in the parallel speedups. In addition, only four blocks were used in the Di Brozolo decomposition of the preconditioning matrix, due to a lack of robustness with *nblocks* > 4. While this is efficient on four processors, the speedup is reduced on eight processors. The combination of these two effects cause a falloff in speedups after four processors, as is apparent in Fig. 8.

The good parallel performance of the Newton methods

TABLE III

Multiple Processor Execution Times and Speedups (Mflops on Single Processor) on Cray C-90 with 256 × 128 Mesh

Method	$M_\infty = 0.7141$	$M_\infty = 0.8016$	$M_\infty = 0.8600$
1 *Processor*			
MAF	6.1 s/143 Mflops	13.5 s/143 Mflops	67.6 s/143 Mflops
Newton–OSOmin	6.0 s/520 Mflops	13.8 s/551 Mflops	32.6 s/606 Mflops
Newton–GMRES	14.0 s/453 Mflops	22.3 s/471 Mflops	33.5 s/456 Mflops
4 *Processors*			
MAF	5.9 s/1.03	13.1 s/1.03	65.6 s/1.03
Newton–OSOmin	2.0 s/3.07	4.2 s/3.29	9.8 s/3.34
Newton–GMRES	5.2 s/2.71	7.6 s/2.92	12.4 s/2.70
8 *Processors*			
MAF	6.0 s/1.01	13.3 s/1.02	66.9 s/1.02
Newton–OSOmin	1.7 s/3.51	3.5 s/3.89	7.7 s/4.23
Newton–GMRES	4.9 s/2.84	7.3 s/3.05	11.6 s/2.87

Note. Iterative solvers use Di Brozolo ILU preconditioning with *nblocks* = 4, nonlinear residual converged by eight orders of magnitude.

makes them considerably more efficient than MAF on multiple processors. Figure 9 shows a histogram plot of the ratio in speedup of the CPU times of the Newton methods over MAF. On a single processor, Newton–OSOmin is 1.6 to 2.3 times faster than MAF, but on eight processors it is up to 8.7 times faster. Newton–GMRES is slower than MAF for two cases on a single processor, but it is up to 5.8 times faster on eight processors.

4.2. 512 × 512 *Mesh*

The next group of tests are performed using a finer 512 × 512 solution mesh. The grid domain and layout is the same as the 256 × 128 mesh, just twice as fine in the x direction and four times as fine in the y direction. Two Mach numbers are tested with this mesh; $M_\infty = 0.7141$ ($K = 1.76$) and $M_\infty = 0.7850$ ($K = 1.215$). The first is subsonic and corresponds to the subsonic case for the 256 × 128 mesh. The second is slightly transonic. One effect of using a finer mesh is that the shock definition is better. In our tests, we find that the better shock definition yields linear systems that tend to be more difficult to solve. Consequently, the $M_\infty = 0.7850$ case for the 512 × 512 mesh is about as difficult to solve as the $M_\infty = 0.8016$ case for the 256 × 128 mesh. The C_p distributions for these two Mach numbers are shown in Fig. 10.

We first present the cases showing optimal convergence on a single processor. Original ILU preconditioning is used

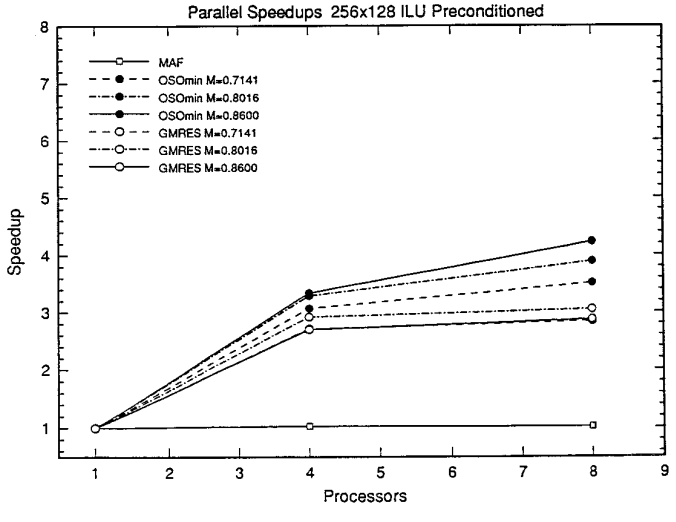

FIG. 8. Parallel speedups on Cray C-90 for Di Brozolo ILU preconditioned cases on 256 × 128 grid.

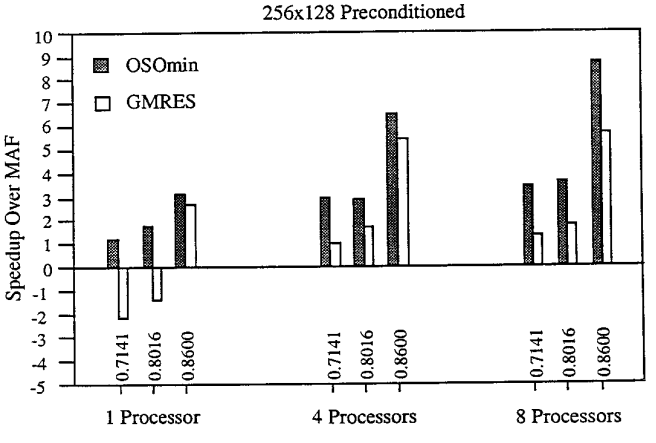

FIG. 9. Ratio of CPU times for Newton–iterative methods to MAF for 256 × 128 problem.

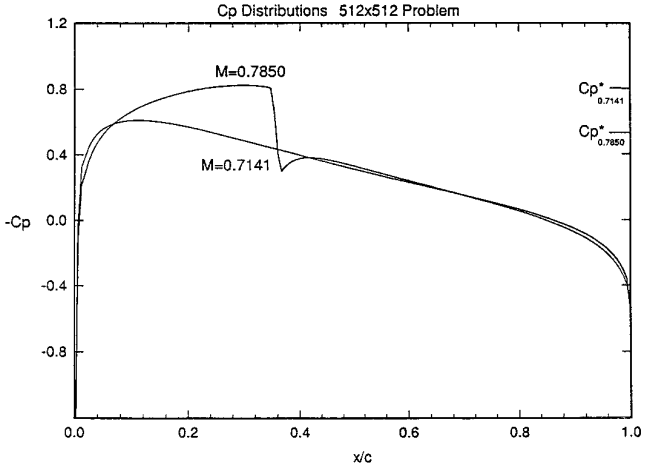

FIG. 10. C_p distributions over NACA 0012 airfoil for two cases tested on 512 × 512 grid.

(i.e., *nblocks* = 1) and the parameters in the methods are tuned to give convergence is the fastest execution time. The optimal parameters for the $M_\infty = 0.7141$ case are $m = 14$ for GMRES(m), and $s = 1$, $k = 1$ for OSOmin(s,k). For the $M_\infty = 0.7850$ case, they are $m = 10$, $s = 6$, and $k = 2$. Because this finer mesh total yields a factor of 8 increase in the total number of gridpoints, the methods are exited at an earlier level than the 256 × 128 tests, to avoid high CPU costs. The methods are exited when the residual 2-norm has been reduced by four orders of magnitude rather than eight, as was used in the 256 × 128 tests. Execution rates and times of the methods for these two Mach numbers are shown in Table IV.

The Newton–OSOmin algorithm again outperforms both MAF and Newton–GMRES in execution time. Although it is 2.6 times faster than MAF for the subsonic case (twice that in the 256 × 128 tests), Newton–OSOmin is only 1.1 times faster for the transonic case. This conflicts with results from the 256 × 128 tests where the iterative methods fared much better for the transonic cases. This is believed to be due to the freestream initial guess for Newton, which causes oscillations for the 256 × 128 case and enhances them for this finer 512 × 512 case. As was discussed in the previous subsection, the convergence could probably be improved by using a mesh sequencing strategy in the Newton iterations.

TABLE IV

Single Processor Execution Times and Rates on Cray C-90 with 512 × 512 Mesh

Method	$M_\infty = 0.7141$	$M_= = 0.7850$
MAF	17.3 s/167 Mflops	54.9 s/167 Mflops
Newton–OSOmin	6.5 s/460 Mflops	47.1 s/539 Mflops
Newton–GMRES	20.7 s/538 Mflops	73.5 s/507 Mflops

Note. Iterative solvers use ILU preconditioning, nonlinear residual converged by four orders of magnitude.

TABLE V

Multiple Processor Execution Times and Speedups (Mflops on Single Processor) on Cray C-90 with 512 × 512 Mesh

Method	$M_\infty = 0.7141$	$M_= = 0.7850$
1 *Processor*		
MAF	17.5 s/167 Mflops	54.6 s/167 Mflops
Newton–OSOmin	12.0 s/546 Mflops	65.3 s/475 Mflops
Newton–GMRES	21.0 s/522 Mflops	79.9 s/502 Mflops
4 *Processors*		
MAF	16.8 s/1.04	52.5 s/1.04
Newton–OSOmin	3.9 s/3.04	19.3 s/3.39
Newton–GMRES	6.9 s/3.05	28.2 s/2.83
8 *Processors*		
MAF	16.7 s/1.05	52.0 s/1.05
Newton–OSOmin	3.43 s/3.49	17.6 s/3.71
Newton–GMRES	6.21 s/3.38	25.1 s/3.19

Note. Iterative solvers use Di Brozolo ILU preconditioning with *nblocks* = 4, nonlinear residual converged by four orders of magnitude.

Table V presents results of tests on multiple processors of the C-90 for the 512 × 512 case. As was the case in the 256 × 128 tests, the Di Brozolo version of ILU is used with *nblocks* = 4 for all cases, resulting in slightly slower convergence of the iterative methods and a change in the optimal convergence parameters for the iterative methods. The values of the parameters m, s, and k that give optimal convergence for the $M_\infty = 0.7141$ case are $m = 11$, $s = 3$, and $k = 1$, and for $M_\infty = 0.7850$, they are $m = 9$, $s = 5$, and $k = 1$. Plots of the parallel speedups are shown in Fig. 11.

The parallel speedups are slightly better than the 256 × 128 cases, due to the larger vector lengths in the iterative methods. The better parallelism of the Newton methods again cause a substantial improvement in CPU time over MAF. A histogram plot showing the speedup in CPU times of the Newton methods over MAF is shown in Fig. 12. On a single processor, Newton–OSOmin is 1.0–2.7 times faster than MAF, and on eight processors it is 3.0–5.3 times faster. Newton–GMRES is slower than MAF on one processor, but is 2.0–2.7 times faster on eight processors.

4.3. 512 × 512 *Mesh—Unpreconditioned Solvers*

While preconditioning can improve the convergence rate of the iterative methods substantially, causing fewer iterations, it also inhibits the parallel performance. Thus, there is a trade-off between reduced computational work and

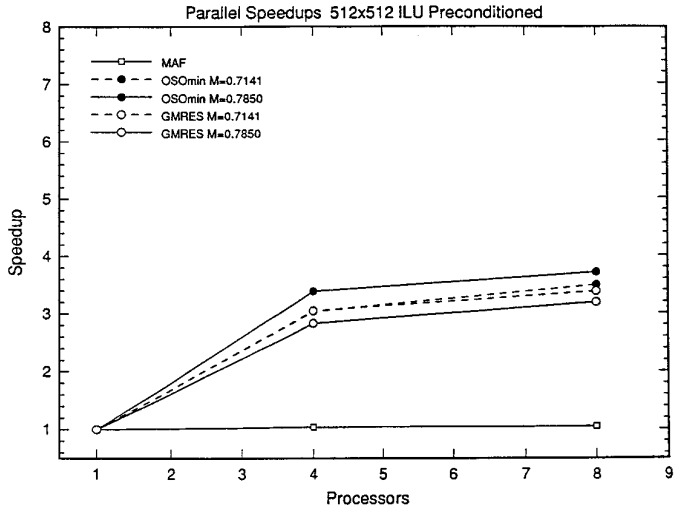

FIG. 11. Parallel speedups on Cray C-90 for Di Brozolo ILU preconditioned cases on 512 × 512 grid.

TABLE VI

Multiple Processor Execution Times and Speedups (Mflops on Single Processor) on Cray C-90 with 512 × 512 Mesh

Method	$M_\infty = 0.7141$	$M_= = 0.7850$
1 *Processor*		
MAF	10.4 s/167 Mflops	34.2 s/167 Mflops
Newton–OSOmin	20.8 s/702 Mflops	104.4 s/783 Mflops
Newton–GMRES	24.1 s/634 Mflops	107.2 s/644 Mflops
4 *Processors*		
MAF	10.0 s/1.04	32.9 s/1.04
Newton–OSOmin	5.4 s/3.84	26.6 s/3.92
Newton–GMRES	6.8 s/3.57	29.5 s/3.64
8 *Processors*		
MAF	9.9 s/1.05	32.6 s/1.05
Newton–OSOmin	3.3 s/6.26	15.3 s/6.84
Newton–GMRES	4.6 s/5.28	18.8 s/5.71

Note. Unpreconditioned, iterative solvers used, nonlinear residual converged by two orders of magnitude.

improved parallel efficiency. In this group of tests, we seek to investigate this trade-off. Unpreconditioned versions of OSOmin(s,k) and GMRES(m) are used in the Newton method and compared to MAF for the same 512 × 512 test cases used in the previous subsection.

Without preconditioning, the robustness of the iterative methods is reduced considerably. Particularly large values of m in GMRES(m) and s in OSOmin(s,k) had to be used for the transonic case. The optimal parameters for the $M_\infty = 0.7141$ case are $m = 12$, $s = 6$, and $k = 1$, and for $M_\infty = 0.7850$ they are $m = 25$, $s = 16$, and $k = 2$. Even with the large increases in m and s, we still found that the unpreconditioned solvers did not converge for the difficult systems that arise in the latter part of the Newton solution. Consequently, the unpreconditioned tests are converged by two orders of magnitude rather than four, which was used in the earlier tests.

Timings on multiple processors of the C-90 for these cases are shown in Table VI. The parallelism of the unpreconditioned solvers is considerably better than the preconditioned cases, as is apparent in the plot of parallel speedups for these cases shown in Fig. 13. This indicates that the bottleneck of parallelization found in the preconditioned tests in the previous sections is the fault of the preconditioner, not of the iterative solvers themselves or the routines associated with the Newton method.

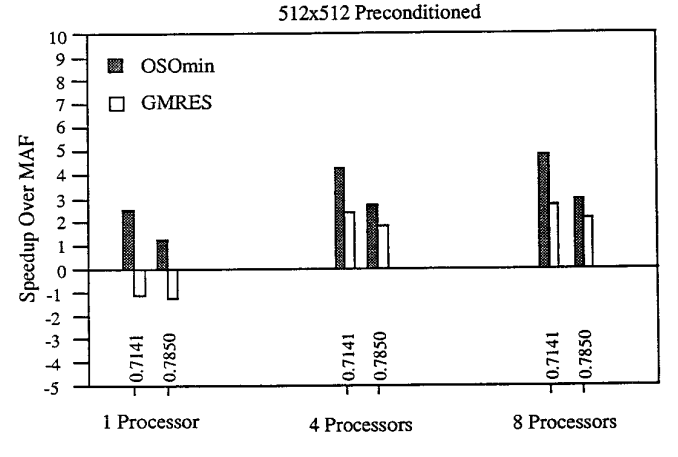

FIG. 12. Ratio of CPU times for Newton–iterative methods to MAF for 512 × 512 problem.

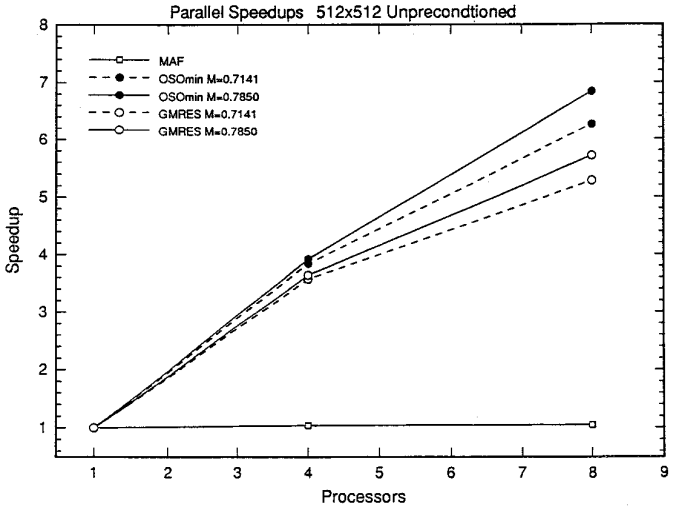

FIG. 13. Parallel speedups on Cray C-90 for unpreconditioned cases on 512 × 512 grid.

A histogram comparing the execution time of the Newton method to MAF is given in Fig. 14. On a single processor, the unpreconditioned iterative methods are about 2.5 times slower than MAF for the subsonic case and about 3.4 times slower for the transonic case. This amounts to being about one-third the speed of the preconditioned cases. However, the slowness on one processor is overcome by good parallel speedups so that on eight processors, the unpreconditioned methods are between 2 and 3 times faster than MAF. While this is not as good as the 2–5 times speedup over MAF from the preconditioned cases, it is still comparable. We expect the unpreconditioned methods to be faster than the preconditioned methods when 16 or more processors are used. Of course, robustness of the unpreconditioned solvers is still an issue.

The Newton method with unpreconditioned solvers is also implemented in data-parallel on the massively parallel Thinking Machine CM-5. The data-parallel programming paradigm is essentially single instruction multiple data (SIMD) and requires the code to be rewritten to a high performance Fortran-type language (i.e., CMFortran). We did not implement the ILU preconditioner in data-parallel, as this is more difficult to implement efficiently. We also did not implement the MAF code on the CM-5. Timings on 64 processors of the CM-5 are given in Table VII.

The execution rates of the Newton method with unpreconditioned solvers on 64 processors of the CM-5 are quite fast. Scaled to the full machine configuration of 512 processors, the execution rates are 8–9 Gflops for Newton–OSOmin and 6–7 Gflops for Newton–GMRES. On eight processors of the C-90, the execution rates for these cases are 4–5 Gflops for Newton–OSOmin and 3–4 Gflops for Newton–GMRES. Thus, the Newton–iterative approach is well suited for massively parallel execution. Higher execution rates are expected on the CM-5 when finer or three-dimensional meshes are used, since the current mesh used does not take full advantage of the machine's capabilities.

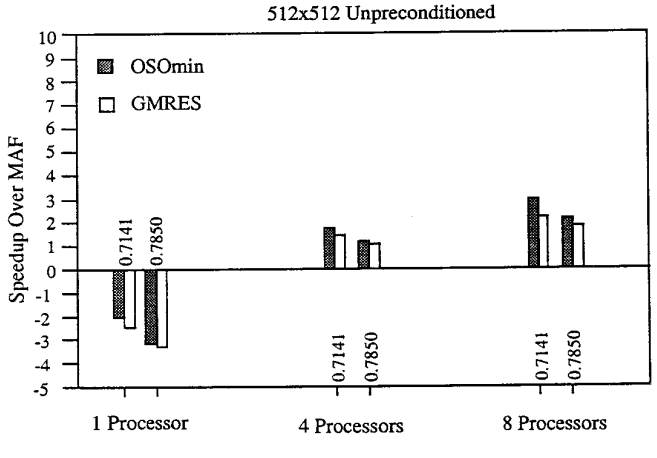

FIG. 14. Ratio of CPU times for Newton–iterative methods to MAF for 512 × 512 unpreconditioned problem.

TABLE VII

Execution Times and Rates of the Newton–Iterative Methods on 64 Processors of the Thinking Machine CM-5

Method	$M_\infty = 0.7141$	$M_= = 0.7850$
Newton–OSOmin	15.0 s/975 Mflops	72.3 s/1131 Mflops
Newton–GMRES	19.2 s/796 Mflops	77.4 s/892 Mflops

Note. Unpreconditioned iterative solvers used, nonlinear residual converged by two orders of magnitude.

5. CONCLUDING REMARKS

An inexact Newton method is used to solve the steady two-dimensional transonic small disturbance equation. Conjugate gradient-like iterative methods are used to solve the large sparse linear systems in each Newton iteration. The two iterative methods utilized are a block s-step solver, orthogonal s-step Orthomin (OSOmin), and the more popular GMRES method. The preconditioning used is a vectorized and parallelized form of ILU. The Newton method with these two solvers is compared against a more traditional approximate factorization implicit solution method MAF in computing the transonic flow over a NACA 0012 airfoil. The efficiency of the methods is the main issue addressed. The 2D TSD equation is only a test case for the viability of out approach. We intend to use this approach for more complicated problems.

In tests on a single processor of the Cray C-90, Newton–OSOmin shows the best results, requiring less CPU time than either MAF or Newton–GMRES for subsonic and moderately transonic cases. Newton–GMRES is slower than MAF for low Mach number cases, but it is faster for strong shock cases. The Newton methods require more work than MAF, but they are faster due to the efficiency of the iterative solvers on the vector architecture. Oscillations in the residual occur in the first several iterations during convergence of the Newton methods for cases with shocks. This results from the absence of a shock in the initial condition. The convergence is very rapid once a reasonably good approximation of the shock is made. Overall results show Newton–OSOmin is 1–3 times faster than MAF for various Mach numbers on a 256 × 128 mesh, while Newton–GMRES ranges from 2 times slower to 3 times faster for the same cases. With a finer 512 × 512 mesh, Newton–OSOmin is 1.2 to 2.4 times faster than MAF but Newton–GMRES is 1–2 times slower. It is ex-

pected that these factors would increase for the finer mesh if a mesh sequencing strategy, in which a coarse mesh is used to generate an initial guess (similar to that proposed by Venkatakrishnan [9]), were used. In tests with unpreconditioned solvers, we find that the Newton methods are 2–3 times slower than MAF on a single processor, indicating that use of the ILU preconditioning reduces the computational workload by a factor of about 3.

On multiple processors of the Cray C-90, the Newton methods show good parallel speedups. The Newton methods with preconditioned solvers show speedups ranging from 3 to 4.5 on eight processors. The parallel speedups of Newton–OSOmin are slightly better than Newton–GMRES, but both methods show good parallel performance. MAF shows almost no parallel speedup. The speedups with the larger 512×512 problem are better than with the 256×128 problem, because of the longer vector lengths in the iterative solvers. The good parallel performance of the Newton methods make them up to nine times faster than MAF on eight processors. In tests with unpreconditioned versions of the iterative solvers, the parallel speedup is increased substantially (i.e., 5.3–6.8), indicating that the preconditioner is the portion of the method preventing better parallelization. Although the unpreconditioned solvers are slower than MAF on one processor, their good parallel speedups make them up to three times faster on eight processors. It is expected this factor will increase if more than eight processors are used.

Finally, in tests on the massively parallel Thinking Machine CM-5 the Newton method with unpreconditioned iterative solvers attains very fast execution rates. Scaled to the full 512 processor configuration, Newton–OSOmin achieves an execution rate of 8–9 Gflops while Newton–GMRES achieves 6–7 Gflops. This is about two times as fast as the execution rate found on eight processors of the C-90. Higher execution rates are expected on the CM-5 when finer or three-dimensional meshes are used. Therefore the use of the Newton–iterative approach for large problems on massively parallel computers is promising. An area that needs further work, however, is the development of parallel preconditioners for massively parallel computers.

ACKNOWLEDGMENTS

The first author was supported by a NASA Graduate Student Fellowship. This work was supported by allocation grants from the Minnesota Supercomputer Institute (MSI) and Cray Research, Inc. The work is also supported in part by NSF Grant CCR-9496327 and by the Army Research Office Contract DAAL03- 89-C-0038 with the University of Minnesota Army High Performance Computing Research Center (AHPCRC) and the DOD Shared Resource Center at the AHPCRC. The authors wish to acknowledge CRAY Research, Inc. for supplying the dedicated computer facilities and for the assistance provided by Mr. Charles Swanson in performing the dedicated runs.

REFERENCES

1. M. Hafez, and S. Paliswamy, ''Calculations of Transonic Flows with Shocks Using Newton's Method and a Direct Solver. Part I. Potential Flows,'' in *Advances in Computer Methods for Partial Differential Equations, VI*, edited by R. Vichnevesky and R. Stepleman (IMACS, Basel, 1987), p. 507.
2. E. E. Bender and P. K. Kholsa, AIAA Paper No. 87-0603, 1987 (unpublished).
3. M. Giles, M. Drela, and W. T. Thomkins, ''Newton Solution of Direct and Inverse Transonic Euler Equations,'' in *Proceedings, AIAA 7th Computational Fluid Dynamics Conference, 1985*, p. 394.
4. L. B. Wigton, ''Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations,'' in *Proceedings, AIAA 8th Computational Fluid Dynamics Conference, 1987*, p. 444.
5. M. Hafez, S. Paliswamy, and P. Mariani, AIAA Paper No. 88-0226, 1988 (unpublished).
6. V. Venkatakrishnan, *AIAA J.* **27**(7), 885 (1989).
7. P. R. McHugh, and D. A. Knoll, *AIAA J.* **32**(12), 2394 (1994).
8. K. Ajmani, M. S. Liou, and R. Dyson, AIAA Paper No. 94-0408, 1994 (unpublished).
9. V. Venkatakrishnan, *AIAA J.* **29**(7), 1092 (1991).
10. K. Ajmani, W. F. Ng, and M. S. Liou, AIAA Paper No. 93-0881, 1993 (unpublished).
11. P. D. Orkwis, and D. S. McRae, *AIAA J.* **30**(1), 78 (1992)
12. P. D. Orkwis, and D. S. McRae, *AIAA J.* **30**(6), 1507 (1992).
13. D. E. Keyes, *ICASE Res. Quart.* **4**(1), 4 (1995).
14. R. G. Melvin, D. P. Young, D. E. Keyes, C. C. Ashcraft, M. B. Bieterman, C. L. Hilmes, W. P. Huffman, and F. T. Johnson, BCSTECH 94-047, Boeing Computer Services, Seattle, 1994 (unpublished).
15. D. Drikakis, E. Schreck, and F. Durst, *J. Fluids. Eng.* **116,** 835 (1994).
16. R. A. E. Makinen, *Finite Elements Fluids,* 457 (1993).
17. H. Jiang, and P. Forsyth, *Computers and Fluids,* Vol. 24, No. 7, pp. 753 (1995).
18. Y. Saad and M. Shultz, *SIAM J. Sci. Statist. Comput.* **7,** 856 (1986).
19. V. Venkatakrishnan, and D. J. Mavriplis, ''Implicit Solvers for Unstructured Meshes,'' in *Proceedings, AIAA 10th Computational Fluid Dynamics Conference, 1991,* p. 115.
20. P. K. W. Vinsome, *Soc. Petroleum Eng. AIME,* **SPE 5729** (1976).
21. A. Chronopoulos, *SIAM J. Numer. Anal.* **28,** 6 (1991).
22. C. Swanson and A. Chronopoulos, ''Orthogonal s-Step Iterative Methods Implemented on a Parallel Computer,'' in *Proceedings, ACM Int. Conf. on Supercomputing, 1992,* p. 456.
23. W. E. Ballhaus, A. Jameson, and J. Albert, *AIAA J.* **16,** 573 (1978).
24. A. S. Lyrintzis, A. M. Wissink, and A. T. Chronopoulos, *AIAA J.* **30**(10), 2256 (1991).
25. H. Van Der Vorst, *SIAM J. Sci. Statist. Comput.* **3,** 350 (1982).
26. P. M. Goorjian, M. E. Meagher, and R. Van Buskirk, *AIAA J.* **23**(4), 492 (1985).
27. G. R. Di Brozolo, and Y. Robert, *Parallel Comput.* **11,** 223 (1989).
28. A. M. Wissink, A. S. Lyrintzis, and A. T. Chronopoulos, AIAA Paper No. 95-0576, 1995 (unpublished).
29. E. M. Murman and J. D. Cole, *AIAA J.* **9,** 114 (1971).
30. S. K. Godunov, *Math. USSR Sb* **47,** 271 (1959).
31. A. Chronopoulos, *J. Comput. Appl. Math.* **40,** 73 (1992).
32. H. A. Van der Vorst, *SIAM J. Sci. Statist. Comput.* **13**(2), 631 (1992).

33. P. Sonneveld, *SIAM J. Sci. Statist. Comput.* **10**(1), 36 (1989).
34. S. Ma and A. T. Chronopoulos, *Int. J. Supercomputing Appl.* **4**(4), 9 (1990).
35. R. W. Freund, *SIAM J. Sci. Comput.* **14**(2), 470 (1993).
36. A. T. Chronopoulos and C. W. Gear, *J. Comput. Appl. Math.* **25,** 315 (1989).
37. A. T. Chronopoulos and C. W. Gear, *Parallel Comput.* **11,** 37 (1989).
38. J. A. Meijerink and H. A. Van der Vorst, *Math. Comput.* **31,** 137 (1977).