AIAA JOURNAL
Vol. 34, No. 11, November 1996

Parallelization of a Three-Dimensional Flow Solver for Euler Rotorcraft Aerodynamics Predictions

Andrew M. Wissink*
University of Minnesota, Minneapolis, Minnesota 55455
Anastasios S. Lyrintzis†
Purdue University, West Lafayette, Indiana 47907
and
Roger C. Strawn‡
NASA Ames Research Center, Moffett Field, California 94035

An approach for parallelizing the three-dimensional Euler/Navier–Stokes rotorcraft computational fluid dynamics flow solver transonic unsteady rotor Navier–Stokes (TURNS) is introduced. Parallelization is performed using a domain decomposition technique that is developed for distributed-memory parallel architectures. Communication between the subdomains on each processor is performed via message passing in the form of message passing interface subroutine calls. The most difficult portion of the TURNS algorithm to implement efficiently in parallel is the implicit time step using the lower-upper symmetric Gauss–Seidel (LU-SGS) algorithm. Two modifications of LU-SGS are proposed to improve the parallel performance. First, a previously introduced Jacobi-like method called data-parallel lower upper relaxation (DP-LUR) is used. Second, a new hybrid method is introduced that combines the Jacobi sweeping approach in DP-LUR for interprocessor communications and the symmetric Gauss–Seidel algorithm in LU-SGS for on-processor computations. The parallelized TURNS code with the modified implicit operator is implemented on two distributed-memory multiprocessor, the IBM SP2 and Thinking Machines CM-5, and used to compute the three-dimensional quasisteady and unsteady flowfield of a helicopter rotor in forward flight. Good parallel speedups with a low percentage of communication are exhibited by the code. The proposed hybrid algorithm requires less CPU time than DP-LUR while maintaining comparable parallel speedups and communication costs. Execution rates found on the IBM SP2 are impressive; on 114 processors of the SP2, the solution time of both quasisteady and unsteady calculations is reduced by a factor of about 12 over a single processor of the Cray C-90.

I. Introduction

ACCURATE numerical simulation of the aerodynamics and aeroacoustics of a helicopter is a complex and challenging problem. The flowfield of modern rotorcraft is characterized by three-dimensional transonic aerodynamic phenomena and complex interaction of the blades with their shed vortices. Accurate prediction of these effects is vital for the control of aerodynamic losses, vibration, and noise. Euler/Navier–Stokes computational fluid dynamics (CFD) methods have been employed in recent years for prediction of these effects.[1–4] Transonic flow is normally encountered by rotors in high-speed forward flight, and by comparison with potential methods, Euler/Navier–Stokes methods more accurately describe shock strength and position as well as viscous-inviscid interaction effects. They also admit vortical solutions without an added wake model.

The use of Euler/Navier–Stokes methods for three-dimensional complex rotorcraft configurations can put heavy demands on existing computer resources. These calculations are typically performed on vector supercomputers of the Cray class and require many hours of CPU time as well as large amounts of memory. Parallel processing offers an alternative to vector processing that is both cheaper and faster. At present, a number of vendors have released multiprocessor machines (e.g., IBM SP2, Intel Paragon, and SGI Power Challenge) that utilize commodity processors the same as those used in high-end workstations. The performance of these machines is scalable with the number of processors, and their price/performance is significantly better than more traditionally used vector processors. Substantial increases in computational power, both in memory and computation speed, are expected in future parallel machines. Some companies are also utilizing parallel processing in the form of networked workstations, which sit idle during off hours, to attain supercomputer performance. An excellent review of the current status of parallel computing in CFD is given by Knight.[5]

Although parallel processing offers considerable potential for increased computational power, parallelizing modern CFD algorithms is not a trivial task. While simpler methods such as explicit schemes can be parallelized rather easily and usually exhibit high performance on parallel machines, they are much less efficient than implicit methods due to poor convergence rates. Implicit schemes that have efficient convergence properties are often difficult to parallelize and tend to perform far below the peak execution rate. This is particularly true on distributed-memory parallel architectures.

In this paper, we investigate a parallel implementation of the well-used Euler/Navier–Stokes rotorcraft CFD code transonic unsteady rotor Navier–Stokes (TURNS).[3,4] The implicit operator used in TURNS for time stepping in both steady and unsteady calculations is the lower upper symmetric Gauss–Seidel (LU-SGS) scheme of Yoon and Jameson.[6] To make the implicit solution more parallelizable, a modified form of this operator is proposed, which is based on the data-parallel lower upper relaxation (DP-LUR) algorithm of Candler et al.[7] and Wright et al.[8] but is designed to be more efficient in the presence of the domain-decomposition implementation. Aside from the modified implicit operator, there are no algorithm changes to the original code, and so the implementation requires a relatively small amount of code rewriting. Communications between processors are performed via message passing interface (MPI) subroutine calls. The distributed solver is implemented on two modern distributed-memory multiprocessors, the Thinking Machines CM-5 and IBM SP2, and its parallel performance is tested

Received Oct. 5, 1995; revision received July 14, 1996; accepted for publication July 15, 1996; also published in *AIAA Journal on Disc*, Volume 2, Number 1.

*NASA Fellow, Department of Engineering and Mechanics, 107 Akerman Hall, 110 Union Street SE; currently Research Scientist, MCAT, Inc., NASA Ames Research Center, Mail Stop 258-1, Moffett Field, CA 94035. Member AIAA.

†Associate Professor, School of Aeronautics and Astronautics, 1282 Grissom Hall. Associate Fellow AIAA.

‡Research Scientist, U.S. Army Aeroflightdynamics Directorate, Aviation and Troop Command, Mail Stop 258-1. Member AIAA.

for three-dimensional quasisteady and unsteady calculations of a helicopter rotor in forward flight. Parallel implementation and performance issues are discussed along with the convergence characteristics of the modified implicit operator. Earlier results of this work were presented in Refs. 9 and 10.

Section II describes the solution algorithm used in the baseline TURNS code. Section III discusses the LU-SGS implicit operator and the variants we propose for more efficient parallelization. Section IV describes the parallel implementation, Sec. V presents results from the IBM SP2 and Thinking Machines CM-5, and some concluding remarks are given in Sec. VI.

II. Baseline Solver

The baseline solver used in this work is the structured-grid single-block Euler/Navier–Stokes rotorcraft CFD code TURNS, developed by Srinivasan et al.[3] and Srinivasan and Baeder[4] in conjunction with the U.S. Army Aeroflightdynamics Directorate at NASA Ames Research Center. The code computes the three-dimensional flowfield of a helicopter rotor (without fuselage) in hover and forward flight conditions. In addition to NASA and the Army, various universities and the four major U.S. helicopter companies use the code. TURNS is a free-wake method, computing the tip vortices and entire vortical wake as part of the overall flowfield solution. The excellent predictive capabilities of TURNS for lifting rotors in hover and forward-flight conditions, in both subsonic and transonic flow regimes, have been validated against wind-tunnel data in other studies.[11,12] The code has also been applied within the framework of overset grids.[13,14] In addition to aerodynamics predictions, TURNS has been used for near-field aeroacoustic calculations by Baeder et al.[15] and for far-field noise prediction by Strawn and Biswas,[16] Strawn et al.,[17] and Lyrintzis et al.,[18] who coupled the near-field solution from TURNS with a linear Kirchoff method.

The governing equations solved by TURNS are the three-dimensional unsteady compressible thin-layer Navier–Stokes equations, applied in conservative form in a generalized body-fitted curvilinear coordinate system

$$\partial_\tau \boldsymbol{Q} + \partial_\xi \boldsymbol{E} + \partial_\eta \boldsymbol{F} + \partial_\zeta \boldsymbol{G} = (\varepsilon/Re)\partial_\zeta \boldsymbol{S} \tag{1}$$

where $\boldsymbol{Q}$ is the vector of conserved quantities; $\boldsymbol{E}$, $\boldsymbol{F}$, and $\boldsymbol{G}$ are the inviscid flux vectors; and $\boldsymbol{S}$ is the viscous flux vector. The generalized coordinates are $\tau = t$, $\xi = \xi(x, y, z, t)$, $\eta = \eta(x, y, z, t)$, and $\zeta = \zeta(x, y, z, t)$, where the coordinate system (x, y, z, t) is attached to the blade. TURNS is run in Euler mode (i.e., $\epsilon = 0$) for all calculations presented in this paper, and so viscous flux determination is not discussed.

The inviscid fluxes are evaluated using Roe's upwind differencing.[19] The use of upwinding obviates the need for user-specified artificial dissipation and improves the shock capturing in transonic flowfields. Third-order accuracy is obtained using the MUSCL approach of Anderson et al.,[20] and flux limiters are applied so that the scheme is total variation diminishing.

The implicit operator used in TURNS for time stepping in both steady and unsteady calculations is the LU-SGS scheme.[6] This operator takes the form

$$LD^{-1}U\,\Delta\boldsymbol{q}^n = -h\boldsymbol{R}(\boldsymbol{q}^n) \tag{2}$$

where D, L, and U are diagonal, lower, and upper tridiagonal matrices, respectively, $\Delta\boldsymbol{q}^n = \boldsymbol{Q}^{n+1} - \boldsymbol{Q}^n$, and h is the time step. A spatially varying time step as proposed in Ref. 21 is used for convergence acceleration of steady-state solutions. The term $\boldsymbol{R}(\boldsymbol{q}^n)$ consists of the spatially differenced inviscid flux vectors at time level n,

$$\boldsymbol{R}(\boldsymbol{q}^n) = \delta_\xi \boldsymbol{E}^n + \delta_\eta \boldsymbol{F}^n + \delta_\zeta \boldsymbol{G}^n \tag{3}$$

Srinivasan and Baeder[4] showed that, for problems similar to those studied in this work, adequate solution accuracy in time-accurate unsteady calculations can be efficiently performed by employing the LU-SGS operator for implicit time stepping. To reduce the implicit factorization error, one applies inner relaxation iterations at each time step, as follows; using the solution at time level n, set the initial condition to $\boldsymbol{Q}^{n+1,0} = \boldsymbol{Q}^n$ and apply LU-SGS to solve the following equation in each inner iteration:

$$[LD^{-1}U]^{n+1,m}\,\Delta\boldsymbol{q}^{n+1,m} = -h\left[\frac{\boldsymbol{Q}^{n+1,m} - \boldsymbol{Q}^n}{h} + \boldsymbol{R}(\boldsymbol{q}^{n+1,m})\right] \tag{4}$$

where $\Delta\boldsymbol{q}^{n+1,m} = \boldsymbol{Q}^{n+1,m+1} - \boldsymbol{Q}^{n+1,m}$. In Eq. (4), n refers to the time level and m to the iteration level. Three inner iterations were used for the unsteady cases in this work. Upon completion of the inner iterations, the solution at the next time level is $\boldsymbol{Q}^{n+1} = \boldsymbol{Q}^{n+1,m_{\max}}$.

Additional algorithm details of TURNS are given in Ref. 3.

III. Implicit Operator

LU-SGS

The LU-SGS method, originally proposed by Yoon and Jameson,[6] is a popular implicit method that has recently been incorporated into a number of well-known CFD codes (e.g., INS3D[22] and OVERFLOW[23]). The primary advantage of the scheme is its very robust stability properties. For three-dimensional problems, three-factor alternating direction implicit schemes can become neutrally stable or even unstable with large time steps. The two-factor LU-SGS scheme is formulated to be stable independent of the size of the time step. In addition, the factorization error introduced by LU-SGS is $\mathcal{O}(\Delta t^2)$ as opposed to a factorization error of $\mathcal{O}(\Delta t^3)$ for ADI methods. Despite the many advantages of LU-SGS, the implicit solution using this operator is the most difficult portion of the TURNS algorithm to implement efficiently in parallel.

Although specifics of the LU-SGS algorithm are discussed elsewhere,[6] a brief discussion of its implementation in TURNS is given here to facilitate description of the parallelization approach. In a standard LU implementation, the diagonal factor D in Eq. (2) contains the 5×5 matrices A, B, and C that are the Jacobians of the flux vectors with respect to the conserved quantities (e.g., $A = \partial \boldsymbol{E}/\partial \boldsymbol{Q}$). These matrices are split into their + and − eigenvalue parts, which are backward and forward differenced, respectively,

$$\begin{aligned}\delta_\xi A &= \nabla_\xi A^+ + \Delta_\xi A^- \\ &= A_j^+ - A_{j-1}^+ + A_{j+1}^- - A_j^-\end{aligned} \tag{5}$$

To compute D^{-1}, the 5×5 flux Jacobians must be inverted at each grid point, leading to costly computation times. Yoon and Jameson propose an efficient spectral approximation for the flux Jacobians

$$A^\pm = \tfrac{1}{2}(A \pm \rho_A I) \pm \varepsilon \rho_A I \tag{6}$$

where ρ_A is the spectral radius of A (in the ξ direction) and ε is set to some small value (e.g., $\varepsilon = 0.001$). With the spectral approximation, the sum of the $A^\pm$, $B^\pm$, and $C^\pm$ terms in D reduce to the scalar quantities ρ_A, ρ_B, and ρ_C, and so inversion of this factor is straightforward. Also, use of a spectral approximation ensures the largest eigenvalues will be located on the diagonal of the flux Jacobians, making L and U diagonally dominant regardless of the size of the time step. With the spectral approximation applied, the D, L, and U factors are

$$\begin{aligned}D &= I + h(\rho_A + \rho_B + \rho_C)_{j,k,l} \\ L &= D - h\left(A_{j-1}^+ + B_{k-1}^+ + C_{l-1}^+\right) \\ U &= D + h\left(A_{j+1}^- + B_{k+1}^- + C_{l+1}^-\right)\end{aligned} \tag{7}$$

and solution of Eq. (2) is obtained by applying the following two-step procedure:

$$L\Delta\boldsymbol{q}^* = -h\boldsymbol{R}(\boldsymbol{q}^n), \qquad U\Delta\boldsymbol{q}^n = D\Delta\boldsymbol{q}^* \tag{8}$$

which can be efficiently performed using a symmetric Gauss–Seidel algorithm, as follows.

Algorithm 1: LU-SGS. Do $j, k, l = 1, \ldots, J_{\max}, K_{\max}, L_{\max}$,

$$\begin{aligned}\Delta\boldsymbol{q}_{j,k,l}^* = D^{-1}h\big[&-\boldsymbol{R}(\boldsymbol{q}^n) + A_{j-1}^+\Delta\boldsymbol{q}_{j-1}^* \\ &+ B_{k-1}^+\Delta\boldsymbol{q}_{k-1}^* + C_{l-1}^+\Delta\boldsymbol{q}_{l-1}^*\big]\end{aligned}$$

End Do.
Do $j, k, l = J_{\max}, K_{\max}, L_{\max}, \ldots, 1$,

$$\Delta q^n_{j,k,l} = \Delta q^*_{j,k,l} - D^{-1}h\left(A^-_{j+1}\Delta q^n_{j+1} + B^-_{k+1}\Delta q^n_{k+1} + C^-_{l+1}\Delta q^n_{l+1}\right)$$

End Do.

The approach currently used to vectorize the LU-SGS algorithm in TURNS is the hyperplane approach, as described by Barszcz et al.[24] Hyperplanes are formed in the solution domain in which the grid point coordinates $j + k + l = \text{const}$ and vectorization is performed across these planes. Although the hyperplane approach leads to good vector execution rates, parallelizing with this approach is difficult for two reasons; 1) the sizes of the hyperplanes vary throughout the grid, leading to load balancing problems, and 2) there is a recursion between the planes, leading to a large amount of communication. Barszcz et al.[24] also describe an alternative approach for parallelizing LU-SGS in a distributed environment by restructuring the data layout using a skew-hyperplane approach. Although the skew-hyperplane approach yields relatively good parallelism, the data layout is complex, the restructuring of data on the left-hand side causes the right-hand side layout to be skewed, and additional communication is required. Overall, the LU-SGS algorithm in its original form is not very conducive to efficient parallel implementation.

Domain decomposition implementations of LU-SGS have also been investigated as an alternative for parallelizing the operator. In this approach, the overall flowfield domain is divided into subdomains and LU-SGS is applied simultaneously to each individual subdomain. Wong et al.[25] utilized domain decomposition LU-SGS for two-dimensional structured-grid reacting flow problems, and Taflin et al.[26] performed similar tests with unstructured grids. Both found that the convergence rate of LU-SGS remains good with a low number of subdomains but deteriorates significantly as the number of subdomains becomes large.

DP-LUR

Candler et al.[7] and Wright et al.[8] have introduced a modified form of LU-SGS called data-parallel lower upper relaxation (DP-LUR) that has been demonstrated to be very efficient in a data-parallel environment for solving reacting flow problems. Essentially, the modification involves transferring the nondiagonal terms to the right-hand side and replacing the symmetric Gauss–Seidel sweeps with Jacobi sweeps. Using the same notation as the LU-SGS algorithm, the DP-LUR algorithm can be written as follows.

Algorithm 2: DP-LUR.

$$\Delta q^{(0)}_{j,k,l} = -D^{-1} \cdot h\boldsymbol{R}(q^n)$$

For $i = 1, \ldots, i_{\text{sweep}}$ Do.
Do $j, k, l = 1, \ldots, J_{\max}, K_{\max}, L_{\max}$ Do,

$$\Delta q^{(i)}_{j,k,l} = D^{-1} \cdot h \times \begin{bmatrix} -\boldsymbol{R}(q)^n + A^+_{j-1}\Delta q^{(i-1)}_{j-1} + B^+_{k-1}\Delta q^{(i-1)}_{k-1} + C^+_{l-1}\Delta q^{(i-1)}_{l-1} \\ -A^-_{j+1}\Delta q^{(i-1)}_{j+1} - B^-_{k+1}\Delta q^{(i-1)}_{k+1} - C^-_{l+1}\Delta q^{(i-1)}_{l+1} \end{bmatrix}$$

End Do.
End for

$$\Delta q^n_{j,k,l} = \Delta q^{(i_{\text{sweep}})}$$

In the DP-LUR algorithm, i_{sweep} is the number of sweeps of the domain (usually 3–6). A certain minimum value is required to achieve convergence, and the robustness can be enhanced by using higher values. The algorithm is very amenable to parallel processing because the Jacobi sweeping strategy uses only nearest neighbor data and therefore allows computations to be carried out simultaneously on multiple processors with each processor holding local data. Nearest-neighbor data that lie on the processor borders are communicated in each global data update (i.e., after each domain sweep), and because only a few global sweeps are required, the total number of communication steps is small. The algorithm can be implemented on parallel processors such that computations are completely load balanced with communications occurring only at the borders of each partition.

Although DP-LUR is much more amenable to parallel processing than LU-SGS, it also requires more computational work. It is well known that in solving linear systems the convergence rate of a Jacobi algorithm is slower than symmetric Gauss–Seidel and will therefore require more iterations for convergence. The inner domain sweeps employed by DP-LUR at each time step are essentially multiple iterations that serve the purpose of approximating the solution to the linear system to a sufficient level of accuracy such that the nonlinear iterations will converge. Whereas LU-SGS performs two domain sweeps at each time step (i.e., one forward and one backward), DP-LUR was found to require five to six inner sweeps to maintain the same nonlinear convergence rate as LU-SGS for the problems investigated in this work (the details of which will be discussed later). The increased number sweeps in DP-LUR at each time step leads to an overall increase in computational work in the implicit solution by a factor of 2.5–3. Although DP-LUR is much more parallelizable than LU-SGS, the question is whether the computational penalty of DP-LUR is the best that we can do.

Hybrid

The DP-LUR algorithm was developed primarily for data-parallel computations. Its convergence is independent of the number of processors used because the on-processor computations and interprocessor communications are performed using the same Jacobi sweeping approach. Although data-parallel implementations generally require the on-processor computations to be performed in the same way as the interprocessor computations, message passing gives the user more control over the communication patterns and allows for the possibility of performing them differently. With this in mind, we propose an algorithm that couples the benefits of both previously introduced algorithms, namely the efficient on-processor computations of LU-SGS and the efficient interprocessor communications of DP-LUR.

The proposed algorithm is referred to as the hybrid algorithm, because it combines methodology from both LU-SGS and DP-LUR. In the hybrid algorithm, the SGS sweeping approach of the original LU-SGS algorithm is applied separately to each processor subdomain and executed simultaneously by all subdomains. Then interprocessor communications are performed using the same Jacobi sweeping strategy as is used in DP-LUR. The use of multiple inner domain sweeps in DP-LUR is retained in the hybrid algorithm to enhance the robustness.

Algorithm 3: Hybrid.

$$\Delta q^{(0)}_{j,k,l} = -D^{-1} \cdot h\boldsymbol{R}(q^n)$$

For $i = 1, \ldots, i_{\text{sweep}}$ Do.
Communicate border $\Delta q^{(i-1)}_{j,k,l}$ data.

$$\Delta q^{*(i)}_{j,k,l} = \Delta q^{(i-1)}_{j,k,l}$$

Perform Algorithm 1 locally to compute $\Delta q^{(i)}_{j,k,l}$.
End for

$$\Delta q^n_{j,k,l} = \Delta q^{(i_{\text{sweep}})}_{j,k,l}$$

On a single domain (i.e., single processor) the hybrid algorithm performs only the SGS sweeps and is therefore identical to the original LU-SGS algorithm. On many processors (i.e., in the limit as the number of subdomains approaches the number of grid points), the hybrid algorithm performs only interprocessor communications using the Jacobi algorithm and is identical to DP-LUR. The hybrid algorithm can thus be considered a mixture of the two algorithm types (i.e., SGS and Jacobi) with SGS having a more dominant influence with a small number of processors and Jacobi being more dominant with a large number of processors. Since SGS has a faster convergence rate than Jacobi, the algorithm should have the fastest convergence and be most robust with fewer numbers of processors.

Parallel implementation of the hybrid algorithm in a domain decomposition format is straightforward and can be performed in the same way as DP-LUR. Border data are communicated to nearest

neighbors at the beginning of each sweep, and each processor simultaneously performs the standard LU-SGS algorithm on its subdomain. Like DP-LUR, the hybrid algorithm maintains load balanced parallelism with only nearest-neighbor communications.

IV. Parallel Implementation

Two important requirements for effectively utilizing available parallel computational power is to develop algorithms that can keep pace with rapidly changing hardware designs and to have methods that can be implemented on a variety of parallel environments (e.g., parallel supercomputers to workstation clusters). To better facilitate these requirements and to avoid a costly Fortran 77 to Fortran 90 rewriting effort, a multiple instruction multiple data (requiring message passing) approach is chosen over a data-parallel or single instruction multiple data approach for parallel implementation of TURNS. The message passing calls used in the code are taken from the MPI standard library, which is supported by most vendors and is becoming a standard in the parallel processing community. To ensure portability, a set of generic message passing subroutines is used. With this protocol, the specific message passing commands can be altered in one line of the code rather than throughout, making conversion to different message passing languages, such as parallel virtual machine, a relatively short procedure.

A domain decomposition strategy that preserves the original construct of the code is used to lay out the domain on an array of processors. The flowfield domain is divided in the wraparound and spanwise directions into subdomains and layed out onto a two-dimensional array of processors, as demonstrated in Fig. 1. Each processor executes a version of the code simultaneously for the portion of the flowfield that it holds. The processors are assigned coordinates that are used to determine the global values of the data each holds. Border data are communicated between processors, and a single layer of ghost cells is used to store this communicated data.

Although the amount of data communicated can be minimized by using square subdomains in the decomposition of the flowfield, this requires dividing the domain in the normal direction. The normal direction is left intact in our implementation to avoid a load imbalance from occurring during application of the boundary conditions in the plane of the rotor. In the region outside of the rotor in the rotor plane, the C-H grid collapses to a plane and an averaging of

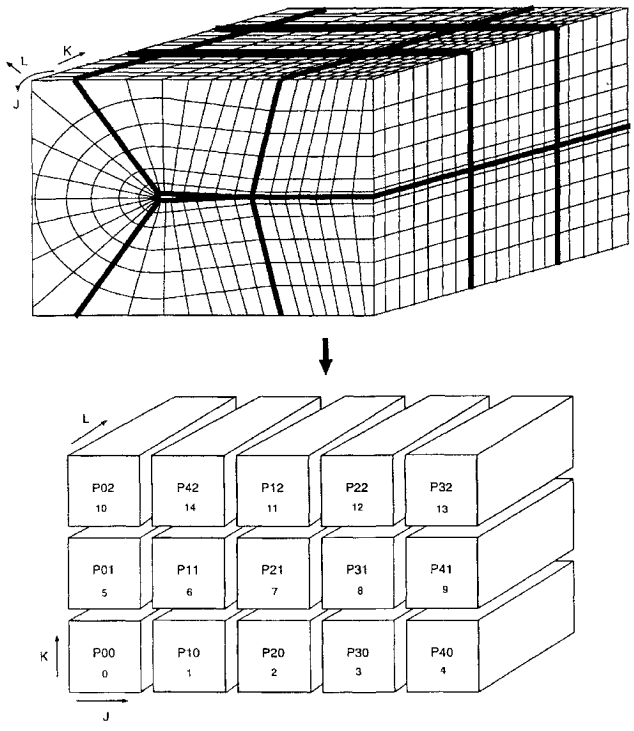


Fig. 1 Partitioning the three-dimensional domain on a two-dimensional array of processors.

data is performed between nodes lying on either side of the plane. This averaging requires communication of data to processors on the other side of the plane. If the normal direction were to be divided, processors not holding data on the C plane would sit idle while those holding data on the plane would perform the communication, inducing a load imbalance. In the present implementation, all processors participate in this communication and the load imbalance is avoided. The reduction in parallel efficiency caused by use of nonsquare subdomains is expected to be less costly than if the load imbalance were allowed to occur.

There are essentially three main portions of the solution algorithm in TURNS. The first is the spatial discretization using the Roe upwinded algorithm to form the right-hand side. The communication required for this step is straightforward. After the flux vectors are determined using the MUSCL routine, they are communicated and stored in the ghost layer and first-order Roe differencing is applied. This communication step could be avoided by using a ghost layer of two cells, but the present approach is easier to implement into the existing code. The second portion of the solution algorithm is application of the boundary conditions. This can be done locally by each processor, with the exception of the averaging of data across the C plane, discussed earlier. The third part of the algorithm is the implicit solution, for which the DP-LUR (Algorithm 2) and hybrid (Algorithm 3) schemes are utilized.

Note that incorporating the hybrid algorithm into the baseline code requires fewer overall code modifications than DP-LUR. Since the hybrid method executes the baseline code's LU-SGS algorithm on each subdomain, the only modification required in the implicit algorithm of the baseline code is the addition of message passing calls to incorporate the interprocessor communications approach. Addition of DP-LUR requires the LU-SGS algorithm in the baseline code to be modified to incorporate the Jacobi sweeping approach, along with the addition of message passing subroutine calls.

V. Results

The parallel implementation of TURNS is tested on two distributed-memory multiprocessors: the 160-node IBM SP2 at NASA Ames Research Center in Mountain View, California, and the 896-node Thinking Machines CM-5 at the Army High Performance Computing Research Center in Minneapolis, Minnesota. The code is used to compute the quasisteady (i.e., blade-fixed) and unsteady forward-flight aerodynamic flowfield of a rotating helicopter rotor without fuselage. Viscous effects have not yet been included in the parallel implementation, and so the code is run in Euler mode for a nonlifting test case.

The test case used in a symmetric operational load survey (OLS) helicopter blade rotating with a tip Mach number of $M_{tip} = 0.665$ and moving forward with an advance ratio of $\mu = 0.258$. The OLS blade has a sectional airfoil thickness to chord ratio of 9.71% and is a one-seventh scale model of the main rotor for the Army's AH-1 helicopter. This particular test case has been used in other studies.[16–18] A $135 \times 50 \times 35$ C-H type grid is used that extends out to 2 rotor radii from the hub in the plane of the rotor and 1.5 rotor radii above and below the plane. The upper half of the grid is shown in Fig. 2.

The parallel performance of the code is investigated with various numbers of processor subdomains on both machines. On the IBM SP2, five different partitions are tested; 1, 4, 8, 19, 57, and 114 processors. The subdomains for these cases are formed as follows; for the 4–8 processor cases, the wraparound direction in the grid is left intact while the spanwise direction is divided into 4 and 8 subdomains, respectively. For the 19, 57, and 114 processor cases, the wraparound direction is divided into 19 subdomains, and the spanwise direction is divided into 3 and 6 subdomains, respectively. Unlike the SP2, which allows specific processor partitions to be chosen, the Thinking Machines CM-5 is available only in processor partitions of 64, 256, and 512 processors. Because of the dimensions of the grid used, it is impossible to divide our problem into subdomains that correspond exactly to these processor partitions, and so the subdomains are formed to utilize most of the processors in the partition. The extra processors simply sit idle. The 57 subdomain case, described earlier, is executed on the 64 node partition. A 228 subdomain case, formed by dividing the wraparound direction into

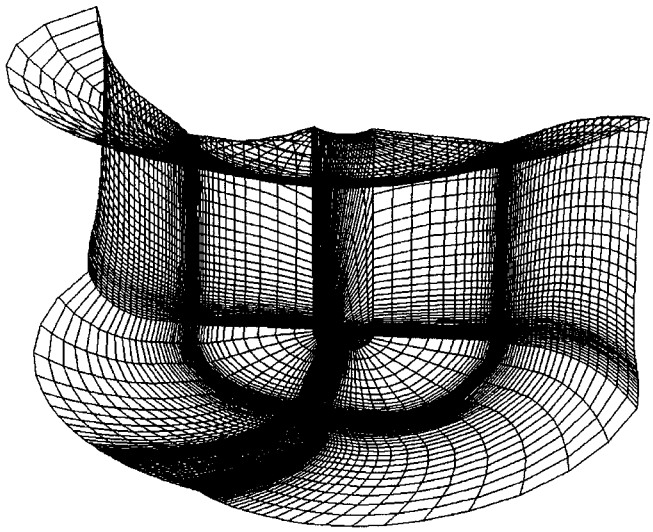
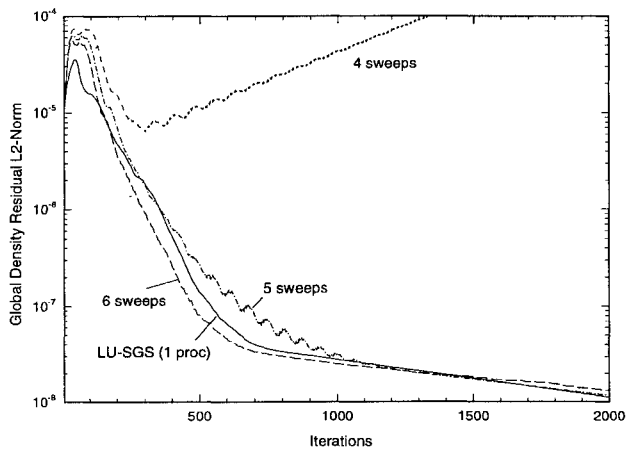
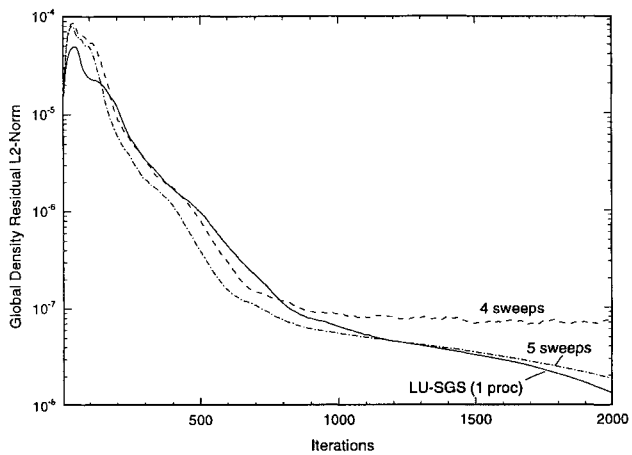


Fig. 2 Upper half of 135 × 50 × 35 C-H type grid.



a) ψ = 0 deg



b) ψ = 90 deg

Fig. 3 Convergence with DP-LUR operator.

19 subdomains and the spanwise direction into 12, is executed on the 256 node partition. A 456 subdomain case, with 19 subdomains in the wraparound and 24 in the spanwise directions, is executed on the 512 node partition. Because of the odd dimensions of the grid in the wraparound direction, we were unable to use a more variable distribution of processors in this direction.

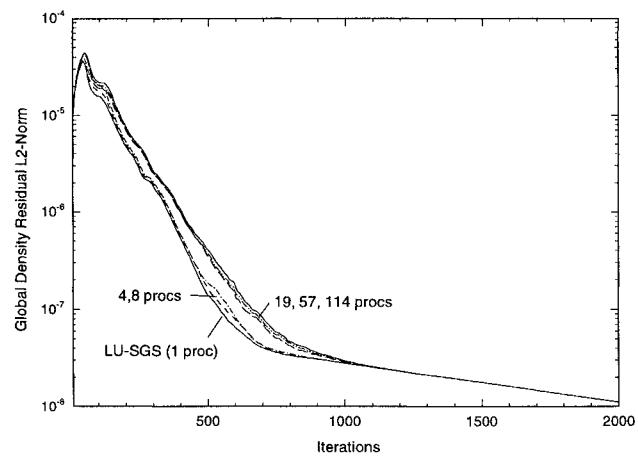
Since the results of the TURNS code have been verified against experimental data in other works, the main objective of these tests is to investigate the parallel performance of the code and the numerical performance of the two implicit operator modifications (i.e., DP-LUR and hybrid). Time-independent quasisteady blade-fixed cases, which are started from ambient conditions, provide a convenient framework for comparison of the numerical performance of the implicit operators, and so results are first presented for quasisteady calculations. Then, results from a more computationally intensive time-accurate unsteady application of TURNS are presented. Both quasisteady and unsteady solutions were verified against the original version of the code run on the Cray C-90 to assure that they are identical. Since the only algorithm modifications occur in the implicit operator, the solutions from both machines are exactly the same if converged to the same level of accuracy.

Quasisteady

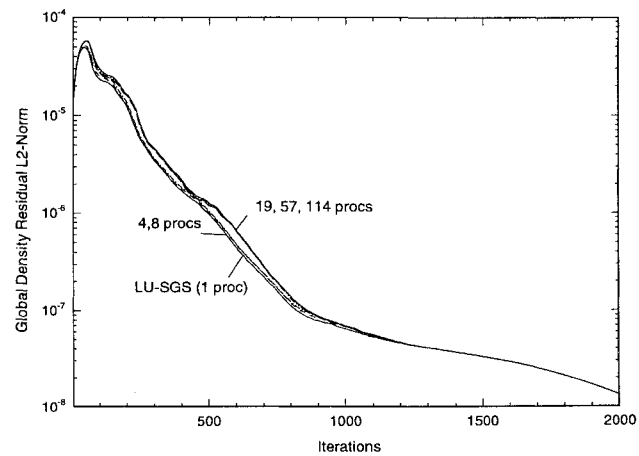
The parallelized code is used to compute the quasisteady flowfield with blade azimuth angles of $\psi = 0$ and 90 deg. The $\psi = 0$ deg case is predominantly subsonic because the tip moves in a direction perpendicular to the freestream. The $\psi = 90$ deg case is more transonic because the tip is moving directly into the freestream and experiences an advancing tip Mach number of 0.835.

The convergence of the global L2 density norm (i.e., residual of continuity equation) for the $\psi = 0$ and 90 deg cases using the DP-LUR operator are shown in Figs. 3a and 3b. Note that the norm of the global residual of all equations, as well as the maximum residual, was also plotted and showed very similar results to the global L2 density norm. A minimum of five inner sweeps (i.e., $i_{\text{sweep}} = 5$) is required for convergence in both cases. Figure 3b appears to indicate that DP-LUR is well suited for transonic flow solutions because it demonstrates more robust convergence for the same number of inner sweeps for the transonic $\psi = 90$ deg case. Because the Jacobi algorithm in DP-LUR is used for both on-processor computations and interprocessor communications, the convergence with DP-LUR will be the same regardless of the number of processors used.

The convergence of the $\psi = 0$ and 90 deg cases using the hybrid operator with $i_{\text{sweep}} = 1$ are shown in Figs. 4a and 4b. Because the



a) ψ = 0 deg



b) ψ = 90 deg

Fig. 4 Convergence with hybrid operator (i_{sweep} = 1).

hybrid operator uses a different algorithm for on-processor computations (i.e., SGS) than interprocessor communications (i.e., Jacobi), its convergence varies with different numbers of processors. With $i_{sweep} = 1$, the total amount of computational work in each iteration is the same as with original LU-SGS, and it is apparent from Figs. 4a and 4b that although the convergence worsens with increasing numbers of processors, the global convergence with only a single sweep is comparable to the original LU-SGS method. Also, as was the case with DP-LUR, the hybrid operator appears to show no adverse effects in the presence of transonic flow with the $\psi = 90$ deg case.

Table 1 compares the time per iteration, percentage communication, and parallel speedup for the hybrid method with $i_{sweep} = 1$ and DP-LUR with $i_{sweep} = 5$. These two cases are chosen because they represent the fewest number of inner sweeps required in both methods to achieve convergence and will consequently have the fastest per iteration time. The percentage of communication is determined by timing the message passing calls in the code and comparing with the overall solution time. Parallel speedup is determined by comparing the time per iteration to the single processor case. The parallel speedups for these cases are plotted in Fig. 5. The parallel speedup and percentage of communication for the two operators is nearly identical. This is expected because the communication routines performed by both operators are identical. DP-LUR incurs a slightly higher percentage of communication and lower parallel speedup, which is probably due to the higher number of inner sweeps, but the difference is small. The major difference between the two operators is in the CPU time per iteration. The hybrid operator requires about 45% less computation time per iteration than DP-LUR. This is because fewer inner domain sweeps are required at each iteration, thereby incurring less computational work.

A comparison of the total solution time for the $\psi = 0$ deg case is shown in Fig. 6. This plot shows the convergence of the L2 density residual vs wall clock time for the hybrid operator with one and two inner sweeps and DP-LUR with five and six inner sweeps, executed on 114 processors of the SP2. The convergence of the baseline method with the original LU-SGS operator on a single processor of the Cray C-90 is also shown for comparison. There is very little difference in wall clock time with one and two sweeps of the hybrid operator for the first three order of magnitude residual reduction. After this, a single sweep is more efficient; DP-LUR is most efficient with six sweeps for the first three order of magnitude reduction, but five sweeps is more efficient thereafter. It is apparent from this plot that the hybrid operator requires less overall CPU time than DP-LUR. The improvement in execution rate as a result of using parallel processing is also clear from this figure. On 114 processors of the SP2, the code with the hybrid operator is about 12 times faster than the baseline method run on a single processor of the Cray C-90. With DP-LUR, it is about seven times faster.

Table 1 Quasisteady timings on IBM SP2

	DP-LUR, $i_{sweep} = 5$			Hybrid, $i_{sweep} = 1$		
Number of procs.	Time/ iteration, s	Percent comm.	Parallel speedup	Time/ iteration, s	Percent comm.	Parallel speedup
1	23.8	—	1.0	13.08	—	1.0
4	6.95	2.8	3.4	3.91	2.3	3.3
8	3.34	5.8	7.1	2.09	4.5	6.3
19	1.34	8.1	17.8	0.706	8.1	18.5
57	0.464	13.5	51.3	0.245	12.5	53.4
114	0.260	20.6	91.5	0.139	18.2	94.1

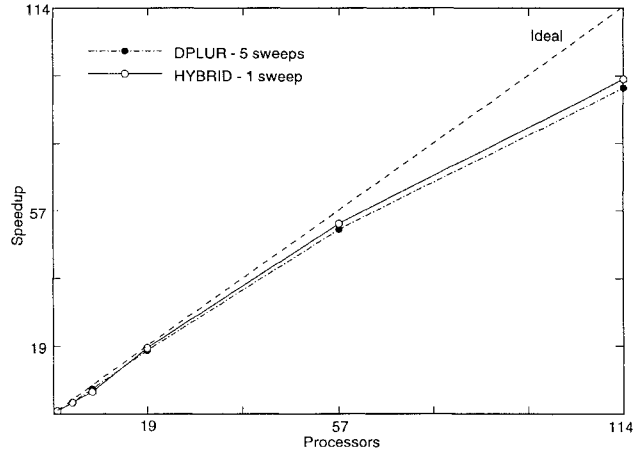


Fig. 5 Parallel speedup of TURNS on SP2.

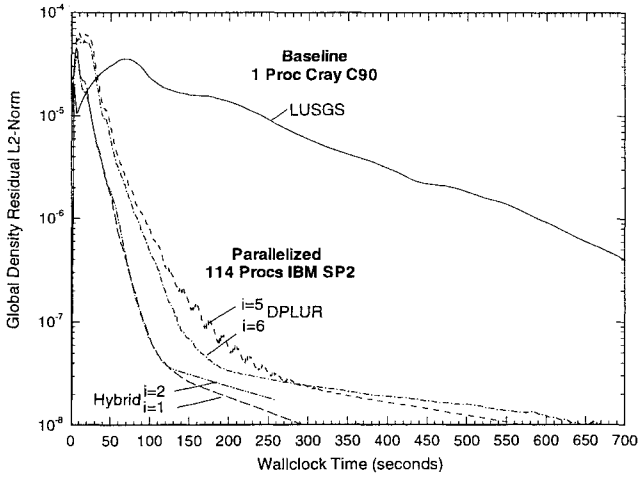


Fig. 6 Comparison of solution times with baseline and parallelized code.

The performance of the hybrid operator is next investigated using a larger number of processors available on the Thinking Machines CM-5. Figure 7a shows the convergence of TURNS for the $\psi = 0$ deg quasisteady case on 57, 228, and 456 processors using $i_{sweep} = 1$ in the hybrid operator. Figure 7b shows the convergence for the same case with $i_{sweep} = 2$. With two sweeps, all processor partitions converge nearly identically with single processor LU-SGS, indicating that two sweeps in the hybrid method can be effectively used to the eliminate the reduction in covergence caused by the domain breakup on this larger number of processors.

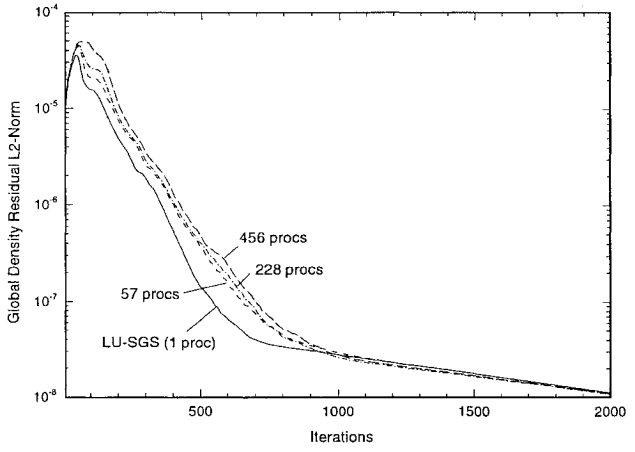
Timings of TURNS with the hybrid operator with one and two inner sweeps on 57, 228, and 456 processors of the CM-5 are presented in Table 2. With two sweeps, the time per iteration increases by about 19%, but there is little difference in parallel speedup and percentage of communication. One difference between the SP2 and CM-5 results that should be pointed out is that the execution times on the CM-5 are significantly slower than what was attained on the SP2. The two machines differ in the processor type used in each node. The SP2 uses RS6000 processors that have a single processor peak execution rate of 260 megaflops, but most applications usually achieve only 15–20% of this. TURNS was found to have an execution rate of about 41 megaflops per processor on the SP2. Each node of the CM-5 contains a SPARC processor that has a peak execution rate of 5 megaflops, and in addition to the processor, it also contains vector units (VU) that accelerate the execution rate to 128 megaflops. Unfortunately, utilization of the VU requires rewriting the code in CMFortran, a high-performance Fortran-type language, which requires significant rewriting effort and is beyond the scope of this work. Consequently, the results presented in Table 2 are determined without utilizing the VU and are significantly slower than the full performance of the machine. Despite the poor on-processor performance, the parallel speedups attained on the CM-5 are good, and the results demonstrate that the hybrid operator is effective at maintaining the convergence rate on a large numbers of processors.

Unsteady

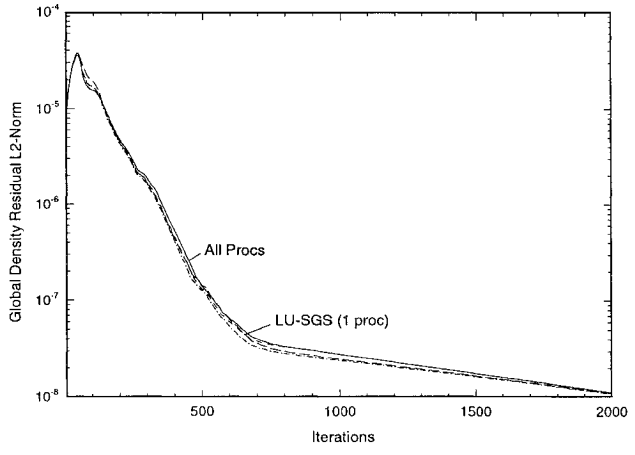
The performance of the hybrid operator is investigated for solution of an unsteady time-accurate case that uses implicit time stepping. Using the quasisteady solution at $\psi = 0$ deg as a starting solution, an unsteady flow calculation is performed for one complete revolution of the OLS blade. The same solution mesh is used to start the unsteady calculation, but the mesh is set to rotate with the blade. A time step corresponding to 0.25 deg azimuth/time step

Table 2 Quasisteady timings on Thinking Machines CM5

Number of procs.	Hybrid, $i_{sweep} = 1$ Time/ iteration, s	Hybrid, $i_{sweep} = 1$ Percent comm.	Hybrid, $i_{sweep} = 1$ Parallel speedup	Hybrid, $i_{sweep} = 2$ Time/ iteration, s	Hybrid, $i_{sweep} = 2$ Percent comm.	Hybrid, $i_{sweep} = 2$ Parallel speedup
57	10.72	8.7	1.0	12.75	9.1	1.0
228	3.34	15.4	3.2/4	4.10	13.0	3.1/4
456	1.97	17.6	5.4/8	2.40	18.3	5.3/8



a) $i_{sweep} = 1$

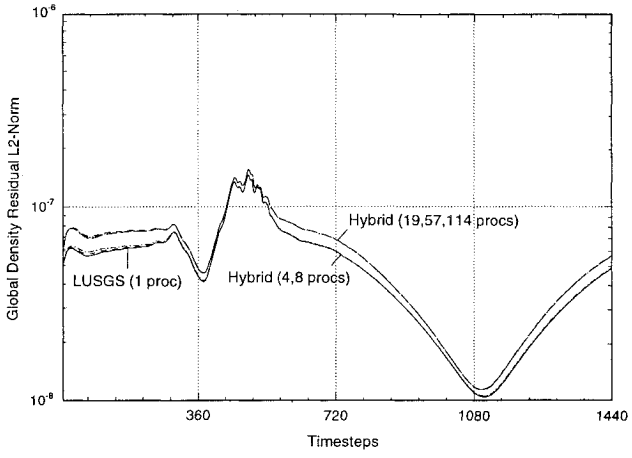


b) $i_{sweep} = 2$

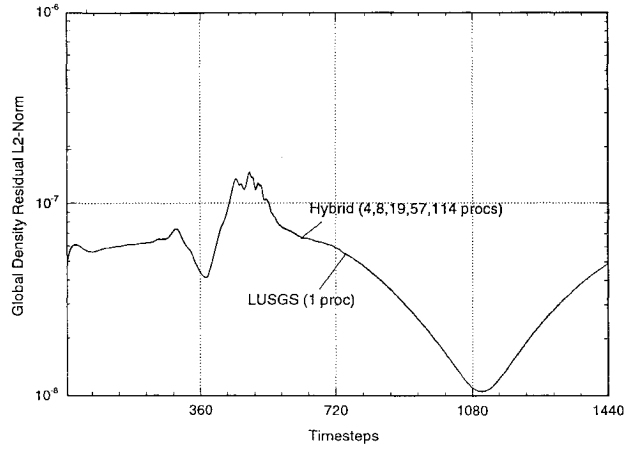
Fig. 7 Convergence with hybrid operator on CM5.

Table 3 Unsteady timings on IBM SP2 (1 rev., 1440 time steps)

Number of procs.	Hybrid, $i_{sweep} = 1$ Total time, min	Hybrid, $i_{sweep} = 1$ Percent comm.	Hybrid, $i_{sweep} = 1$ Parallel speedup	Hybrid, $i_{sweep} = 2$ Total time, min	Hybrid, $i_{sweep} = 2$ Percent comm.	Hybrid, $i_{sweep} = 2$ Parallel speedup
1	986.3	—	1.0	—	—	—
4	286.0	2.2	3.4	329.0	2.6	3.0
8	152.4	4.3	6.5	175.2	4.9	5.6
19	51.6	7.8	19.1	63.7	7.9	15.5
57	18.4	13.7	53.6	22.6	14.0	43.6
114	10.2	18.0	96.7	12.3	19.3	80.2



a) $i_{sweep} = 1$



b) $i_{sweep} = 2$

Fig. 8 Unsteady residuals using hybrid operator.

is used, and so a complete revolution is performed in 1440 time steps. Three relaxation iterations (i.e., $m_{max} = 3$) are performed in each time step to reduce the factorization error introduced by the implicit operator. Table 3 shows timing statistics using the hybrid method with one and two inner sweeps. It should be pointed out that the parallel speedup of the hybrid operator with two sweeps appears artificially low in this table. Executing the hybrid operator with two sweeps on a single processor introduces unnecessary computational work, because the second sweep simply duplicates the first. Thus, the parallel speedups on multiprocessor cases with two sweeps are determined by comparing with the time of the single processor case with one sweep, and the added computational work with two sweeps is reflected artificially in the parallel speedup.

Figures 8a and 8b show plots of the norm of the global density residuals for the unsteady case with one and two sweeps, respectively. Although the CPU time is optimal with one inner sweep, a comparison of these figures shows that use of two sweeps effectively causes the convergence to be nearly identical to single processor LU-SGS for all processor partitions tested.

The benefit of parallel computation is realized in this relatively computationally demanding unsteady calculation. On a single processor of the Cray C-90, this calculation required about 2 h of CPU time running at an execution rate of 320 megaflops. On 114 processors of the SP2, the calculation is performed in only about 10 min, which is a reduction in solution time by a factor of 12.

VI. Concluding Remarks

A parallelization approach is introduced for the Euler/Navier–Stokes rotorcraft CFD code TURNS. The parallel implementation utilizes a domain decomposition strategy designed for efficient execution on distributed-memory parallel architectures. An algorithm modification of the implicit LU-SGS operator is proposed to improve parallelization in the implicit solution. Performance of the parallelized solver is demonstrated on two multiprocessors, the IBM SP2 and Thinking Machines CM-5, for Euler calculations of three-dimensional quasisteady and unsteady aerodynamic flowfields of a helicopter blade in forward flight.

With the hybrid and DP-LUR modifications of the implicit LU-SGS operator, the code exhibits good parallel performance for both quasisteady and unsteady calculations. The parallel speedup and percentage of communication is nearly identical for both hybrid and DP-LUR, but the total solution time with the hybrid operator

is about 45% faster because fewer inner sweeps are required at each iteration. The hybrid operator is most efficient on small to moderately sized processor partitions, requiring no additional computational work than LU-SGS while maintaining good parallelism. On larger numbers of processors (i.e., ≥ 19), the hybrid operator requires an additional domain sweep to match the convergence rate of LU-SGS, requiring approximately 20% more computational work. However, good parallel performance is demonstrated with up to 456 processors, and it is expected that the improved parallel performance outweighs the cost of the added computational work. Impressive reductions in solution time are achieved on the IBM SP2 multiprocessor. For both quasisteady and unsteady calculations, the parallelized code with the hybrid operator on 114 processors of the SP2 yielded a 12-fold reduction in solution time over the vectorized baseline code run on a single processor of the Cray C-90.

Although the code studied in this work is primarily used for rotorcraft applications, the parallelization approach is not unique to this application and could readily be applied to other codes that use the LU-SGS implicit operator. It is anticipated that the hybrid algorithm modification of LU-SGS presented here can be extended in a straightforward way to include viscous effects. Since Navier–Stokes calculations tend to be much more computationally demanding than the Euler calculations presented here, it is expected that the benefits of parallel processing should be most realized for these applications.

Acknowledgments

The first author was supported by a NASA Graduate Student Fellowship. This work was supported by allocation grants from the Minnesota Supercomputer Institute and computer time on the Cray C-90 was provided by a grant from the Pittsburgh Supercomputing Center. Computer time on the IBM SP2 was provided by a grant from the Computational Aerosciences branch of NASA Ames Research Center. The authors wish to acknowledge the support of G. R. Srinivasan of JAI Associates for his assistance with the TURNS code.

References

[1]Srinivasan, G. R., and Sankar, L. N., "Status of Euler and Navier–Stokes CFD Methods for Helicopter Applications," *Proceedings of the 2nd AHS International Aeromechanics Specialists' Conference* (Bridgeport, CT), Vol. 2, American Helicopter Society, Alexandria, VA, 1995, pp. 6-1–6-19.

[2]Wake, B. E., and Sankar, L. N., "Solution of Navier–Stokes Equations for the Flow over a Rotor Blade," *Journal of the American Helicopter Society*, Vol. 34, April 1989, pp. 13–23.

[3]Srinivasan, G. R., Baeder, J. D., Obayashi, S., and McCroskey, W. J., "Flowfield of a Lifting Rotor in Hover: A Navier–Stokes Simulation," *AIAA Journal*, Vol. 30, No. 10, 1992, pp. 2371–2378.

[4]Srinivasan, G. R., and Baeder, J. D., "TURNS: A Free-Wake Euler/Navier–Stokes Numerical Method for Helicopter Rotors," *AIAA Journal*, Vol. 31, No. 5, 1993, pp. 959–962.

[5]Knight, D. D., "Parallel Computing in Computational Fluid Dynamics," *77th Fluid Dynamics Panel Symposium on Progress and Challenges in CFD Methods and Algorithms* (Seville, Spain), AGARD Conf. Proceedings 578, Paper 3, AGARD, Neuilly-Sur-Seine, France, 1996, pp. 5.1–5.7.

[6]Yoon, S., and Jameson, A., "A Lower-Upper Symmetric-Gauss Seidel Method for the Euler and Navier–Stokes Equations," *AIAA Journal*, Vol. 26, No. 9, 1988, pp. 1025, 1026.

[7]Candler, G. V., Wright, M. J., and McDonald, J. D., "A Data Parallel Lower-Upper Relaxation Method for Reacting Flows," *AIAA Journal*, Vol. 32, No. 12, 1994, pp. 2380–2386.

[8]Wright, M. J., Candler, G. V., and Prampolini, M., "Data-Parallel Lower-Upper Relaxation Method for the Navier–Stokes Equations," *AIAA Journal*, Vol. 34, No. 7, 1996, pp. 1371–1377.

[9]Wissink, A. W., Lyrintzis, A. S., Strawn, R. C., Oliker, L., and Biswas, R., "Efficient Helicopter Aerodynamic and Aeroacoustic Predictions on Parallel Computers," AIAA Paper 96-0153, Jan. 1996.

[10]Wissink, A. W., Lyrintzis, A. S., and Strawn, R. C., "On Improving Parallelism in the Transonic Unsteady Rotor Navier Stokes (TURNS) Code," *77th Fluid Dynamics Panel Symposium on Progress and Challenges in CFD Methods and Algorithms* (Seville, Spain), AGARD Conf. Proceedings 578, Paper 6, AGARD, Neuilly-Sur-Seine, France, 1996, pp. 6.1–6.11.

[11]Srinivasan, G. R., Raghavan, V., Duque, E. P. N., and McCroskey, W. J., "Flowfield of a Lifting Rotor in Hover by a Navier–Stokes Method," *Journal of the American Helicopter Society*, Vol. 38, No. 3, 1993, pp. 3–13.

[12]Srinivasan, G. R., "A Free-Wake Euler and Navier–Stokes CFD Method and Its Application to Helicopter Rotors Including Dynamic Stall," JAI Associates, Inc., TR 93-01, Mountain View, CA, Nov. 1993.

[13]Srinivasan, G. R., and Ahmad, J. U., "Navier Stokes Simulation of Rotor-Body Flowfields in Hover Using Overset Grids," *Proceedings of the 19th European Rotorcraft Forum* (Cernobbio, Italy), Paper C15, European Helicopter Society, 1993, pp. C15-1–C15-12.

[14]Duque, E. P. N., and Srinivasan, G. R., "Numerical Simulation of a Hovering Rotor Using Embedded Grids," *Proceedings of the 48th Annual Forum of the American Helicopter Society* (Washington, DC), Vol. 1, American Helicopter Society, Alexandria, VA, 1992, pp. 429–445.

[15]Baeder, J. D., Gallman, J. M., and Yu, Y. H., "A Computational Study of the Aeroacoustics of Rotors in Hover," *Proceedings of the 49th Annual Forum of the American Helicopter Society* (St. Louis, MO), American Helicopter Society, Alexandria, VA, 1993, pp. 55–71.

[16]Strawn, R. C., and Biswas, R., "Computation of Helicopter Rotor Noise in Forward Flight," *Journal of the American Helicopter Society*, Vol. 40, No. 3, 1995, pp. 66–72.

[17]Strawn, R. C., Biswas, R., and Lyrintzis, A. S., "Helicopter Noise Predictions Using Kirchhoff Methods," *Proceedings of the 51st Annual Forum of the American Helicopter Society* (Fort Worth, TX), Vol. 1, American Helicopter Society, Alexandria, VA, 1995, pp. 495–508; also *Journal of Computational Acoustics* (to be published).

[18]Lyrintzis, A. S., Koutsavdis, E. K., and Strawn, R. C., "A Comparison of Computational Aeroacoustic Prediction Methods," *Proceedings of the 2nd AHS International Aeromechanics Specialists' Conference* (Bridgeport, CT), Vol. 1, American Helicopter Society, Alexandria, VA, 1995, pp. 3-58–3-69; also *Journal of the American Helicopter Society* (to be published).

[19]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 3, 1981, pp. 357–372.

[20]Anderson, W. K., Thomas, J. L., and van Leer, B., "A Comparison of Finite Volume Flux Vector Splittings for the Euler Equations," AIAA Paper 85-0122, Jan. 1985.

[21]Srinivasan, G. R., Chyu, W. J., and Steger, J. L., "Computation of Simple Three-Dimensional Wing-Vortex Interaction in Transonic Flow," AIAA Paper 81-1206, June 1981.

[22]Yoon, S., and Kwak, D., "Three-Dimensional Incompressible Navier Stokes Solver Using Lower-Upper Symmetric-Gauss–Seidel Algorithm," *AIAA Journal*, Vol. 29, No. 6, 1991, pp. 874, 875.

[23]Kandula, M., and Buning, P. G., "Implementation of LU-SGS Algorithm and Roe Upwinding Scheme in OVERFLOW Thin-Layer Navier–Stokes Code," AIAA Paper 94-2357, June 1994.

[24]Barszcz, E., Fatoohi, R., Venkatakrishnan, V., and Weeratunga, S., "Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors," NASA Rept. RNR-93-007, April 1993.

[25]Wong, C. C., Blottner, F. G., and Payne, J. L., "A Domain Decomposition Study of Massively Parallel Computing in Compressible Gas Dynamics," AIAA Paper 95-0572, Jan. 1995.

[26]Taflin, D., Soetrisno, M., and Imlay, S., "A Parallel Algorithm for Hybrid Structured-Unstructured Grids," AIAA Paper 96-0287, Jan. 1996.