Concurrent-Processing Adaptations of a Multiple-Grid Algorithm

Gary M. Johnson* and Julie M. Swisshelm†
Institute for Scientific Computing, Fort Collins, Colorado
and
Swarn P. Kumar‡
Colorado State University, Fort Collins, Colorado

A multiple-grid algorithm for the accelerated solution of the Euler and Navier-Stokes equations is restructured to enhance its suitability for implementation on concurrent processors of both the single-instruction, multiple-data and the multiple-instruction, multiple-data type. Several of the resultant adaptations are used to carry out two-dimensional Euler and Navier-Stokes simulations on the CDC CYBER 205, the Cray X-MP, and the Denelcor HEP. Results are presented for subcritical and shocked supercritical inviscid flows and for turbulent viscous flow. These are compared with flow simulations obtained using the basic scalar, sequential multiple-grid algorithm. The performance advantages of the concurrent-processing adaptations are illustrated.

Introduction

THE recent past has seen substantial improvements in both computing hardware and fluid dynamics algorithm performance. Despite this progress, the computational fluid dynamicist's reach still far exceeds his grasp. For example, the requirements posed by the integrated design of airframe and engine components and the need for detailed, highly accurate flow-physics simulations clearly illustrate that much additional innovation in the fields of computer architecture and algorithm design is called for.

The architectural features of greatest interest in present- and next-generation supercomputers are those that make possible vector and parallel processing. These are each forms of concurrent processing. The sequential scalar processors used in previous-generation hardware may be referred to as being of single-instruction, single-data (SISD) architecture. SISD computation has been supplanted by concurrent processing because the technological limits for performance enhancement of SISD machinery are rapidly approaching. In order to maintain present hardware performance trends, various forms of concurrency are being exploited. Vector processors use single-instruction, multiple-data (SIMD) architecture to perform a single operation simultaneously on a collection of operands. Parallel processors represent the next step in generality. They use multiple-instruction, multiple-data (MIMD) architecture to perform simultaneously several distinct operations, with each operation accessing its own collection of operands. SIMD supercomputers include the Cray 1 and CDC CYBER 205. The Cray X-MP, Cray 2, CDC CYBERPLUS, Denelcor HEP, and ETA[10] are of MIMD type.

One notable recent trend in fluid dynamics algorithm design has been the adaptation of multiple-grid methods to the computation of Euler and Navier-Stokes flows. Research in this area has been pursued by Ni,[1] Johnson,[2,3] Jespersen,[4,5] Jameson,[6,7] and others.[8-12] At present, multiple-grid algorithms are being used to accelerate the convergence of steady flow simulations; however, it appears that their utility may extend to the time-accurate computation of unsteady flows.[13,14]

The introduction of multiple-grid methods preceded the arrival of modern concurrent processors. Consequently, their design has typically been based on the sort of sequential, scalar reasoning appropriate for SISD machines. The purpose of this paper is to examine one particular multiple-grid algorithm,[3,15] to restructure its information flow, thereby allowing the creation of alternative schemes suitable for concurrent processing, and to illustrate the performance of some of the resultant algorithms on SIMD and MIMD machines. The need for such restructured algorithms is particularly acute in cases where they will be implemented on MIMD machines having high multitasking overhead. Because the present aim is to be illustrative rather than exhaustive, results are presented for a select subset of the rather large range of possible combinations of model equations, algorithms, and concurrent-processing supercomputers. In particular, simultaneous-grid-updating algorithms for two-dimensional Euler and Navier-Stokes simulations are implemented on the CYBER 205, Cray X-MP, and HEP.

Model Equations

The two-dimensional model equations describing Euler, thin-layer Navier-Stokes, or full Navier-Stokes flows may be written in conservation-law form as

$$q_t = -(F_x + G_y) \tag{1}$$

where the appropriate definitions of F and G are as follows. For the full Navier-Stokes equations,

$$F = f - Re^{-1}p, \quad G = g - Re^{-1}r$$

while, for their thin-layer version,

$$F = f, \quad G = g - Re^{-1}s$$

and, for the Euler equations,

$$F = f, \quad G = g$$

Presented as Paper 85-1508 at the AIAA Seventh Computational Fluid Dynamics Conference, Cincinnati, OH, July 15-17, 1985; received Aug. 14, 1985; revision received Oct. 30, 1986.

*Director. Associate Fellow AIAA.
†Staff Scientist. Member AIAA.
‡Assistant Professor of Computer Science; presently Boeing Computer Services Company, Bellevue, WA.

The vector of conservation variables q, the inviscid flux vectors f and g, and the viscous stress vectors p, r, and s are defined as

$$q=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix},\quad f=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ (E+p)u\end{bmatrix},\quad g=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ (E+p)v\end{bmatrix}$$

$$p=\begin{bmatrix}0\\ \tau_{xx}\\ \tau_{xy}\\ \beta_x\end{bmatrix},\quad r=\begin{bmatrix}0\\ \tau_{xy}\\ \tau_{yy}\\ \beta_y\end{bmatrix},\quad s=\begin{bmatrix}0\\ \mu u_y\\ (\lambda+2\mu)v_y\\ \gamma\kappa Pr^{-1}e_y+(\lambda+2\mu)vv_y\end{bmatrix}$$

where

$$\tau_{xx}=\lambda(u_x+v_y)+2\mu u_x$$

$$\tau_{yy}=\lambda(u_x+v_y)+2\mu v_y$$

$$\tau_{xy}=\tau_{yx}=\mu(u_y+v_x)$$

$$\beta_x=\gamma\kappa Pr^{-1}e_x+u\tau_{xx}+v\tau_{xy}$$

$$\beta_y=\gamma\kappa Pr^{-1}e_y+u\tau_{yx}+v\tau_{yy}$$

Here, ρ, u, v, p, and E are, respectively; density, velocity components in the x and y directions, pressure, and total energy per unit volume. This final quantity may be expressed as

$$E=p/(\gamma-1)+\rho(u^2+v^2)/2$$

where the specific internal energy e has been related to the pressure and density by the simple law of a calorically perfect gas

$$e=p/[\rho(\gamma-1)]$$

with γ denoting the ratio of specific heats. The coefficient of thermal conductivity κ and the viscosity coefficients λ and μ are assumed to be functions only of temperature. Furthermore, by invoking Stokes' assumption of zero bulk viscosity, λ may be expressed in terms of the dynamic viscosity μ as

$$\lambda=-\tfrac{2}{3}\mu$$

Re and Pr denote the Reynolds and Prandtl numbers, respectively.

Although the equations of motion are written in Cartesian coordinates, this does not represent any loss of generality. Viviand[16] has shown that their strong conservation law form may be maintained under an arbitrary time-dependent transformation of coordinates. The generalized coordinate version of these equations, which is employed in the computations to be discussed subsequently, may be found in Steger.[17]

The thin-layer approximation is based on the observation that when a highly stretched mesh is used to resolve large gradients normal to a surface, computer limitations usually imply that diffusion terms parallel to the surface are calculated on such a relatively coarse mesh that they are essentially not resolved. Consequently, this approximation is implemented by using a body-fitted coordinate system and neglecting the viscous terms in the coordinate direction along the body. For Cartesian coordinates, with x representing the body-conforming coordinate, the thin-layer version of the Navier-Stokes equations is as given above.

The effects of turbulence are simulated by means of a two-layer algebraic eddy-viscosity model. The eddy viscosity is determined by the method of Baldwin and Lomax.[18]

Algorithms

The multiple-grid algorithms described here each consist of a fine-grid solution procedure and a coarse-grid acceleration scheme. The fine-grid procedure solves the unsteady equations of motion and may, if desired, do so in a time-accurate manner. A variety of implicit and explicit methods may be used to construct the fine-grid procedure. Here, because of its simplicity and ubiquity in computational aerodynamics, we choose to use the explicit, two-step Lax-Wendroff scheme known as MacCormack's method.[19] The fine grid is constructed such that the number of points in each direction is expressible as $n(2^p)+1$ for p and n integers such that $p\geq 0$ and $n\geq 2$, where p is the number of grid coarsenings and n is the number of coarsest-grid intervals. A

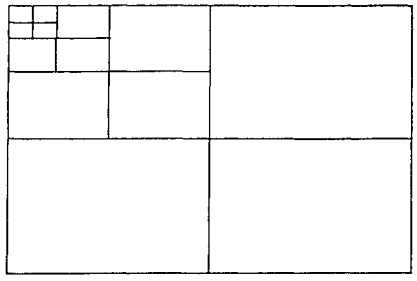

Fig. 1 Grid structure.

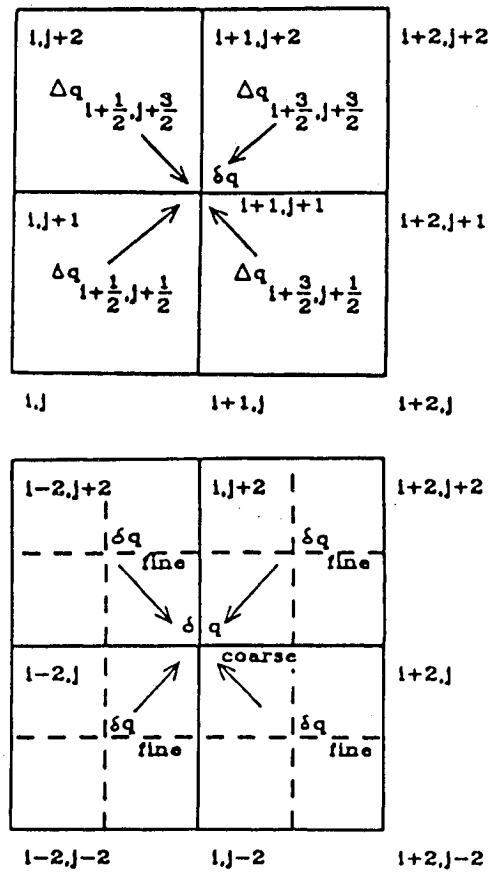

Fig. 2 Comparison of fine- and coarse-grid schemes: a) one-step Lax-Wendroff scheme on fine grid and b) course-grid scheme.

collection of successively coarser grids is then created by a recursive process that deletes every other point in each coordinate direction. The resulting grid structure is illustrated in Fig. 1.

Information is transferred from the fine grid to each of the coarser grids. This transfer may be accomplished either by a sequential cascading of information through successively coarser grids or by a simultaneous communication directly from the fine grid to all the coarser grids. In any case, a coarse-grid scheme is then used to rapidly propagate the resolvable components of this fine-grid information throughout the computational domain. A number of coarse-grid schemes are possible, some of which of which are described subsequently. All of these variations accelerate convergence to the steady state while maintaining the accuracy determined by the fine-grid discretization.

Fine-Grid Procedure

The forward predictor—backward corrector version of MacCormack's method may be written as

$$\Delta q_{i,j} = -\frac{\Delta t}{\Delta x}(F^n_{i+1,j} - F^n_{i,j}) - \frac{\Delta t}{\Delta y}(G^n_{i,j+1} - G^n_{i,j})$$

$$\delta q_{i,j} = -\frac{\Delta t}{2\Delta x}[(F^n_{i+1,j} - F^n_{i,j}) + (F'_{i,j} - F'_{i-1,j})]$$

$$-\frac{\Delta t}{2\Delta y}[(G^n_{i,j+1} - G^n_{i,j}) + (G'_{i,j} - G'_{i,j-1})]$$

where

$$\delta q_{i,j} = [q(t+\Delta t) - q(t)]_{i,j}$$

$$q'_{i,j} = q^n_{i,j} + \Delta q_{i,j}$$

$$F'_{i,j} = F(q'_{i,j}), \quad G'_{i,j} = G(q'_{i,j})$$

First derivatives in the viscous terms are backward-differenced in the predictor and forward-differenced in the corrector.

Coarse-Grid Schemes

Although it is quite probable that a large variety of coarse-grid acceleration schemes may be constructed, we limit our attention to those explicit schemes based on Lax-Wendroff methods. Such a coarse-grid scheme may be expressed as

$$\delta q_{\text{coarse}} = \Delta t q_t + \frac{\Delta t^2}{2} q_{tt}$$

By introducing Eq. (1), this may be rewritten as

$$\delta q_{\text{coarse}} = -\Delta t(F_x + G_y) - \frac{\Delta t^2}{2}(F_x + G_y)_t$$

Observing that

$$\Delta q = -\Delta t(F_x + G_y)$$

we obtain

$$\delta q_{\text{coarse}} = \Delta q - \frac{\Delta t^2}{2}(F_x + G_y)_t \tag{2}$$

Various coarse-grid schemes may now be derived, according to the way in which the second term on the right-hand side of Eq. (2) is treated.

If we let

$$-(F_x + G_y)_t = [A(F_x + G_y)]_x + [B(F_x + G_y)]_y$$

where A and B are the Jacobian matrices

$$A = \frac{\partial F}{\partial q} \qquad B = \frac{\partial G}{\partial q}$$

we obtain the class of Jacobian-based acceleration schemes, of which the method due to Ni[1] is a member.

If, on the other hand, we let

$$(F_x + G_y)_t \simeq \frac{1}{\Delta t}[(F_x + G_y)^{n+1} - (F_x + G_y)^n]$$

where

$$F^n = F(q^n), \qquad G^n = G(q^n)$$

$$F^{n+1} = F(q^n + \Delta q), \quad G^{n+1} = G(q^n + \Delta q)$$

we obtain the class of flux-based schemes introduced in Ref. 15. In both cases of schemes, Δq is approximated by a restriction of the fine-grid value of δq, and second-order accurate spatial differencing is used. For example, a simple Jacobian-based coarse-grid scheme may be written as

$$\delta q_{i,j} = \frac{1}{4}\left\{\left[\left(I + \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i-1,j-1} + \left[\left(I + \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i-1,j+1} + \left[\left(I - \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+1,j-1} + \left[\left(I - \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+1,j+1}\right\}$$

This scheme is contrasted in Fig. 2 with its one-step Lax-Wendroff analog, written on the fine grid. That one-step scheme may be written as

$$\delta q_{i+1,j+1} = \frac{1}{4}\left\{\left[\left(I + \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+1/2,j+1/2} + \left[\left(I + \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+1/2,j+3/2} + \left[\left(I - \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+3/2,j+1/2} + \left[\left(I - \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B\right)\Delta q\right]_{i+3/2,j+3/2}\right\}$$

where Δq is not approximated as a restriction of some δq, as in the coarse-grid scheme, but is rather computed as

$$\Delta q_{i+1/2,j+1/2} = -\frac{\Delta t}{2\Delta x}[(F_{i+1,j} + F_{i+1,j+1}) - (F_{i,j} + F_{i,j+1})]$$

$$-\frac{\Delta t}{2\Delta y}[(G_{i,j+1} + G_{i+1,j+1}) - (G_{i,j} + G_{i+1,j})]$$

Information Flow

In the sequential grid updating algorithm, illustrated in Fig. 3a, the solution is advanced over one multiple-grid cycle as follows. First, a fine-grid correction, δq_1, is computed. Then δq_1 is restricted to the next-coarser grid where δq_2 is computed. The δq_2 correction is both restricted to grid 3 and prolonged to grid 1, where it provides an additional update

to the fine-grid solution. On grids 3 through N-1, the procedure is analogous to that on grid 2. When δq_N has been computed and prolonged to grid 1 to provide the Nth update to the fine-grid solution, the next multiple-grid cycle is ready to begin.

Observe that when the components of the sequential grid updating algorithm (namely, the fine- and coarse-grid schemes) are both explicit, it is particularly easy to vectorize. However, the effectiveness of vectorizing the coarse-grid scheme is limited by the progressively shorter vectors that may be constructed on the successively coarser grids. Such an explicit sequential algorithm may also be run on an MIMD machine by splitting each grid, in turn, across the total number of processors available. An implicit sequential grid updating algorithm would probably vectorize less well and also require additional redesign to run on a parallel processor.

The parallel coarse-grid algorithm, illustrated in Fig. 3b, removes the dependence of grids 3 through N on their immediate predecessors. In particular, δq_1 is now restricted to each of grids 3 through N. All these coarse grids may then be updated simultaneously and independently of each other. This allows the mesh points on grids 2 through N to be assembled into one vector in order to improve performance on an SIMD computer. Alternatively, the coarse grids could each be updated simultaneously on separate processors of an MIMD machine. This would be attractive, for example, if the coarse-grid scheme were implicit.

A further possibility is the fully parallel algorithm, illustrated in Fig. 3c. Here δq_1 from the previous cycle is restricted to each of the coarse grids. This makes all the grids 1 through N independent of each other and allows their simultaneous update.

Implementation

The multiple-grid algorithms described in this paper may be implemented in a variety of ways. Several of the more significant choices made for the present study are described here. These choices concern the exclusion of dissipative effects from the coarse-grid schemes, the enforcement of boundary conditions, and the selection of restriction and prolongation operators. Considerations pertaining to vectorization and parallel processing are also discussed.

Convective Coarse-Grid Schemes

Dissipative effects have a local character, and their influence need not be taken into account in the construction of coarse-grid schemes. Rather, it is the convective terms, with their global character, that are the key element in coarse-grid propagation. Hence, coarse-grid schemes for viscous flow computations may be formulated on the basis of the inviscid equations of motion. Such a scheme leads to a multiple-grid convergence acceleration procedure that is independent of the nature of the dissipative terms retained in the viscous model equations. That is to say: A coarse-grid scheme based on the Euler equations may be employed, without modification, to accelerate the convergence of viscous flow computations based on the Navier-Stokes equations, the thin-layer equations, or any other viscous model equations that contain the full inviscid Euler equations.

Boundary Conditions

Boundary conditions are enforced only on the fine grid. This has the advantage of decoupling the coarse-grid scheme from both the physical and numerical nature of these boundary conditions. That is to say: The coarse-grid scheme always sees a Dirichlet problem. Any numerical damping terms that may be necessary are also applied only on the fine grid. This enhances the modularity of the coarse-grid scheme.

Restriction and Prolongation Operators

For all the results to be discussed subsequently, linear interpolation has been used as the prolongation operator. For the sequential grid updating algorithm, injection is used as the restriction operator. The parallel coarse-grid algorithm introduces averaging into the restriction operator for grids 3 through N. The fully parallel algorithm further employs underrelaxation of the fine-grid information restricted from the previous cycle.

Vectorization

As both the fine-grid solution procedure and the coarse-grid acceleration schemes used here are explicit, the resultant multiple-grid algorithms are readily vectorizable. Such vectorization of the sequential algorithm has been performed for computation on a CDC CYBER 205. First, the code was rewritten to take full advantage of the automatic vectorization performed by the CYBER 205 compiler. For the conservation vector q and the flux vectors F and G, four quantities must be computed at every point in the two-dimensional domain. These vectors are stored in three-dimensional arrays. The array indices were arranged in decreasing length from left to right, and the DO loops containing these indices were likewise ordered so that the innermost loop corresponds to the longest dimension (i.e., the first index), the second innermost loop corresponds to the next longest dimension (the second index), and so on. These modifications enable access to contiguous locations in the vectors so that the loops automatically vectorize. When nested DO loops result in access to every point in the two-dimensional domain, including the boundaries, entire matrices are treated as single long vectors containing the whole flowfield. With these changes, the code compiled with the automatic vectorization option ran approximately two to four times as fast as the code using only scalar optimization.

Further vector speedup was obtained by implementing CYBER 205 explicit vector FORTRAN. Bit vectors were created for use in WHERE blocks to control storage for vectorized computations that involved only the interior points of the domain. Dynamic storage was introduced so that temporary vectors could be used to reduce the operations count. Vector intrinsic functions (such as Q8VGATHR, Q8VCMPRS, etc.) were used to build contiguous vectors from the array elements needed on the coarse grids.

The parallel coarse-grid algorithm has been vectorized in a similar fashion, with the additional feature of combining the points to be updated on grids 2 through N into one long vec-

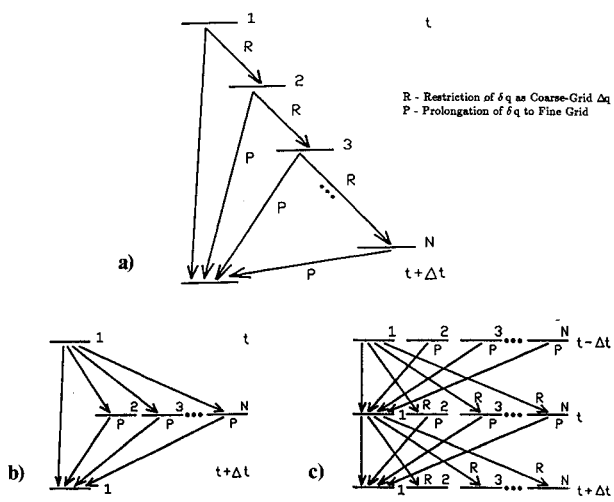

Fig. 3 Multiple-grid algorithm information flow: a) sequential algorithm, b) parallel coarse grids, and c) fully parallel scheme.

tor. This minimizes the vector startup overhead and thus further improves the peformance of the algorithm on the SIMD computer.

Multitasking

When attempting to multitask an algorithm for execution on an MIMD machine, we are concerned with multitasking overhead and algorithm granularity. By granularity we mean the time required to execute a multitaskable segment of the algorithm on a single processor.[20] For a given multitasking overhead, the best speedup is obtained when algorithm granularity is maximal. Large granularity is usually introduced by top-down programming, which exploits global parallelism in the algorithm. Bottom-up programming, on the other hand, exploits algorithm parallelism at a low level by making many partitionings, each on small code segments, such as DO loops containing independent statements.[21]

The sequential multigrid algorithm contains many opportunities for creating small granularity parallelism but relatively few opportunities for the sort of large granularity necessary to produce good multitasking speedup in the face of nontrivial multitasking overhead. This observation, together with the desirability of nonsequential multigrid schemes for reasons of algorithm flexibility, led to the construction of the parallel multigrid algorithms described above. In these algorithms, grids that are independent of one another may be updated simultaneously on separate processors. In fact, such a simple strategy may result in a poor load balance across processors because of the different amounts of work inherent in updating grids of different coarseness. However, more refined strategies are possible. Grids may, for example, be grouped together into tasks of approximately equal work, or they may be melded into tasks with other large-grained multitaskable code segments in order to equilibrate processor loading. Notice further that, by multitasking large-grained structures, the vectorization potential of code within these structures remains intact.

Results

To investigate the performance of the concurrent-processing adaptations to the basic multiple-grid algorithm, a number of preliminary computational experiments were performed.[22] The results of more comprehensive development and testing are reported here. For purposes of comparison, the test cases being used are the same as some of those previously employed in the development of the scalar processing version of the sequential grid updating algorithm.

The full Euler equations are solved for both subsonic and transonic flow. The thin-layer version of the Navier-Stokes equations is solved for attached and separated, laminar and turbulent, subsonic flows. All computations are performed in two dimensions. Extension of the parallel algorithms to the full Navier-Stokes equations or to three dimensions presents no essential difficulties.

Problem Specification

We consider the inviscid flow through an unstaggered cascade of bicircular arc airfoils at zero angle of attack, as illustrated in Fig. 4a, and the viscous flow through a similar cascade of sting-mounted airfoils, as shown in Fig. 4b. The boundary conditions used are also indicated in these figures. At the upstream domain boundary, total pressure, total temperature and flow angle are specified. At the downstream boundary, the static pressure is fixed. Along inviscid lateral boundaries, the tangency condition is applied while, along solid walls, the no-slip condition is applied and the temperature specified. Symmetry and periodicity are invoked to limit the size of the computational domain. Uniform flow at the isentropic Mach number implied by the ratio of exit static pressure to upstream total pressure is used as an initial state.

Table 1 Overall vector processing speedups

Case	Scalar sequential multigrid	Vector sequential multigrid	Vector parallel coarse-grid
Inviscid subcritical	1.0	2.99	3.18
Inviscid supercritical	1.0	2.89	3.04
Viscous turbulent	1.0	2.72	2.77

Table 2 Coarse-grid vector processing speedups

Case	Scalar sequential multigrid	Vector sequential multigrid	Vector parallel coarse-grid
Inviscid subcritical	1.0	1.67	1.80
Inviscid supercritical	1.0	1.73	1.87
Viscous turbulent	1.0	1.73	1.86

Table 3 Multitasked coarse-grid performance

	2 processors		4 processors	
Machine	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.78	0.89	3.06	0.77
Denelcor HEP I	1.59	0.80	—	—

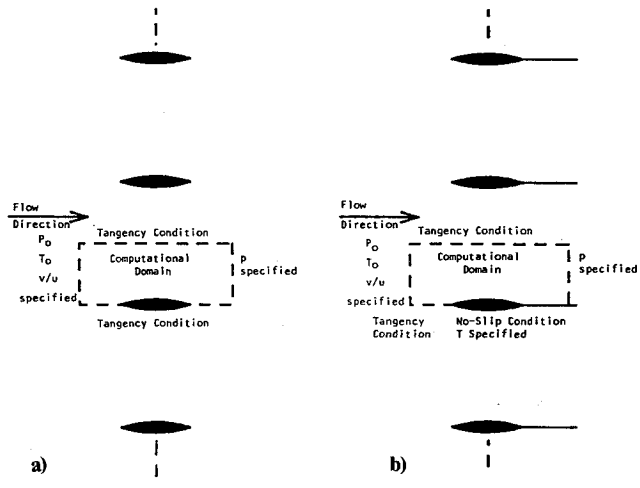

Fig. 4 Model problems: a) inviscid and b) viscous bicircular arc cascade problems.

For the subcritical cases, the ratio of exit static pressure to upstream total pressure is 0.8430191, yielding an isentropic upstream Mach number of 0.500 while, for the supercritical cases, a ratio of 0.7369520 is employed, resulting in an upstream Mach number of 0.675. In the viscous cases, the Reynolds numbers, based on cascade gap and critical speed, span the approximate range 8.4×10^3–2.0×10^5.

Details concerning the physical behavior of the test cases and the scalar performance of the sequential algorithm may be found in Refs. 3 and 15. It should be noted that all modifications to the algorithm presented here produce results identical to those obtained using the original explicit MacCormack method. More sophisticated applications of the sequential multigrid scheme are described in Ref. 23.

Table 4 Multitasked fine-grid performance

Machine	2 processors		4 processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.91	0.96	3.58	0.90
Cray X-MP with microtasking	1.93	0.97	3.78	0.95
Denelcor HEP I	1.59	0.80	3.10	0.78

Table 5 Multitasked complete scheme performance

Machine	2 processors		4 processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.87	0.94	3.30	0.83

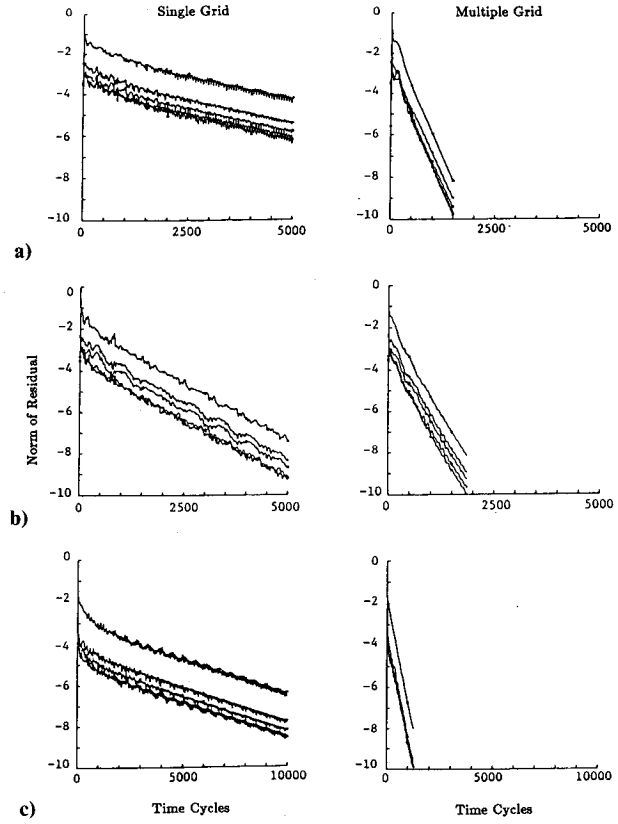

Fig. 5 Convergence histories: a) subcritical inviscid flow, b) supercritical inviscid flow, and c) turbulent viscous flow.

Algorithm Performance

The scalar sequential algorithm yields multiple-grid speedups of 6.9, 2.9, and 8.2 for selected inviscid subcritical, inviscid supercritical, and turbulent viscous flows, respectively. The grids used in these computations have on the order of 20,000 points. Convergence histories for these cases are shown in Fig. 5. Explicit vectorization of this algorithm results in vectorization speedups in the vicinity of 3.0, 2.9, and 2.7 for the respective cases cited above. It is expected that for more complex applications requiring finer meshes, vectorization performance will improve.

The parallel coarse-grid algorithm maintains essentially the same convergence behavior as the sequential algorithm. Consequently, the multiple-grid speedups obtained with it are virtually identical to those of the sequential algorithm. Vectorization of the parallel coarse-grid algorithm yields speedups of 3.2, 3.0, and 2.8 for the three test cases considered. The vectorization results are summarized in Tables 1 and 2. In Table 1, the overall speedups for the explicitly vectorized parallel coarse-grid scheme are recorded and contrasted with both the scalar and explicitly vectorized versions of the sequential coarse-grid scheme. Although the explicitly vectorized sequential scheme is highly efficient, a noticeable performance improvement results from use of the parallel scheme. Table 2 compares the performance of the parallel coarse-grid code segment with the analogous segments in the other versions. By this measure, the performance improvement over the explicitly vectorized sequential code is about 8%. One should note that a performance gain of this size is significant since the potential for further vectorization of the code is quite low. Furthermore, larger gains are expected when the parallel coarse-grid scheme is used in three-dimensional flow simulations, which effectively use the maximum vector length available on the CYBER 205.

The fully parallel scheme, in its present implementation, requires underrelaxation of the coarse-grid corrections and consequently does not perform as well as the other algorithms in terms of multiple-grid speedup. Efforts are presently under way to remove the underrelaxation requirement. Until these are complete, we will concentrate our SIMD and MIMD work on the sequential and parallel coarse-grid algorithms.

The theoretical maximum speedup on a p-processor MIMD machine is p. Varying overhead requirements of the multiple-grid algorithms will obviously result in distinct actual multitasking speedups. Using a top-down multitasking approach, the parallel coarse-grid algorithm has been implemented on a four processor Cray X-MP and on a Denelcor HEP I. Initially, only the coarse grids were multitasked so that the performance of parallel grids on a multiprocessor could be evaluated. Then the fine-grid computations were partitioned and multitasked, and the resultant code was integrated with the parallelized coarse grids. Load balancing of the entire scheme completed the study of performance resulting from the top-down approach.

Multitasking results are shown in Tables 3–5. The performance measures are based on a comparison of multitasked code segments with their unitasked analogs. The parallel coarse-grid scheme results contained in Table 3 were obtained with a five-grid multigrid sequence length on the X-MP and a three-grid sequence on the HEP. Observe that efficiencies of nearly 90% have been obtained using two processors but that efficiency deteriorates to 77% when four processors are used. This deterioration is a result of distributing multigrid structures containing unequal amounts of work across four processors, which creates a less than ideal load balance. Table 4 shows results obtained from multitasking the fine-grid scheme. The fine-grid tasks are fairly evenly balanced, and this code segment performs well on both two and four processors. Processor utilization of 90% or better on the X-MP and of nearly 80% on the HEP is achieved. When the four-processor case is recomputed on the X-MP using the low-overhead version of multitasking known as microtasking,[24] efficiency is improved to 95%. The fully multitasked multigrid algorithm performance on the X-MP is shown in Table 5. On two processors, 94% efficiency is obtained while, on four processors, with speedup by a factor of 3.3 over the unitasked code, efficiency is 83%.

Observe that our objective in this work has been to attain good multitasking performance by concentrating on the large-grained structures inherent in the algorithm rather than simply to take advantage of small-scale parallelism, for example, at the DO loop level. Our approach thus minimizes the overhead associated with multitasking and preserves opportunities for vectorization in each of the tasks.

Our next objective is to investigate further the bottom-up approach by microtasking the entire code. Although it exploits parallelisms on a small scale, microtasking on the X-MP incurs very low overhead and, as seen in the initial

results obtained for the fine-grid computations, promises highly efficient performance.

Conclusions

Several concurrent-processing adaptations of a multiple-grid algorithm for the accelerated solution of the Euler and Navier-Stokes equations have been introduced and examined.

By simultaneously updating as many grids as possible, while maintaining the convergence behavior of the basic sequential updating scheme, significant opportunities for vectorizing and multitasking the resultant algorithms are introduced.

The parallel coarse-grid algorithm allows the construction of one long vector across all the coarse grids. This counteracts the tendency toward progressively poorer vectorization performance on successively coarser grids.

The class of simultaneous grid updating algorithms introduces an alternative possibility for multitasking of multiple-grid methods. Instead of simply splitting each grid in turn across a number of processors, individual processors may (to cite the simplest example) each be assigned a particular grid.

The vectorized parallel coarse-grid code segment performed 80–87% better than the analogous scalar sequential code. The multitasking efficiency of the same segment was 80–89% on two processors. The utilization of four processors is a less efficient 77% when only the coarse grids are multitasked.

The multitasked fine-grid code segment yielded a speedup of 3.58 on four processors. With microtasking, this speedup increased to 3.78, thus producing an efficiency of 95%.

The multitasking efficiency of the complete algorithm run on four processors is 83%, while on two processors it is 94%.

Acknowledgments

The research reported here was funded by the Institute for Scientific Computing. Computer time was contributed by Cray Research, Denelcor, and the Los Alamos National Laboratory. All of this support is gratefully acknowledged.

References

[1]Ni, R. H., "A Multiple Grid Scheme for Solving the Euler Equations," AIAA Paper 81-1025, 1981.

[2]Johnson, G. M., "Multiple-Grid Acceleration of Lax-Wendroff Algorithms," NASA TM-82843, 1982.

[3]Johnson, G. M., "Multiple-Grid Convergence Acceleration of Viscous and Inviscid Flow Computations," *Journal of Applied Mathematics and Computation*, Vol. 13, Nov. 1983, pp. 375–398.

[4]Jespersen, D. C., "A Multigrid Method for the Euler Equations," AIAA Paper 83-0124, 1983.

[5]Jespersen, D. C., "Design and Implementation of a Multigrid Code for the Euler Equations," *Applied Mathematics and Computation*, Vol. 13, Nov. 1983, pp. 357–374.

[6]Jameson, A., "Solution of the Euler Equations for Two-Dimensional Transonic Flow by a Multigrid Method," *Applied Mathematics and Computation*, Vol. 13, Nov. 1983, pp. 327–355.

[7]Jameson, A. and Yoon, S., "Multigrid Solution of the Euler Equations Using Implicit Schemes," AIAA Paper 85-0293, 1985.

[8]Davis, R. L., "The Prediction of Compressible, Laminar Viscous Flows Using a Time-Marching Control Volume and Multigrid Technique," AIAA Paper 83-1896, 1983.

[9]Stubbs, R. M., "Multiple-Gridding of the Euler Equations with an Implicit Scheme," AIAA Paper 83-1945, 1983.

[10]Usab, W. J. Jr. and Murman, E. M., "Embedded Mesh Solutions of the Euler Equation Using a Multiple-Grid Method," AIAA Paper 83-1946, 1983.

[11]Dick, E., "A Multigrid Technique for Steady Euler Equations Based on Flux-Difference Splitting," *Lecture Notes in Physics*, No. 218, Springer-Verlag, Berlin, 1985, pp. 198–202.

[12]Koeck, C. and Chattot, J. J., "Computation of Three-Dimensional Vortex Flows Past Wings Using the Euler Equations and a Multiple-Grid Scheme," *Lecture Notes in Physics*, No. 218, Springer-Verlag, Berlin, 1985, pp. 308–313.

[13]Stubbs, R. M., Private communication, 1983.

[14]Jespersen, D. C., "A Time-Accurate Multiple-Grid Algorithm," AIAA Paper 85-1493, 1985.

[15]Johnson, G. M., "Flux-Based Acceleration of the Euler Equations," NASA TM-83453, 1983.

[16]Viviand, H., "Formes Conservatives des Equations de la Dynamique des Gaz," *La Recherche Aérospatiale*, Jan.–Feb. 1974, pp. 65–66.

[17]Steger, J. L., "Implicit Finite Difference Simulation of Flow About Arbitrary Geometries with Application to Airfoils," AIAA Paper 77-665, 1977.

[18]Baldwin, B. S. and Lomax, H., "Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, 1978.

[19]MacCormack, R. W., "The Effect of Viscosity in Hypervelocity Impact Cratering," AIAA Paper 69-354, 1969.

[20]Larson, J. L., "Practical Concerns in Multitasking on the Cray X-MP," Workshop on Using Multiprocessors in Meteorological Models, Reading, England, Dec. 3–6, 1984, ECMW.

[21]Larson, J. L., "Multitasking on the Cray X-MP-2 Multiprocessor," *Computer*, Vol. 17, July 1984, pp. 62–69.

[22]Swisshelm, J. M., Johnson, G. M., and Kumar, S. P., "Parallel Computation of Euler and Navier-Stokes Flows," *Applied Mathematics and Computation*, Vol. 19, July 1986, pp. 321–331.

[23]Chima, R. V., "Analysis of Inviscid and Viscous Flows in Cascades with an Explicit Multiple-Grid Algorithm," NASA TM-83636, 1984.

[24]Booth, M., "Multitasking—A Brief Design Summary," Cray Research, Mendota Heights, MN, Internal Memo, 1985.