AIAA JOURNAL
Vol. 33, No. 1, January 1995

Parallel, Scalable Parabolized Navier–Stokes Solver for Large-Scale Simulations

A. K. Stagg*
Cray Research, Inc., Pasadena, California 91109
D. D. Cline† and G. F. Carey‡
University of Texas at Austin, Austin, Texas 78758
and
J. N. Shadid§
Sandia National Laboratories, Albuquerque, New Mexico 87185

A massively parallel formulation for the solution of the parabolized Navier–Stokes equations has been developed for multiple instruction, multiple data (MIMD) computers. All functionality of the serial version of the code has been preserved in the parallel implementation, including grid generation, linear system solvers, and shock fitting. The computational domain is automatically decomposed and load balanced on individual processing elements. Performance timings are carried out for various MIMD architectures. Numerical tests indicate high parallel efficiency, with fixed solution time as the computational work is scaled with the number of processors.

I. Introduction

MASSIVELY parallel computers offer tremendous potential for satisfying the increased demand for high performance computing. The term massively parallel is used here to describe a computer architecture where several hundred autonomous processors are interconnected through a high bandwidth communication network such as a mesh, hypercube, or torus. Since the memory in such a machine is localized to each individual processor, messages must be passed along the communication interconnect for the sharing of nonlocal data. With interconnection topologies designed for hardware scalability, the performance of the ensemble can be scaled as more processors are added to the system. Scaling the network bandwidth using localized memories avoids the access bottleneck associated with shared memory architectures when additional processing elements are added to increase the computational performance.

Since massively parallel computers are scalable, application software written for these machines must also scale. The development of scalable software is a relatively new concept and is the major factor prohibiting the entrance of these new architectures into mainstream computing. In the field of computational aerodynamics, the development of a scalable application code has far-reaching consequences. All aspects of the parallel aerodynamic simulation must scale across multiple processors nonetheless maintaining the functionality of the serial aspects of the simulation. This functionality includes, for example, the generation of grids with distributed geometry, the construction of distributed linear systems, and the invocation of parallel solvers to solve the linear systems. In addition, a convenient interface to these machines must be developed which enables the user to store and retrieve large amounts of data as well as perform post-processing for scientific visualization and analysis.

The Beam–Warming algorithm has proven to be a popular technique for the solution of the Navier–Stokes equations and forms the basis for many aerodynamic simulators.[1] In the following sections we will describe the development of a highly parallel, scalable version of the Beam-Warming algorithm for implementation on massively parallel computers of the multiple instruction, multiple data (MIMD) type. We have incorporated our parallel analogue to the Beam-Warming algorithm into a massively parallel parabolized Navier–Stokes (PNS) code for the simulation of three-dimensional supersonic flowfields.[2] This algorithm is based on a spatial marching technique to generate solutions for steady, three-dimensional flows by solving a sequence of two-dimensional problems. Iterative solution techniques are described which are invoked to solve the distributed linear systems. We will describe the application of generalized coordinate techniques to distributed geometries and demonstrate the creation of scalable grids which automatically provide equal distribution of the computational work.

Received Feb. 17, 1994; revision received July 26, 1994; accepted for publication July 29, 1994.

*Computational Scientist, Jet Propulsion Laboratories. Student Member AIAA.

†Research Scientist, J. J. Pickle Research Campus, High Performance Computing Facility.

‡Professor, College of Engineering.

§Senior Member of the Technical Staff.

II. Parabolized Navier–Stokes Formulation

The PNS equations have been successfully applied to the solution of three-dimensional, steady flow past complex configurations.[3,4] The equation set represents a mixed hyperbolic/parabolic system for which a spatial marching scheme originating from an initial plane of data is well posed. Let (ξ, η, ζ) denote a generalized coordinate system with (x, y, z) the physical coordinates. If ξ is taken to be the marching direction, with η and ζ defined to be the normal and cross plane directions, respectively, then the PNS equations can be expressed in the form

$$\frac{\partial \boldsymbol{E}}{\partial \xi} + \frac{\partial \boldsymbol{F}}{\partial \eta} + \frac{\partial \boldsymbol{G}}{\partial \zeta} = 0 \tag{1}$$

where the respective flux vectors can be decomposed to inviscid and viscous contributions as

$$\boldsymbol{E} = \frac{1}{J}[\xi_x \hat{\boldsymbol{E}} + \xi_y \hat{\boldsymbol{F}} + \xi_z \hat{\boldsymbol{G}}] \tag{2a}$$

$$\boldsymbol{F} = \frac{1}{J}[\eta_x(\hat{\boldsymbol{E}} - \hat{\boldsymbol{E}}_v) + \eta_y(\hat{\boldsymbol{F}} - \hat{\boldsymbol{F}}_v) + \eta_z(\hat{\boldsymbol{G}} - \hat{\boldsymbol{G}}_v)] \tag{2b}$$

$$\boldsymbol{G} = \frac{1}{J}[\zeta_x(\hat{\boldsymbol{E}} - \hat{\boldsymbol{E}}_v) + \zeta_y(\hat{\boldsymbol{F}} - \hat{\boldsymbol{F}}_v) + \zeta_z(\hat{\boldsymbol{G}} - \hat{\boldsymbol{G}}_v)] \tag{2c}$$

The detailed form of the components of the individual flux vectors are described elsewhere[5] and are not critical to the present development. In this notation, J represents the Jacobian of the generalized transformation, whereas ξ_x, η_x, ζ_x, etc. are the metric coefficients. The solution vector consists of five unknowns,

$$\boldsymbol{U}^T = [\rho \quad \rho u \quad \rho v \quad \rho w \quad \rho e_t] \tag{3}$$

for the density, three components of momentum, and total energy of the flowfield.

The PNS marching procedure is implemented as an initial-value problem where U is spatially evolved in ξ using a parallel analogue to the Beam-Warming algorithm. The computational domain for each spatial marching step is taken to be the cross-plane region bounded between the body surface and bow shock. At the body surface, no-slip conditions are specified, whereas an explicit shock fitting procedure is used to construct the outer bow shock in accordance with the Rankine-Hugoniot relations. Details of the serial version of the code may be found in Ref. 6.

III. Parallelization Issues

The topology of the computer architecture refers to the interconnection scheme which links the processors and defines the communication paths within the machine. The hypercube has been used successfully to design large-scale massively parallel systems. Functionally, the hypercube interconnect offers considerable flexibility from the program developer's viewpoint. Communication channels and neighboring processors can be defined using a binary gray-code algorithm which encompasses several standard geometrical configurations[7]. These include a one-dimensional ring, a two-dimensional mesh, as well as a three-dimensional mesh. One of the primary advantages of the hypercube is that global communication can be carried out in $\mathcal{O}(\log_2 p)$ operations, where p is the number of processors. The primary disadvantages are that the number of processors can be scaled only in powers of 2, and the number of interconnections per processor is relatively high.

In recent years, higher communication bandwidths have been achieved, enabling simpler interconnection schemes to be explored. Mesh-based architectures offer simpler parallel programming and scalability to a large number of processors. In contrast to the hypercube, the processors are not so tightly interconnected, and thus global operations may be more expensive. Communication between subdomains can occur locally between neighboring processors along the interconnect, or nonlocally, with intermediate hops. Nonlocal communication typically occurs in global operations such as a global summation, global maximum, or global minimum of a scalar quantity. Messages passed with intermediate hops incur a latency penalty proportional to the number of hops between the source and destination processors. On mesh architectures, routing of messages may also encounter contention, whereby messages from differing processors compete for the same communication link. Message contention can lead to serious communication penalties, thus algorithms designed for mesh architectures should avoid these situations. For subdomains which are physically adjacent in computational space, it is clearly desirable to assign these regions to processors which reside as nearest neighbors in the interconnect.

Load balancing refers to the manner in which the computational work is distributed over the ensemble of processors. With each processor in the MIMD computer working autonomously, synchronization occurs only when messages are passed between processors. As such, the overall performance of the ensemble is limited by the slowest processor. Distributing the workload uniformly reduces the frequency of wait states for incoming messages and increases the overall efficiency of the parallel algorithm.

IV. Generalized Coordinates and Domain Decomposition

We have developed a domain decomposition and load balancing strategy which satisfies the parallel criteria already outlined and simultaneously providing a functionality which is equivalent to that of the serial version of the PNS scheme. To implement the marching procedure in parallel, we define generalized coordinates (ξ, η, ζ) which act as global distributed coordinates in the computational domain. The mapping procedure between physical space and computational space is shown in Fig. 1. As with serial versions of PNS, the flowfield cross plane bounded by the body, shock, and symmetry planes is mapped to a square computational domain of unit dimension. For convenience we have shown here a cross plane coincident with the (y, z) plane. However, in general this may be a curved surface in (x, y, z) space.

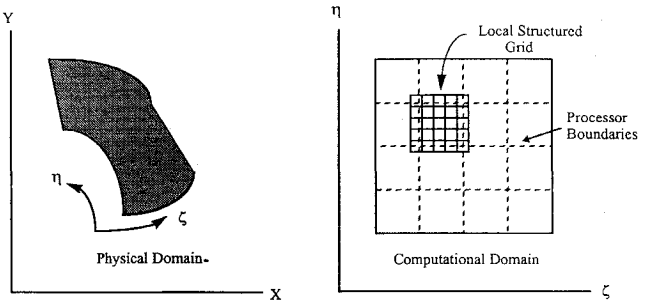

Fig. 1 Application of distributed generalized coordinates for automatic domain decomposition and load balancing of irregular two-dimensional domains.

The mapped (η, ζ) domain is easily decomposed to a uniform rectangular partition of subdomains, where each subdomain is to be mapped to the topology of the massively parallel architecture. In the present application, we have used a binary gray-code algorithm to map the subdomains onto a two-dimensional mesh embedded within a hypercube topology. This defines the nearest neighbor protocol for data communication between adjacent subdomains for a hypercube interconnect. A uniform structured grid can then be generated locally within each subdomain (a block-structured grid scheme). This implies equal load balancing across subdomains. Similar mapping schemes may be developed for other processor interconnections. Consequently, the computational work per subdomain is constant, and load balancing is preserved. Nearest neighbor communication is enforced in the mapping procedure as geometrically adjacent regions in physical space are mapped onto nearest neighbor processors in the computational space.

In addition to the distributed computing requirements, the decomposition strategy for the parallel PNS scheme has to be based at the processor level. Because of the spatial evolution of the PNS solution surface, the boundaries of the physical domain change shape to conform to the local body surface coordinates and shock position. Therefore, simple static decompositions of a mesh will not suffice. As with most computational fluid dynamics (CFD) applications, it is further required that the mesh be directionally biased to locally refine the grid where gradients in the flow are large. The parallel grid generation procedure must satisfy local and global constraints. Each processor must be able to generate a local grid independent of the neighboring processors. The grid must provide overlap of the computational subdomains so that adjacent processors share boundary information. The mesh points in physical space must be biased near the body to resolve thin viscous regions. Each of these requirements are satisfied by the fact that the (η, ζ) coordinates are globally defined across the domain, yet distributed among the processors. Therefore, functional relationships expressed in terms of η and ζ are automatically parallelized in the physical space. Such relationships include, for example, algebraic stretching functions for the distribution of grid points in the global mesh. Since the mesh generation is performed at the processor level, scalability may be achieved by simply specifying the number of grid points which make up each subdomain and the number of processors in each coordinate direction. The grid generation procedures we have developed have equal functionality on anywhere from 1 to 1000 processors with no special interface required by the user.

Figure 2 shows an algebraic grid for a typical cross plane of a conical body at angle of attack. The different shading of the local grids delineates the subdomains in physical space which are mapped onto different processors. This grid was constructed on 256 processors of an nCUBE/2 hypercube. The hypercube was configured as a two-dimensional mesh with 16 processors aligned in each coordinate direction. The global grid size is 130×130. It is important to note that there are an equal number of grid points in each subdomain, although the global mesh is clearly biased in the direction of the body surface.

In the PNS application, cross-plane grids are generated in parallel at each new marching step using updated coordinates for the body surface and bow shock location. Since the bow shock is advanced

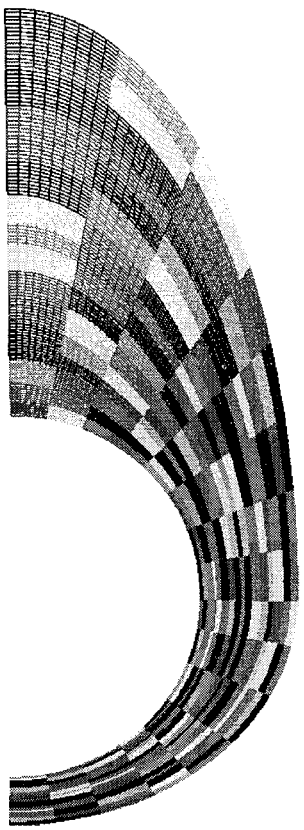

Fig. 2 Parallel grid generation and subdomain distribution.

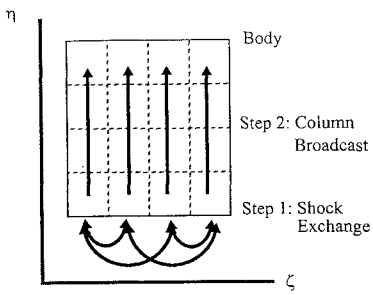

Fig. 3 Broadcast of shock Courant condition.

explicitly, a restriction on the maximum allowable spatial step is calculated using the following Courant–Fredricks-Lewy (CFL) condition:

$$\mathrm{CFL} = \frac{|\lambda|\Delta\xi}{\Delta\eta} \leq 1 \tag{4}$$

where λ represents the maximum eigenvalue of the Euler equations along the perimeter of the shock boundary.[6,8] The maximum eigenvalue λ is determined through a binary exchange localized to the row of processors at the shock boundary. The Courant step size stored by each shock boundary processor is then broadcast along columns to all intervening processors between the shock and wall boundaries. This two-step process is illustrated in Fig. 3.

V. Parallel Beam-Warming Factorization

The parallel implementation of the Beam-Warming algorithm for the cross-plane flow solves follows the traditional alternating-direction implicit (ADI) method with extensions to a distributed memory architecture. In traditional ADI, the multidimensional differential operator is approximately factored into components which correspond to a series of linear systems for each coordinate direction. In PNS applications, this factorization produces independent 5×5 block tridiagonal systems which correspond to one-dimensional solutions along lines in the η and ζ directions. Information from the normal (or cross-plane) coordinate direction is used explicitly and appears on the right-hand side (RHS) of the system of equations. The MIMD implementation of the ADI algorithm solves independent lines in parallel. Local memory limits may be a serious consideration for some parallel domain decomposition schemes. In the present case, each line is decomposed across the processors. Independent processors work in parallel to construct the elements of the sparse 5×5 block tridiagonal matrix for each line. Locally, on each processor, a subset of the equations associated with a subinterval of an entire global line is generated. A parallel line solution is carried out for each of the independent block tridiagonal systems. Thus, at any step of the algorithm, subsets of the processor array are cooperating on the solution of a particular line in parallel with other subsets of processors solving other independent lines. After all line solves for a particular coordinate direction have been completed, the algorithm is repeated for the other coordinate direction. Further details are provided in the next section.

VI. Distributed Sparse Matrix Solution

There have been a number of basic solution techniques which have been successfully applied to the distributed block tridiagonal line solves. Here we shall describe three such methods for which performance results will be presented.

Parallel Block Cyclic Reduction

As a direct solver for the distributed 5×5 block tridiagonal systems, a parallel version of the block cyclic reduction (BCR) algorithm has been developed.[9] This algorithm proceeds by eliminating the blocks associated with all of the even grid points in a particular line. The remaining unknowns are then renumbered, and the algorithm is applied recursively until just two blocks of unknowns remain. This small system is then solved directly. The solution of unknowns is determined by a back substitution method. Since the parallel BCR technique is a true direct method it is the most robust of the solution algorithms implemented in this study. However, the maximum amount of parallelism of this algorithm is limited due to the idling of processors as the elimination process proceeds with fewer blocks to eliminate than the number of available processors. In addition, implementation of the algorithm is relatively difficult due to the complexity of the nonlocal communication of matrix coefficients and vector results.

Next, two iterative solution techniques are described. These methods are generally characterized by simpler local communication. However, they require iteration and, thus, more operations necessary to obtain the solution. On a serial computer there would be no question that the direct method would be faster. However, in the parallel environment the impact of communication and load imbalance must also be considered in their relative evaluation. This is the question explored next by our implementation of the two iterative methods.

Localized Direct Solver

We have developed a parallel iterative solver based on a serial version of the block Thomas algorithm. In the original serial version of the PNS code, a block Thomas algorithm is used to solve the block tridiagonal systems which arise in the ADI formulation. The development of the parallel localized direct (LD) solver was motivated by an interest in investigating the performance and feasibility of implementing the same serial block Thomas algorithm locally on each processor. This new solver is a hybrid method in which localized direct solves are performed in conjunction with an iteration procedure to update information on the boundaries between processors.

Consider the block tridiagonal matrix distributed over two processors as illustrated in Fig. 4. The upper left portion of the matrix resides on processor X, while the lower right portion resides on processor Y. The physical grid points are denoted by the dark circles. Each open circle is a ghost point which contains overlap information corresponding to the next physical grid point on the neighboring processor. The set of equations corresponding to the three-point difference stencil at grid point 3 on processor X is given by

$$[A]_{2x}\boldsymbol{U}_{2x} + [B]_{3x}\boldsymbol{U}_{3x} + [C]_{1y}\boldsymbol{U}_{1y} = \mathrm{RHS} \tag{5}$$

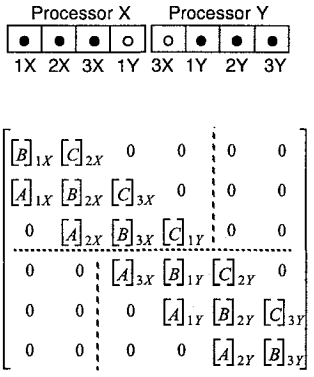

Fig. 4 Distributed block tridiagonal matrix.

Of course, U_{1y} resides on processor Y and will be determined by processor Y; but for the moment consider U_{1y} to be known by processor X. Then the set of equations can be written as

$$[A]_{2x}U_{2x} + [B]_{3x}U_{3x} = \text{RHS} - [C]_{1y}U_{1y} \tag{6}$$

The same reasoning is applied to processor Y, where the $[A]_{3x}U_{3x}$ terms are considered known and subtracted from the corresponding elements of the right-hand side vector. Following this operation, the submatrix on each processor is block tridiagonal in form, and the serial block Thomas algorithm can be applied locally on each processor in parallel. Since U_{1y} and U_{3x} are not known in practice, an initial guess is made for these terms by processor X and processor Y, respectively. The right-hand side at the processor boundaries is modified, and an intermediate solution is obtained. Message passing is introduced to update the ghost locations with the most recent values of U_{1y} and U_{3x}. An updated, modified right-hand-side vector is then computed, and the block Thomas algorithm is applied to obtain the next intermediate solution. This process is repeated iteratively until the solution converges along a given line. Conceptually, this algorithm reduces to a block Jacobi method in the limit of one real grid point and two ghost points per processor. However, it is impractical to run the application with such a fine granularity due to efficiency limitations encountered through Amdahl's law.[10]

The attractive feature of the LD solver is that its parallel implementation only requires the modification of the right-hand-side vector at the processor boundaries and the introduction of message passing so that these modified terms can be updated iteratively. The code for the serial block Thomas algorithm is not altered. Since the procedure is iterative in nature, a global convergence test which involves message passing must be implemented as well.

Quasiminimum Residual Conjugate Gradient Squared Method

The transpose-free quasiminimum residual conjugate gradient squared (QMR-CGS) method is a Krylov subspace method for the iterative solution of sparse nonsymmetric linear systems.[9,11] This algorithm incorporates a simple recurrence relationship for constructing the Krylov basis. As a preconditioner a local lower-upper (LLU) factorization of the distributed matrix is used. This technique uses a simple domain decomposition of the line and forms a reduced system on each processor. The subdomain problems are then decoupled by using old values of the dependent variables to eliminate contributions from other processors. Using this technique the LLU preconditioner requires the same communication at each iteration as the standard Jacobi preconditioner. In effect, this method corresponds to a block Jacobi technique with the block defined by the equations corresponding to each of the internal grid points on the subinterval. However, we generalize this concept by using old LU factors from previous line solves as approximate factors to precondition subsequent line solves. This leads to a reduced factorization form of the LLU preconditioner and saves the computational expense associated with the construction of new LU factors for each line solve.

VII. Primary Operations in Parallel Parabolized Navier–Stokes

The parallel PNS algorithm retains all functionality of the serial algorithm, including linear system solves, grid generation, and automatic determination of the marching step size. The key steps of the parallel implementation are as follows.

1) Processor identification numbers and the gray code for the hypercube interconnect are generated.

2) Processor 0 reads flowfield and geometry information from the input file and broadcasts the data to all processors.

3) Initial starting planes are constructed in parallel, including the grid and solution vector at each point. Starting flow solutions can be input in parallel from a distributed file system or a default solution profile may be assumed.

4) A binary exchange is carried out between shock boundary processors to determine the maximum allowable step size. These processors broadcast the step size and the computed shock slope along the columns of processors.

5) Grid and metric coefficients are constructed for the current marching plane. Interprocessor communication is invoked to update the ghost point locations for the metric terms.

6) Right-hand-side vectors and smoothing terms are constructed at each grid point.

7) Block tridiagonal systems are constructed locally along $\eta =$ const lines and loaded into sparse matrix format.

8) Independent systems are solved in parallel in the first step of the ADI scheme to obtain the intermediate solution vector.

9) Matrix multiplication is performed to construct the right-hand-side vectors for line solves in the alternate direction.

10) Block tridiagonal systems are constructed locally along $\zeta =$ const lines and loaded into a sparse matrix format.

11) In the second step of the ADI scheme, independent systems are solved in parallel for the solution vector increment, which is added to the previous marching plane solution to obtain the solution for the current marching plane. Communication is invoked between neighboring processors to update ghost point locations with the new solution vector.

12) Processors repeat the procedure beginning with step 4 for the next marching plane.

VIII. Discussion of Results

The parallel PNS code has been tested for performance and scalability to demonstrate the capabilities of massively parallel aerodynamic simulations. In the following discussion, we will present some typical results that we have obtained in our initial investigations. Validity of the parallel solution was determined by detailed comparisons with the serial version of the code. All results are for 64-bit precision calculations.

Scaled Performance

Scalability tests have been performed for the three solution methods discussed in Sec. VI. In conducting scalability tests, the granularity of the computational work per processor is held fixed as the global problem size grows in proportion to the number of processors. Solution results are presented in Table 1 for the LD, QMR-CGS, and BCR algorithms.

The QMR-CGS calculations were performed with the offset equal to 10. That is, the LLU preconditioning factors in the QMR-CGS algorithm were recomputed every 10 line solves. For some decompositions there were fewer than 10 line solves per processor and, thus, the LU factors were computed only once per processor for

Table 1 Solution time per marching step for various solvers on the *n*CUBE/2 hypercube, s

Processors	1	4	16	64	256	1024
Global Grid	16^2	32^2	64^2	128^2	256^2	512^2
LD	1.49	4.07	4.71	5.00	4.82	4.53
QMR-CGS (offset = 10)	7.94	8.44	8.33	8.34	8.16	8.30
BCR	3.00	3.37	3.56	3.73	3.91	4.09

Table 2 Solution time per marching step for scaled decompositions on the *n*CUBE/2 hypercube, s

Grid	Number of processors: 1	4	16	64	256	1024
1024^2	—	—	—	—	190.9	50.9
512^2	—	—	—	189.8	51.3	14.7
256^2	—	—	188.9	51.3	14.7	4.5
128^2	—	186.1	50.9	14.6	4.5	—
64^2	164.3	49.9	14.9	4.5	1.7	—
32^2	45.0	14.2	4.4	1.6	—	—
16^2	13.4	4.2	1.5	—	—	—
8^2	4.3	1.5	—	—	—	—

Table 3 LD performance for scaled decompositions on various high performance computers, s

Machine	Proc. / Grid	1 32^2	4 64^2	16 128^2	64 256^2
*n*CUBE/2		5.70	15.22	17.02	17.25
Intel iPSC/860		3.49	5.70	6.18	6.30
Intel Delta		3.46	5.54	5.79	5.92

each direction in these cases. In general, for a fixed-size problem, the number of iterations required for convergence increases with the number of processors assigned to the solution.[2] On a typical solution, the average number of iterations by the QMR-CGS algorithm per line solve was less than 15. Solution times are reported as the time necessary to evolve the solution plane a single marching step for flow past a conical body at angle of attack. This time includes the time required to generate the cross-plane grid, compute the metric coefficients, set up the sparse matrix system, and solve to convergence. Approximately 75% of the total time is spent in the line solves, and about 8% of the total time is spent in communication outside the solver. The remainder of the solution time is in setting up the grids and the systems of equations. In general, these percentages are approximate and will vary according to the global grid size and number of processors. Overall, the results indicate the LD solver and the BCR solver outperform the QMR-CGS iterative solver by a factor of two. However, experience has shown the LD solver to be less robust than the other solvers in the sense that the number of iterations to converge each system may depend upon the flow conditions and the amount of artificial viscosity added to the system. The BCR algorithm is considerably more difficult to implement in parallel than the QMR-CGS method, however, it has proven to be the most robust solver. One difficulty with the current version of BCR is that it requires $2^n + 1$ global grid points, and a significant modification in the parallel code was necessary to handle this limitation. It is most important to compare the scaled performance of the parallel PNS algorithm as the global problem size is expanded. The solution time for all solution methods is essentially constant for the parallel code. Using all 1024 processors of the *n*CUBE/2, the parallel PNS code is evolving over 1.3×10^6 unknowns every 4 s with the BCR solver. A summary of timings shown in Table 2 demonstrates the scalability of the PNS algorithm for both fixed and scaled problem sizes. A conjugate gradient squared (CGS) algorithm was used in the generation of these results.

Comparison of Machines

The present PNS application was originally developed on Sandia's 1024-node *n*CUBE/2 hypercube. The *n*CUBE has a configuration of 4 MB per node with each processor having a clock speed of 20 MHz. Peak performance for each node is rated at 2.7 mflops in double precision floating point. This application has since been ported to the Intel iPSC/860, 64-node hypercube. The Intel machine is based on the i860 processor. Sandia's installation consists of 33-MHz processors each having a peak 64-bit precision performance of 50 mflops. Sandia's iPSC/860 is configured with 32 MB per node on the lower 16 processors, whereas the remaining 48 processors have 8 MB each.The Intel version of the PNS procedure has also been successfully executed on Caltech's Intel Delta. The Delta machine is based on Intel's 40-MHz i860 processor. The Delta has 576 of these processors arranged as a 16×36 mesh. Each node of the Delta has 16 MB of local memory. The high bandwidth communication channels linking the nodes operate at 11-MB/s. For the Delta implementation, gray-coded communication channels were calculated based on a hypercube mapping to the two-dimensional mesh and thus non-neighbor communication was encountered throughout the execution. In porting from the Intel iPSC/860 to the Delta, only one line of code was changed so that the two-dimensional mesh could be allocated in powers of 2 to support the gray-code mapping scheme.

The performance timings shown in Table 3 are provided for the LD solver. This version of the code is written entirely in Fortran and was compiled using the highest optimization levels available on each machine. We consider the Delta results to be preliminary as no attempt has been made to optimize the code for the two-dimensional mesh interconnect. The Delta timings are included here for completeness. Since the test problem was identical to that used in developing Table 1, comparisons between the two tables are relevant.

Sample Calculation—Large-Scale Conical Flows

To demonstrate the performance capabilities of massively parallel systems, we present PNS calculations obtained using extremely fine grids. The fine grid results discussed in this section were obtained using 256 processors of an *n*CUBE/2 hypercube.

The geometry of interest is the 7.5-deg half-angle cone at 22.6-deg angle of attack in a Mach 2.94 flow. These flow conditions are very similar to those in the experimental cone studies conducted by Nebbeling and Bannink.[12] However, a turbulence model has not been implemented in this PNS code, so the Reynolds number based on cone length was reduced from 6,750,000 to 500,000 in our simulation. In spite of this difference in Reynolds number, flow features we have observed are qualitatively very similar to the findings of Nebbeling and Bannink.

The largest parallel PNS calculation we have performed involves a 513×513 grid over half of the cross plane (1,315,845 unknowns per marching plane). For each marching plane, solutions were generated in approximately 12 s. The initial solution at $\xi = 0.1$ was marched approximately 10,800 steps to reach the end of the cone at $\xi = 1$.

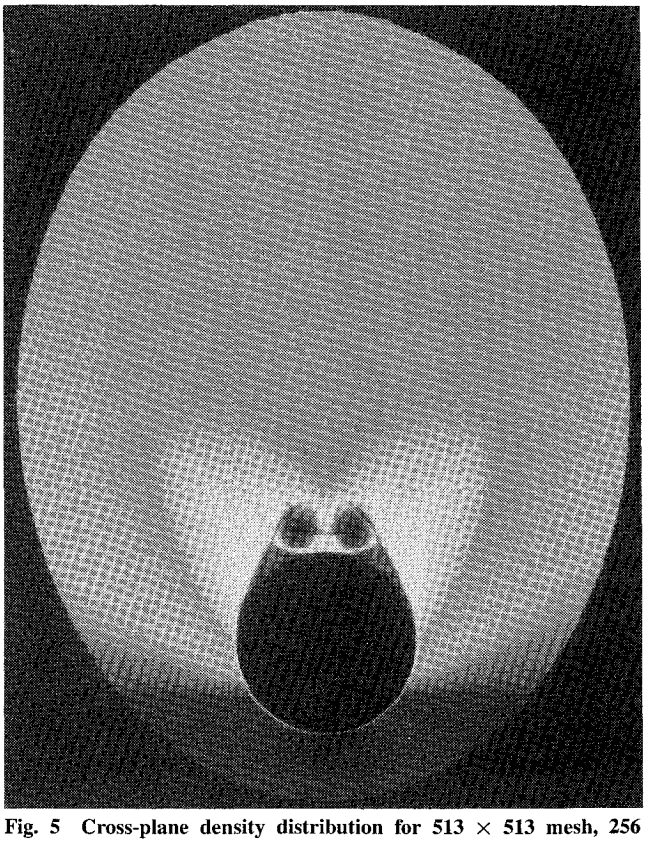

Fig. 5 Cross-plane density distribution for 513 × 513 mesh, 256 processors.

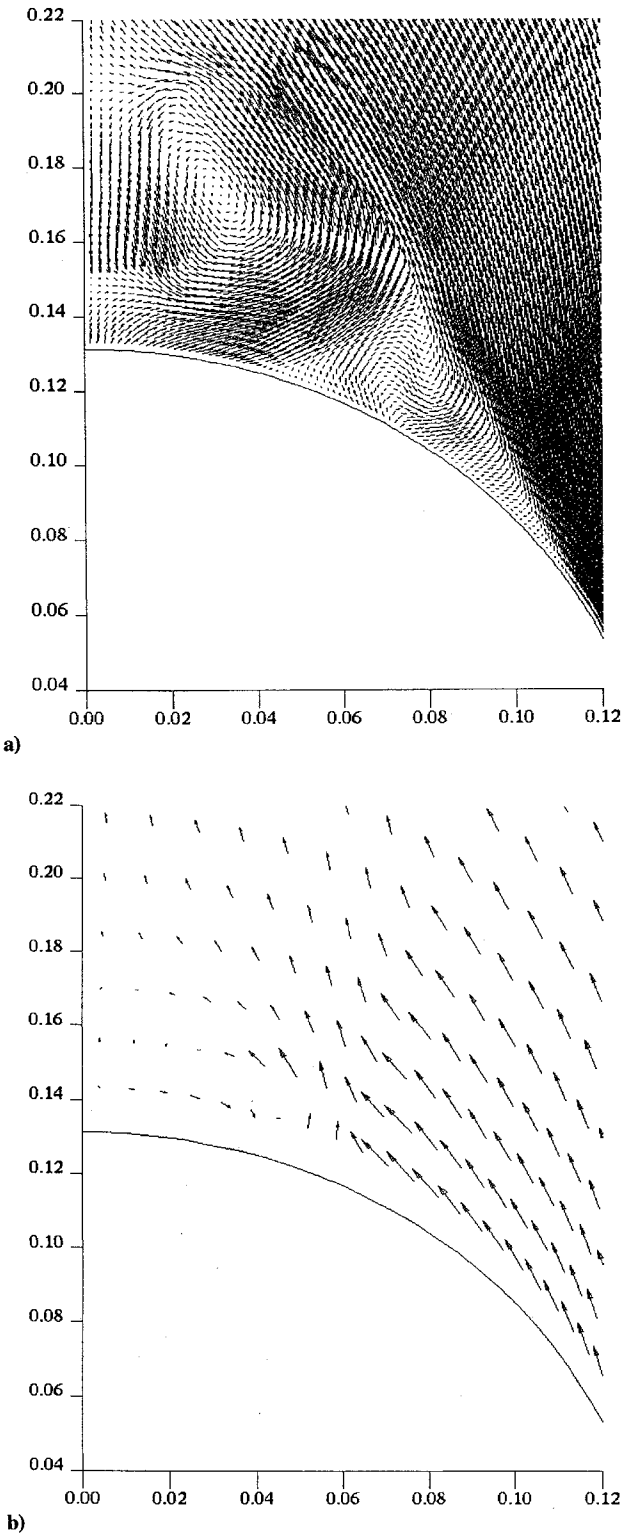

Fig. 6 Velocity vector plot: a) 513 × 513 mesh and b) 65 × 33 mesh.

A starting solution for a 65 × 65 grid was interpolated bilinearly to the 513 × 513 grid to start the fine grid calculation. The addition of second-order implicit smoothing was used to damp errors associated with the interpolation and was turned off after the first 30 marching steps.

The cross-plane density shown in Fig. 5 corresponds to the axial location at the end of the cone. The large primary vortices are visible, as well as numerous imbedded secondary shock waves. Also, a normal shock wave sitting between the primary vortices and parallel to the leeward cone surface is clearly observed. This normal shock wave was observed by Nebbeling and Bannink in schlieren photos. It was also detected by Degani and Schiff in a turbulent PNS calculation but was not clearly resolved.[13]

A cross-plane velocity vector plot is shown in Fig. 6a for the same axial location. The flow region shown is the leeward side of the cone near the primary vortex. In this figure every other data point was dropped in each coordinate direction so that the velocity arrows would be visible. The core of the primary vortex is easily observed. No secondary vortex core is visible; however, a complex region of counter-rotating flow has been resolved adjacent to the primary vortex. The normal shock wave between the primary vortices as well as other secondary shocks are visible in this velocity vector plot. In Fig. 6b, the same vector plot is shown for a mesh which is 33 points normal the body and 65 points in the circumferential direction. This size of mesh is typical of PNS solutions run on conventional workstations and small mainframe computers. Notice, by comparison, the location and size of the primary vortex is quite different in the coarse mesh solution. The lack of boundary-layer resolution is also apparent, resulting in the loss of resolution of the counter-rotating flow structure near the wall. The cross-plane shock which bridges the two primary vortices is also not apparent, as it is in the 513 × 513 vector plot.

IX. Conclusions

We have developed a highly parallel, scalable version of the parabolized Navier–Stokes solution algorithm. The primary goal of this effort has been to demonstrate how conventional serial algorithms and solution procedures for aerodynamic applications can be implemented into a massively parallel computing environment. Our results demonstrate that a high degree of parallel efficiency can be extracted from algorithms of this type. Fixed solution times can be achieved as the problem is scaled over an increasing number of processors. A nodal-based grid scheme has been developed as part of this effort which relieves the user from dealing with domain decomposition issues as part of the numerical simulation. This aspect of the PNS marching procedure is important since static mesh decompositions in applications of this type are not sufficient.

The parallel PNS code enables high performance analysis of aerodynamic flows to be conducted at unprecedented scales. High resolution solutions that have been obtained thus far reveal finely detailed flow structures which would otherwise be missed on coarser meshes. We are presently adapting the numerical procedure so that more complex geometries and starting conditions may be accommodated within the code. Additional work is also necessary to examine adaptive grid strategies in conjunction with the load balance requirements of distributed architectures.

As massively parallel computers become more prevalent, the types of analysis described herein will certainly become routine. Parallel interfaces to the application must be developed if serial bottlenecks are to be avoided. This is particularly true of CFD applications which traditionally require large amounts of solution input/output (I/O). Hopefully, machine designers and software developers will begin to bridge the gaps which exist on current generation machines to transition massively parallel computing into mainstream production environments.

Acknowledgments

The authors would like to thank Ray Tuminaro of Sandia National Laboratories for the contribution of the parallel version of the block cyclic reduction algorithm. In addition, they would also like to acknowledge Sandia Laboratories for the use of the *n*CUBE and iPSC as well as the Caltech MIMD Consortium for use of the Intel Delta in carrying out the performance timings in this paper.

References

[1]Beam, R. M., and Warming, R. F., "An Implicit Factored Scheme for the Compressible Navier–Stokes Equations," *AIAA Journal*, Vol. 16, 1978, pp. 393–401.

[2]Cline, D. D., Stagg, A. K., Shadid, J. N., and Carey, G. F., "A Parallel ADI Implementation for Navier–Stokes Applications," *Proceedings of the*

Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, April 9–16, 1992.

[3]Vigneron, Y. C., Rakich, J. V., and Tannehill, J. C., "Calculation of Supersonic Viscous Flow Over Delta Wings with Sharp Subsonic Leading Edges," AIAA, Washington, DC, 1978 (AIAA Paper 78-1137).

[4]Davis, R. T., Barnett, M., and Rakich, J. V., "The Calculation of Supersonic Viscous Flows Using the Parabolized Navier–Stokes Equations," *Computers and Fluids*, Vol. 14, No. 3, 1986, pp. 197–224.

[5]Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere, New York, 1984.

[6]Cline, D. D., "The Effect of Disparate Shock Eigenvalues on the Numerical Solution of Supersonic Conical Flows," Ph.D. Dissertation, Univ. of Texas, Austin, TX, 1987.

[7]Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D., *Solving Problems on Concurrent Processors*, Vol. 1, Prentice Hall, Englewood Cliffs, NJ, 1988.

[8]Cline, D. D., and Carey, G. F., "Symbolic Eigenvalue Analysis For Adaptive Stepsize Control in PNS Shock Stabilization," *Computers and Fluids*, Vol. 17, No. 4, 1989, pp. 527–535.

[9]Shadid, J. N., and Tuminaro, R. S., "A Comparison of Preconditioned Nonsymmetric Krylov Methods on a Large-Scale MIMD Machine," Sandia National Lab. Rept. SAND 91-0333, Sandia National Labs., Albuquerque, NM, 1991.

[10]Amdahl, G., "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities," *American Federation of Information Processing Societies Conference Proceedings*, Vol. 30, 1967, pp. 483–485.

[11]Freund, R. W., "A Transpose Free Quasi-Minimum Residual Method for Non-Hermitian Linear Systems," 1991 (RIACS Tech. Rept.).

[12]Nebbeling, C., and Bannink, W. J., "Experimental Investigation of the Supersonic Flow Past a Slender Cone at High Incidence," *Journal of Fluid Mechanics*, Vol. 87, Pt. 3, 1978, pp. 475–496.

[13]Degani, D., and Schiff, L. B., "Computation of Supersonic Viscous Flows around Pointed Bodies at Large Incidence," AIAA Paper 83-0034, 1983.