JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 33, No. 5, September–October 1996

Launch-Vehicle Simulations Using a Concurrent, Implicit Navier–Stokes Solver

Stephen Taylor* and Johnson C. T. Wang†
California Institute of Technology, Pasadena, California 91125

A large-scale multibody launch-vehicle simulation at supersonic speed is described. The simulation was conducted on a 256-node Intel Delta machine using a concurrent implementation of the Aerospace Launch System Implicit/Explicit Navier–Stokes code. This general Navier–Stokes solver has been used for a broad range of practical simulations. Those of interest involve a variety of nozzle flows and multibody launch-vehicle configurations such as the Titan IV. The code utilizes a finite volume total-variation-diminishing scheme for computing both steady and unsteady solutions to the three-dimensional compressible Navier–Stokes equations. The scheme is second-order accurate in space and is fully vectorized for operation on Cray computers. A line-by-line relaxation algorithm is used to accelerate the convergence for steady-state solutions. The code employs a variety of features that increase its practical utility. These include multibody configurations, turbulence modeling, and propellant-burning capabilities. The procedures for extending this code to distributed-memory parallel computers are described. Results of the application to a Titan IV launch vehicle at freestream Mach number 1.6 are discussed.

Nomenclature

E, F, G	= flux vectors
$\bar{E}, \bar{F}, \bar{G}$	= transformed flux vectors
e	= total energy per unit volume
$\hat{e}_x, \hat{e}_y, \hat{e}_z$	= unit vectors in the x, y, and z directions, respectively
J	= Jacobian of the transformation
k	= thermal conductivity
p	= pressure
$\boldsymbol{q}$	= heat transfer vector
T	= temperature
t	= time
U	= vector of conserved variables
$\bar{U}$	= transformed vector of conserved variables
u, v, w	= Cartesian velocity components
$\boldsymbol{V}$	= Cartesian velocity vector
x, y, z	= Cartesian coordinates
γ	= ratio of specific heats
Δ	= finite difference of a variable
λ	= bulk viscosity
μ	= viscosity
τ	= viscous stress tensor

Subscripts

i, j, k	= indices for cell center
ξ, η, ζ	= transformed coordinates

Superscripts

n	= index for time step
T	= transport of a matrix or vector

Introduction

THIS paper describes a concurrent implementation of the Aerospace Launch System Implicit/Explicit Navier–Stokes (ALSINS) code.[1] This general code has been used for a broad range of practical flow simulations, which include a variety of nozzle flows and multibody launch-vehicle configurations such as the one illustrated in Fig. 1. Notice that the computational grid is irregular in shape as well as in structure and the computational domain contains multiple bodies.

The basic governing equations utilized by the ALSINS code are the three-dimensional, compressible Navier–Stokes equations in conservation-law form. The code solves this set of equations in a transformed domain where the irregular computational domain and cells are mapped into a rectangular block with rectangular cells.

The numerical scheme uses a finite volume approach: The governing differential equations are discretized into a set of finite difference equations that equate the change of states at a cell center to the net sum of fluxes across the cell surfaces. This approach allows the three-dimensional operator of the Navier–Stokes equations to be reduced to the sum of three one-dimensional operations. The inviscid (convective) fluxes are evaluated using an extension of the total-variation-diminishing (TVD) algorithm of Harten.[2] The viscous fluxes are computed using a standard central difference scheme. The numerical solutions are second-order in space. For unsteady-flow problems, the time derivatives representing the change of states at the cell center are discretized using a first-order-accurate explicit Euler scheme.

For the steady-flow problems, the solution is obtained by using an implicit algorithm that allows Courant–Friedrichs–Lewy numbers

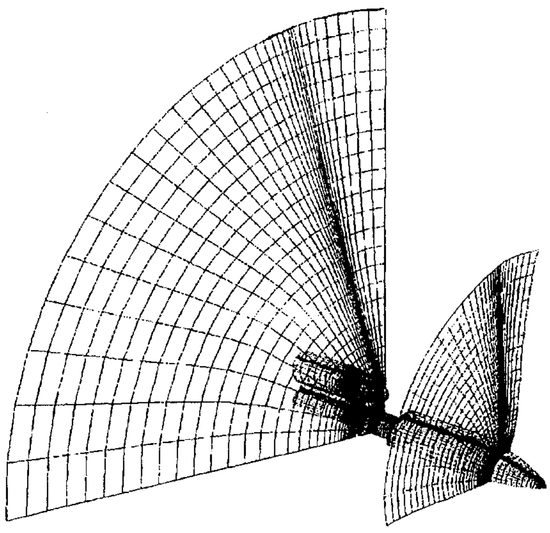
Fig. 1 Multibody launch-vehicle simulation.

Received March 28, 1995; revision received Jan. 30, 1996; accepted for publication Feb. 9, 1996. Copyright © 1996 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Assistant Professor, Department of Computer Science, Scalable Concurrent Programming Laboratory, MS 256-80. Member AIAA.

†Visiting Associate, Department of Computer Science, Scalable Concurrent Programming Laboratory, MS 256-80.

on the order of hundreds.[1] Using this implicit algorithm, the governing equations are cast into a block-tridiagonal system of equations with the changes of state as dependent variables and the net sums of fluxes as nonhomogeneous parts. A full set of equations is given in Ref. 1. In this paper we extract only those parts of the equations that are necessary to explain the method of concurrent execution.

The ALSINS code is unusual in that it allows complex irregular calculations to be conducted in a manner that is highly vectorizable. Thus, the code executes efficiently on Cray-style computers and provides a reliable vehicle for performance comparisons between parallel machines and vector supercomputers. This paper concerns experiments that utilize the code on scalable multicomputers. We present results for large-scale simulations of the Titan IV launch vehicle shown in Fig. 1. This application utilizes all of the concepts described in this paper, including inviscid and viscous solutions, implicit techniques, irregular grid decompositions, and multiple bodies.

Basic Equations

The basic governing equations to be solved are the conservation-law form of the three-dimensional Navier–Stokes equations in the Cartesian coordinate system:

$$\frac{\partial U}{\partial t} + \nabla \cdot \boldsymbol{F} = 0 \tag{1}$$

$$U = [\rho \quad \rho u \quad \rho v \quad \rho w \quad e]^T \tag{2}$$

$$\boldsymbol{F} = E\hat{\boldsymbol{e}}_x + F\hat{\boldsymbol{e}}_y + G\hat{\boldsymbol{e}}_z \tag{3}$$

$$E = \begin{bmatrix} \rho u \\ \rho u^2 + p + \tau_{xx} \\ \rho uv + \tau_{xy} \\ \rho uw + \tau_{xz} \\ (e + p + \tau_{xx})u + \tau_{xy}v + \tau_{xz}w + q_x \end{bmatrix} \tag{4}$$

$$F = \begin{bmatrix} \rho v \\ \rho uv + \tau_{xy} \\ \rho v^2 + p + \tau_{yy} \\ \rho vw + \tau_{yz} \\ \tau_{yx}v + (e + p + \tau_{yy})v + \tau_{yz}w + q_y \end{bmatrix} \tag{5}$$

$$G = \begin{bmatrix} \rho w \\ \rho uw + \tau_{zx} \\ \rho vw + \tau_{zy} \\ \rho w^2 + p + \tau_{zz} \\ \tau_{zx}u + \tau_{zy}v + (e + p + \tau_{zz})w + q_z \end{bmatrix} \tag{6}$$

$$\tau_{xx} = -2\mu\frac{\partial u}{\partial x} - \lambda\nabla\cdot\boldsymbol{V}, \qquad \tau_{yy} = -2\mu\frac{\partial v}{\partial y} - \lambda\nabla\cdot\boldsymbol{V}$$

$$\tau_{zz} = -2\mu\frac{\partial w}{\partial z} - \lambda\nabla\cdot\boldsymbol{V}, \qquad \tau_{xz} = \tau_{zx} = -\mu\left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right)$$

$$\tau_{yx} = \tau_{xy} = -\mu\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) \tag{7}$$

$$\tau_{yz} = \tau_{zy} = -\mu\left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial y}\right)$$

$$\boldsymbol{q} = q_x\hat{\boldsymbol{e}}_x + q_y\hat{\boldsymbol{e}}_y + q_z\hat{\boldsymbol{e}}_z = -k\nabla T$$

For a polytropic gas, the energy e is related to the pressure p by the equation of state $p = (\gamma - 1)[e - \rho(u^2 + v^2 + w^2)/2]$. The viscosity and thermal conductivity are decomposed into molecular and turbulent components as $\mu = \mu^M + \mu^T$ and $k = k^M + k^T$, respectively. Here μ^T is evaluated using Baldwin–Lomax turbulence model.[3]

Governing Equations in the Transformed Domain

We apply to the basic equations a general coordinate transformation of the form

$$\xi = \xi(x, y, z), \qquad \eta = \eta(x, y, z), \qquad \zeta = \zeta(x, y, z) \tag{8}$$

Thus, Eq. (1) can be written in the following form:

$$\frac{\partial \bar{U}}{\partial t} + \frac{\partial \bar{E}}{\partial \xi} + \frac{\partial \bar{F}}{\partial \eta} + \frac{\partial \bar{G}}{\partial \zeta} = 0 \tag{9}$$

where

$$\bar{U} = U/J, \qquad \bar{E}J = E\xi_x + F\xi_y + G\xi_z$$

$$\bar{F}J = E\eta_x + F\eta_y + G\eta_z, \qquad \bar{G}J = E\zeta_x + F\zeta_y + G\zeta_z$$

In Eq. (9), J is the Jacobian of the transformation:

$$J = \frac{\partial(\xi, \eta, \zeta)}{\partial(x, y, z)} \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \tag{10}$$

Equation (8) transforms an irregular computational domain, such as that shown in Fig. 1, into a rectangular one, and irregular cells into regular cells. Equation (9) reduces to Eq. (1) if $\xi = x$, $\eta = y$, and $\zeta = z$. The appeal of this transformation is that irregular computational domains yield regular domains that can be calculated with a high degree of vectorization on Cray-style architectures.

Explicit Difference Equations

Carrying out a finite volume integration over the cell and using the first-order Euler differencing for the time-derivative term, the difference form of Eq. (9) becomes

$$\bar{U}^{n+1}_{i,j,k} = \bar{U}^{n}_{i,j,k} - \Delta t \frac{J_{i,j,k}}{\Delta\xi\Delta\eta\Delta\zeta}\Bigg[\left(\bar{E}_{i+\frac{1}{2},j,k} - \bar{E}_{i-\frac{1}{2},j,k}\right)\Delta\eta\Delta\zeta$$
$$+ \left(\bar{F}_{i,j+\frac{1}{2},k} - \bar{F}_{i,j-\frac{1}{2},k}\right)\Delta\xi\,\Delta\zeta$$
$$+ \left(\bar{G}_{i,j,k+\frac{1}{2}} - \bar{G}_{i,j,k-\frac{1}{2}}\Delta\xi\Delta\eta\right)\Bigg] \tag{11}$$

The terms inside the bracket, representing the fluxes across the cell surfaces, are depicted in Fig. 2.

There are two parts, inviscid and viscous, in each surface flux term. We will use superscript inv to indicate the inviscid part, and vis the viscous part. For example,

$$\bar{E}_{i+\frac{1}{2},j,k} = \bar{E}^{\text{inv}}_{i+\frac{1}{2},j,k} + \bar{E}^{\text{vis}}_{i+\frac{1}{2},j,k} \tag{12}$$

Details for computing the surface fluxes are given in Refs. 4–7. For the purposes of this paper, it is sufficient to point out that the computation of the inviscid part of $\bar{E}$ requires information concerning

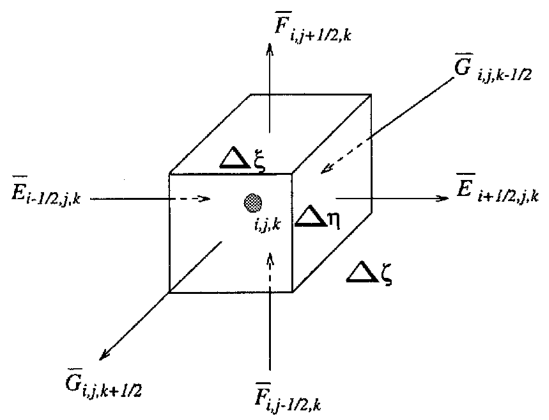

Fig. 2 Flux terms for finite volume calculation.

$\bar{U}^n$ at positions i, $i \pm 1$, and $i \pm 2$ for second-order spatial accuracy, i.e.,

$$\bar{E}^{\text{inv}}_{i+\frac{1}{2},j,k} = \bar{E}^{\text{inv}}\left(\bar{U}^n_{i-1,j,k}, \bar{U}^n_{i,j,k}, \bar{U}^n_{i+1,j,k}, \bar{U}^n_{i+2,j,k}\right) \tag{13}$$

and

$$\bar{E}^{\text{inv}}_{i-\frac{1}{2},j,k} = \bar{E}^{\text{inv}}\left(\bar{U}^n_{i-2,j,k}, \bar{U}^n_{i-1,j,k}, \bar{U}^n_{i,j,k}, \bar{U}^n_{i+1,j,k}\right) \tag{14}$$

Note that in Eqs. (13) and (14), the values of j and k are kept the same. The central difference scheme is used to compute the viscous terms, therefore:

$$\bar{E}^{\text{vis}}_{i+\frac{1}{2},j,k} = \bar{E}^{\text{vis}}\left(\bar{U}^n_{i+1,j\pm 1,k}, \bar{U}^n_{i+1,j,k\pm 1}, \bar{U}^n_{i,j\pm 1,k}, \bar{U}^n_{i,j,k\pm 1}, \bar{U}^n_{i\pm 1,j,k}\right) \tag{15}$$

Unlike Eq. (13), for a given j and k, because of the cross derivatives for the viscous terms, $\bar{E}^{\text{vis}}$ depends also on $\bar{U}^n$ at $j \pm 1$ and $k \pm 1$. For example, see the definition of τ_{xz} in Eq. (7).

Implicit Difference Equations

To accelerate the convergence for steady-state problems, a numerical technique was presented using a line relaxation procedure. The discretized equation in the ith direction is a block tridiagonal system:

$$\tilde{A}^- \Delta U^n_{i-1,j,k} + \tilde{D}\, \Delta U^n_{i,j,k} + \tilde{A}^+ \Delta U^n_{i+1,j,k} = \text{RHS} \tag{16}$$

where $\tilde{A}^-$, $\tilde{D}$, and $\tilde{A}^+$ are preconditioned 5×5 matrices. The unknowns ΔU^n are defined as

$$\Delta U^n_{i,j,k} = U^{n+1}_{i,j,k} - U^n_{i,j,k} \tag{17}$$

In Eq. (16), RHS is the second term of the right-hand side in Eq. (11). For a steady-state solution $\Delta U^n = 0$ by definition. When this condition is reached, from Eq. (16) it follows that RHS $= 0$. This represents a solution to the steady-state Navier–Stokes equations. Figure 2 shows a computational cell of size $\Delta\xi\, \Delta\eta\, \Delta\zeta$ with cell center at ξ, η, ζ and the surface fluxes.

Simulation Results

The concurrent solution methods described here have been validated in Ref. 8 using standard shock-tube results and a complex shock interaction problem involving steady-state effects from a supersonic flow (Mach 3.17) over two inclined faces.[9,10] Here, we show the results of using the same code for a complex multibody simulation of the Titan IV launch vehicle at Mach 1.6. A similar calculation with much coarser grid has been presented in Ref. 11, using a Cray Y-MP.

The Titan IV launch vehicle exhibits acoustic buffeting due to aerodynamic effects centered near to the main guidance systems and payload fairing. These effects can degrade the guidance systems and can be detrimental to delicate payloads. Accurate flowfield simulations of the effects have been produced at Mach 1.6 with zero angle of attack. These results have been used to understand and resolve a variety of design issues. Prediction of the effects near the transonic region and at nonzero angles of attack will further enhance our understanding of the aerodynamic phenomena involved in complex launch-vehicle configurations.

Figure 3 shows a cross section through the three-dimensional pressure field computed around the Titan IV vehicle; these results are based on a freestream Mach number of 1.6 (note that all results are nondimensionalized). The results were obtained by running the Concurrent-ALSINS code on the Intel Delta Machine using 256 computing nodes. Figure 4 shows the calculated surface pressure on the centerline of the vehicle compared with experimental wind-tunnel data. Notice the close correspondence between experimental and computed values, indicating that the flowfield provides an accurate characterization of the behavior of the vehicle in the neighborhood of the measured results. We believe the one point that does not agree arises from a cable in the wind tunnel that connects the core vehicle to the booster. The vehicle's acoustic buffeting arises from a recirculation region immediately above the main guidance system and adjacent to the payload fairing. Figure 5 shows the vector velocity field in this region and highlights the presence of the recirculation.

In addition to determining buffeting characteristics, these flowfield simulations have a variety of other practical uses. From a calculated pressure field, it is possible to determine the forces acting on the vehicle and thereby predict its stability in different regimes of flight. The results can be used to predict the aerodynamic drag of the vehicle, which is useful to engineers in determining load characteristics. By extending the calculated bow shock structure, it is possible to determine the strength of the sonic boom created by the vehicle. Calculated temperature contours are used for thermal protection considerations.

The ALSINS code has previously been used for a wide variety of nontrivial simulations on Cray-style architectures. In summary, these results are as follows:

1) *Cray X-MP.* On the available Cray X-MP, the largest simulation we were able to execute was a vectorized single-body launch-vehicle calculation as presented in Ref. 11. This involved a grid of dimensions $79 \times 56 \times 10$ (44,240 grid points). The limiting factor was the available memory; this computation used 5.03 Mwords (113 words/cell). The performance for this simulation was 0.68×10^{-4} (s/cell)/iteration.

2) *Cray II.* The constraints imposed by the X-MP forced us to transfer the code to an available Cray II architecture. Using this machine, the largest simulation we have been able to execute is a complex multibody launch-vehicle configuration[11]; the grid for this simulation is shown in Fig. 1. This grid has dimensions $159 \times 86 \times 62$ (847, 788 grid points), uses 65 Mwords (77 words/cell), and executes at 0.78×10^{-4} (s/cell)/iteration.

3) *Intel Delta.* The driving force behind our consideration of parallel architectures is the memory and scheduling constraints imposed by the available Cray machines. On the double-wedge problem presented in Ref. 8, a 145-node Intel Delta machine, running at approximately 70% utilization, obtained a performance of 0.28×10^{-4} (s/cell)/iteration. The computational grid had dimensions $72 \times 108 \times 108$ (839,808 grid points) and used 100 Mwords of memory (128 words/cell). The Titan IV results took approximately 120 h of machine time on a 256-node machine, at approximately 60% utilization. Although there are numerous optimizations that we could perform, we consider these results acceptable and are more interested in solving larger problems. The results indicate that it is reasonable to expect that during the next two years we will be able to execute problems at least 16 times larger and 35 times faster than our largest previous simulation, using a 2000-node Intel Paragon machine currently under installation.

Unfortunately, because of the cost of these large-scale simulations, it has not yet been possible to perform a side-by-side comparison of convergence on the Cray and Intel machines. We expect there to be some difference due to the segmentation of the line-relaxation technique involved in the parallel algorithms. However, we have not perceived a substantial difference in the convergence that would cause us concern; we suspect this is due to the pipelining effect inherent in the algorithms.

Domain Decomposition

The basic numerical scheme outlined in previous sections is implemented on parallel machines using the technique of domain decomposition. This technique involves dividing the data structures of the program and operating on the parts independently.

To cope with complex geometries it is necessary to decompose the irregular grids into blocks of varying sizes. However, a completely arbitrary decomposition is not needed. For a wide variety of practical launch-vehicle configurations it is sufficient to decompose the domain along the axes. Figure 6 shows an example of this type of decomposition. An irregular grid, such as that shown in Fig. 1, is first transformed into a regular computational grid through the transformation previously described. The computational grid is then decomposed into blocks of varying sizes along each axis. For

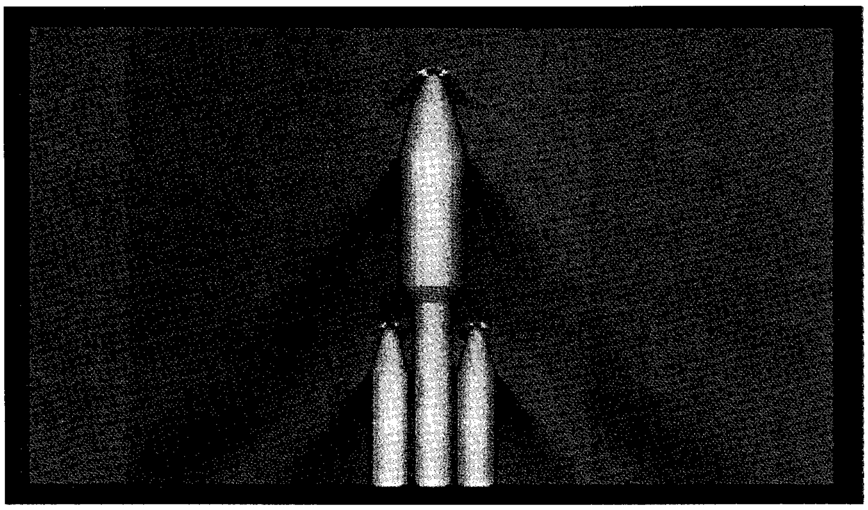

Fig. 3 Simulated pressure contours.

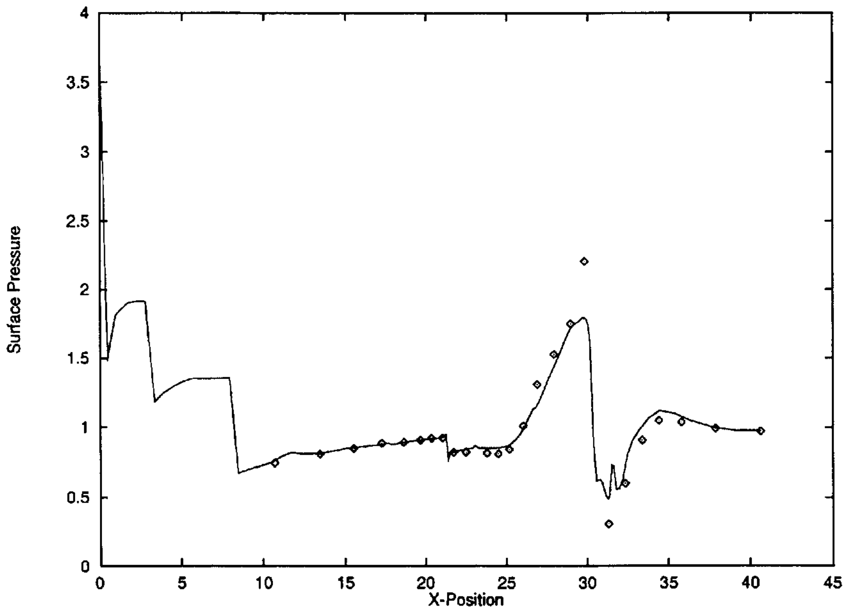

Fig. 4 Predicted surface pressure in comparison with wind-tunnel data: ——, calculated surface pressure and ◇, experimental results.

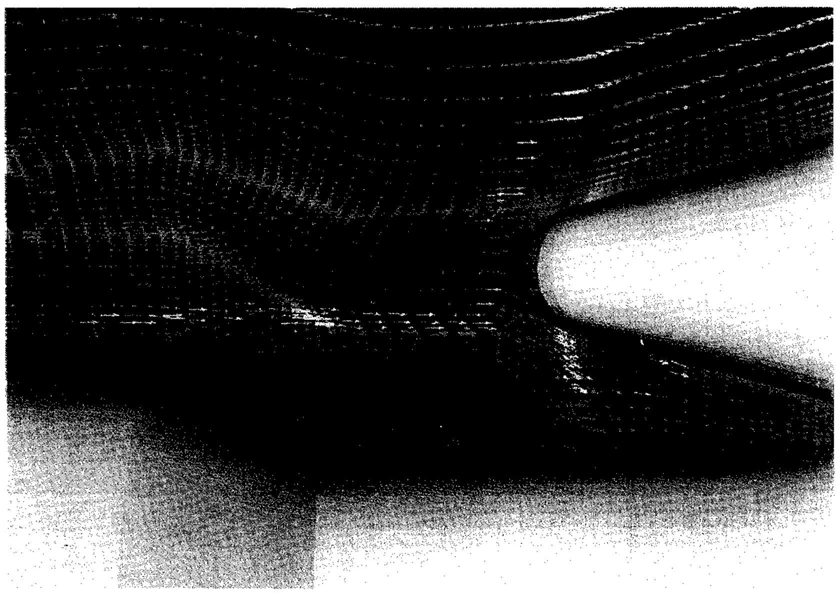

Fig. 5 Simulated velocity field in proximity to SRM booster.

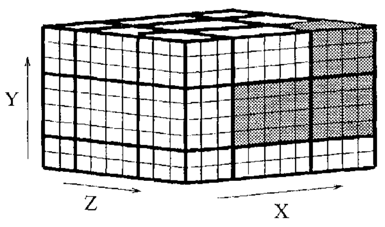

Fig. 6 Irregular domain decomposition.

example, in Fig. 6, decomposition along the X axis yields blocks with X dimensions containing 3, 5, and 4 cells; decomposition of the Y and Z axes yields blocks with dimensions containing 2, 4, and 3 cells.

In generating the decomposition, it is necessary to specify the boundary conditions for each block independently. In addition to the standard solid-wall and freestream boundary conditions, we add a further condition that specifies a face resulting from a cut through the domain.

This decomposition is capable of modeling a variety of single-body geometries, but is not sufficiently flexible to model multibody launch vehicles such as that illustrated in Fig. 1. To generalize the basic decomposition strategy, we consider some blocks in the decomposition as holes. These holes are surrounded by solid boundaries and thus represent multibody configurations. Figure 6 uses shaded areas to signify the location of holes, representing multiple bodies, in the decomposition.

Concurrent Algorithm

Given the decomposition previously outlined, it is straightforward to build an iterative concurrent algorithm. Equations (13) and (14) are used to compute the inviscid flux across a cell surface. The second-order-accurate numerical scheme requires two cells of information from other blocks at time $t - 1$. Thus to compute a block in the decomposition, it is necessary to communicate neighboring faces from adjacent blocks in the decomposition. Unfortunately, this communication scheme is not sufficient to calculate the viscous effects as indicated in Eq. (15). Recall that this is due to the cross derivative terms of the viscous shear. Thus each block of the decomposition needs to have available the 12 neighboring corners from adjacent blocks. The shaded regions in Fig. 7 illustrate the types of communication structure necessary to compute all cells in a single block at time t based on values at time $t - 1$.

Implicit Solutions

Although this basic communication structure is sufficient for the explicit simulations, it requires some changes to the numerical scheme to handle implicit calculations. Recall that a line-by-line relaxation scheme is used for this calculation. Unfortunately, this scheme is a sequential algorithm that iterates through an entire dimension of the domain; it cannot, therefore, be executed concurrently within each block. However, in the steady state, the change in dependent variables from one time step to another will be zero:

$$\Delta U^n_{i,j,k} = U^{n+1}_{i,j,k} - U^n_{i,j,k} = 0$$

We take advantage of this fact by setting the required ΔU^n from an adjacent block equal to zero. From previous experience in using the sequential code, we know that this approach does not cause instability and even speeds up convergence.

Model for Time

Recall from Eq. (11) that the value of Δt is bounded by the Courant number. To find the next Δt at step t, it is necessary to combine information from all blocks in the decomposition. Although for steady-state problems the use of a local Δt has been suggested in the literature, our experience is that this yields results that compare poorly with experimental data. Therefore, we use a uniform Δt even for implicit calculations. It is calculated by finding a local minimum Δt for all blocks independently and then combining these values to provide the minimum value over the entire domain.

Table 1 Basic iterative algorithm

```
block(...)
{  load geometry data into block
   calculate local $\Delta t$ and send to minimum calculation
   while (time not exhausted) {
      receive global $\Delta t$
      extract 6 faces and 12 corners from block and send to neighbors
      receive 6 faces and 12 corners
      compute dependent variables at $t + \Delta t$ using
      faces, corners, and $\Delta t$
      calculate new local $\Delta t$ and send to minimum calculation
   }
}

minimum(...)
{  receive a local $\Delta t$ from each block
   calculate the minimum and broadcast minimum to all blocks
}
```

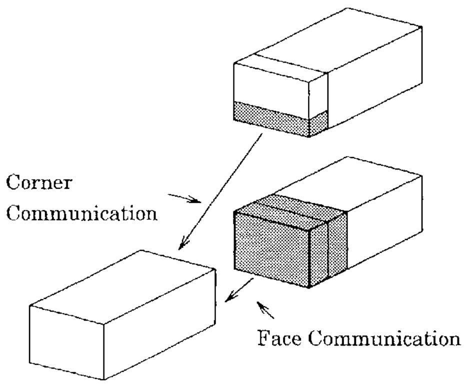

Fig. 7 Communication structure.

Turbulence Modeling

To complete the basic algorithm we require a method for utilizing turbulence models on parallel machines. For simplicity we achieve this by requiring that blocks next to a solid boundary be thick enough to contain the turbulent boundary layer inside a single block. This allows all turbulent effects to be calculated without communication or adding sequential operations to the concurrent algorithm.

Single-Body Algorithm

To simplify the implementation, we represent each block in the domain as a concurrent process and associate with it communication channels to eighteen neighbors; these channels connect a block to the six neighbors required for face exchanges and twelve neighbors used for corner exchanges. End-around connections are used at the boundaries to ensure that every block has a full complement of neighbors. It is trivial to establish the appropriate channels using the techniques described in Ref. 12. In outline, the basic algorithm may now be expressed as shown abstractly in Table 1.

A block process can be executed at any computer in the machine. Each block loads the appropriate geometry information from a unique file concurrently. In this manner a block is informed of its boundary conditions. Notice that the algorithm is self-synchronizing by virtue of the explicit number of face and corner exchanges required. In the cases where the boundary conditions signify values communicated are unimportant, we send only a single number and discard it at the receiver. The load and compute steps in the algorithm are simply existing Fortran code from the original ALSINS code. This code is modified to deal with a single additional boundary condition: that of a face in the domain.

Multibody Algorithm

The scheme is adapted to deal with multibody simulations by allowing any block to represent a hole in the domain. A block is informed that it represents a hole when the geometry information is loaded. If a block represents a hole, the compute function does nothing, the extract function returns a vector of length l, and the calculate function returns infinity.

Load Balancing

The computation will be reasonably well balanced if the blocks are similar in size. For the most part, our blocks are sufficiently large that a simple mapping technique is sufficient. Each block is numbered to identify it uniquely, and mapping is achieved by mapping the ith block to the ith computer, for the most part placing contiguous blocks close together. Most of our current experiments have been based on this simple technique.

Further Concurrent Extension

The research described here employs a number of constraints to simplify the initial implementation and allow existing launch-vehicle grids to be utilized. A single computational domain with holes is employed for multibody problems. The domain was decomposed only along the domain axes. Multiple bodies were represented by holes in the decomposition. This allowed a trivial communication and synchronization structure to be used in which every block (including those representing holes) simply exchanged values with neighbors at each iteration.

This approach has a number of shortcomings that have become apparent in our work on the more complex nine-booster Delta II launch vehicle. For more complex geometries it is difficult to obtain a single computational domain when designing the grid. If this can be achieved, it results in a complex and wasteful gridding structure. This occurs because of the fine grid required to characterize boundary layers. When a single domain is used, the boundary-layer gridding must be extended into the outer reaches of the grid. This causes the resulting grids to utilize a high density of grid points in areas that might otherwise require only a few isolated points. A corresponding waste of storage and computing resources is caused when this type of grid structure is used by the solver.

To solve these problems we have developed a multiblock solver in which the block interfaces and the grid points on the interface of the basic domain decomposition need not be matched.[13] An example of the gridding structure used by this solver is shown in Fig. 8 for the double-wedge problem. Notice how the boundaries at each step along the length of the wedge do not match.

This solver substantially simplifies grid generation. It allows isolated regions in the flowfield that require a refined mesh to capture the physics of the flow, without incurring overheads where a large mesh is sufficient. The entire computational domain is divided into blocks of different size with arbitrary topological arrangement. However, conceptually, we view the blocks as overlapped. At the block interfaces each block has at least two cells that protrude into the adjacent blocks. These cells we term fictitious cells, and they serve to characterize the block interface. The cells are used to construct the fluxes at the interface and preserve the second-order accuracy and TVD properties of our algorithms.

The use of an interpolation scheme has a number of implications for the concurrent algorithms. Each block in the domain may now communicate an arbitrary number of messages, of varying size, at each time step. However, all of the necessary information to construct the messages and predict their destination can be obtained statically during grid generation. To simplify the task of using the algorithms on different geometries, we have developed a table-driven communication and mapping structure using the concurrent graph abstraction developed within the group. Tables are constructed in files during grid generation, and this generic library is used to distribute the computation on machines of differing size and architecture.

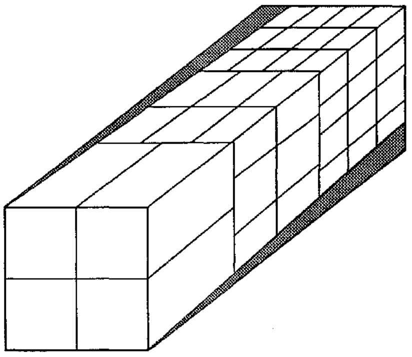
Fig. 8 Mismatched grid structure.

Conclusion

This paper gives a status report on the development and application of an industrial-strength Navier–Stokes code for concurrent supercomputers. The code has been carefully validated using symmetric three-dimensional shock interaction problems for which there are experimental data, analytical results, and computations using non-TVD algorithms. Large-scale calculations are shown here that exercise various options within the code, including turbulence modeling, irregular decompositions, multibody configurations, and implicit techniques. The efficiency of the concurrent calculations is discussed. The concurrent formulation utilizes domain-decomposition techniques that allow the irregular computational grid to be decomposed into regular blocks of varying sizes. This is achieved by decomposing each axis independently. Multibody configurations are modeled through holes in the decomposition that serve simply to synchronize the concurrent algorithm. A further improvement in the capability of the concurrent code that allows nodally mismatched grids has been described.

Acknowledgment

This research is sponsored by the Advanced Research Projects Agency, ARPA Order 8176, monitored by the U.S. Office of Naval Research under Contract N00014-91-J-1986.

References

[1]Wang, J. C. T., and Widhopf, G. F., "An Efficient Finite Volume TVD Scheme for Steady State Solutions of the 3-D Compressible Euler/Navier–Stokes Equations," AIAA Paper 90-1523, June 1990.

[2]Harten, A., "High Resolution Schemes for Hyperbolic Conservation Laws," *Journal of Computational Physics*, Vol. 49, 1983, pp. 357–393.

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

[4]Wang, J. C. T., Widhopf, G. F., and Chen, S. H., "Three Dimensional Finite Volume TVD Scheme for Geometrically Complex Steady State and Transient Flows," AIAA Paper 88-0288, Jan. 1988.

[5]Wang, J. C. T., and Widhopf, G. F., "A High Resolution TVD Finite Volume Scheme for Euler Equations in Conservation Form," *Journal of Computational Physics*, Vol. 84, No. 1, 1989, pp. 145–173.

[6]Wang, J. C. T., and Widhopf, G. F., "Numerical Simulation of Blast Flowfields Using a High Resolution TVD Finite Volume Scheme," *International Journal of Computers and Fluids*, Vol. 18, No. 1, 1990, pp. 103–137.

[7]Widhopf, G. F., and Wang, J. C. T., "A TVD Finite Volume Technique for Nonequilibrium Chemically Reacting Flows," AIAA Paper 88-2711, 1988.

[8]Wang, J. C. T., and Taylor, S., "A Concurrent Navier–Stokes Solver for Implicit Multibody Calculations," *Parallel CFD93* (Paris), edited by A. Ecer, J. Hauser, P. Leca, and J. Periaux, Elsevier, New York, 1995, pp. 295–307.

[9]Shanka, V., Anderson, D., and Kutler, P., "Numerical Solutions for Supersonic Corner Flow," *Journal of Computational Physics*, Vol. 17, No. 2, 1975, pp. 160–180.

[10]Charwat, A. F., and Redekeopp, L. G., "Supersonic Interference Flow Along the Corner of Intersecting Wedges," *AIAA Journal*, Vol. 5, No. 3, 1967, pp. 480–488.

[11]Wang, J. C. T., Than, P. T., and Widhopf, G. F., "Multi-Body Launch Vehicle Flowfield Simulation," AIAA Paper 91-0072, Jan. 1991.

[12]Chandy, K. M., and Taylor, S., *An Introduction to Parallel Programming*, Jones and Bartlett, Boston, 1992.

[13]Wang, J. C. T., and Taylor, S., "A Concurrent, Nodal Mismatched, Implicit Navier–Stokes Solver," *Parallel CFD94* (Kyoto), edited by N. Satofuka, J. Periaux, and A. Ecer, Elsevier, New York, 1995, pp. 393–398.

J. C. Adams
Associate Editor

Color reproduction courtesy of Advanced Research Projects Agency