[7]Gatski, T. B., and Speziale, C. G., "On Explicit Algebraic Stress Models for Complex Turbulent Flows," *Journal of Fluid Mechanics*, Vol. 254, 1993, pp. 59–78.

[8]Abid, R., Morrison, J. H., Gatski, T. B., and Speziale, C. G., "Prediction of Complex Aerodynamic Flows with Explicit Algebraic Stress Models," AIAA Paper 96-0565, Jan. 1996.

[9]Hinze, J. O., *Turbulence*, McGraw–Hill, New York, 1975, Chap. 1.

[10]Speziale, C. G., Sarkar, S., and Gatski, T. B., "Modeling the Pressure-Strain Correlation of Turbulence: An Invariant Dynamical Systems Approach," *Journal of Fluid Mechanics*, Vol. 227, 1991, pp. 245–272.

[11]Speziale, C. G., and Xu, X.-H., "Towards the Development of Second-Order Closure Models for Non-Equilibrium Turbulent Flows," *Proceedings of the 10th Symposium on Turbulent Shear Flows*, Vol. 3, Pennsylvania State Univ., State College, PA, 1995, pp. 23.7–23.12.

[12]Rogers, M. M., Moin, P., and Reynolds, W. C., "The Structure and Modeling of the Hydrodynamic and Passive Scalar Fields in Homogeneous Turbulent Shear Flow," Stanford Univ., TR TF-25, Stanford, CA, Aug. 1986.

Numerical Simulation of Viscous Flow over Rotors Using a Distributed Computing Strategy

Ashok Bangalore,* Ralph L. Latham,† and Lakshmi N. Sankar‡
Georgia Institute of Technology, Atlanta, Georgia 30332-0150

Introduction

SCIENTIFIC computing using parallel computers has been a topic of active research for the past several years. Several parallel supercomputers are being used today, such as the Intel Paragon, Connection Machine CM-5, MASPAR, KSR, and Cray Y/MP systems. The recent versions are multiple instruction multiple data machines, and they differ from each other in the number of processors used (16–64,000), the computational power of the individual processors, and the strategy for passing messages and information between processors. In general, the computer codes must be tailored to individual architecture for maximum performance, although common machine-independent programming practices using Fortran-90 and High Performance Fortran are evolving. The parallel supercomputers are expensive and usually are used heavily. Typical turnaround times for a computational fluid dynamics (CFD) application may vary from one day to several days. Turnaround times of this order and the high equipment cost of these machines make them impractical for small research labs, universities, and many aerospace firms.

A new, inexpensive way of parallel computing, using a network of heterogeneous computers known as distributed computing, is available.[1] Many of these workstations are idle after work hours, because each of them is not individually big enough to solve large-scale problems. Distributed computing allows this idle time to be used effectively. For the distributed computing to be effective, standard, portable, efficient methods for interprocessor communications are needed. In the present study, interprocessor communications was achieved through a software called Parallel Virtual Machine (PVM), which is being developed by the joint efforts of Oak Ridge National Laboratory and Emory University.[2]

Many researchers have begun to study the use of PVM in distributed CFD applications. For example, Smith and Pallis[3] have implemented the PVM software to solve the Navier–Stokes equations using MEDUSA, an overset grid flow solver on a cluster of heterogeneous workstations at NASA Ames Research Center. They have reported significant gain in computation time by using the PVM software. They also point out the economics of using workstations as opposed to using supercomputers. More recently, Weed and Sankar[4] ported a multiblock Euler/Navier–Stokes code (TEAM) to a distributed network and reported significant gain in computation time when solving large-scale problems.

PVM Implementation

The basic idea in implementing PVM software in the existing Navier–Stokes solver[5,6] is to divide the computational domain into smaller blocks and solve each block of grid points on a different processor. The original Navier–Stokes solver is an unsteady, three-dimensional finite volume-based code and has been extensively documented in Ref. 5. In the present application, the computational grid is a C-H grid around the fixed wing or rotor blade with 121 points in the streamwise direction, 28 points in the spanwise direction, and 41 points in the normal direction. The complete domain is divided into two or three spanwise zones, depending on the number of processors used. Splitting the domain in the spanwise direction is efficient and convenient because the spanwise direction is treated explicitly. Figure 1 shows schematically the division of spanwise zones. Using the PVM message-passing routines, the flow data are sent from the slave processes to the master process at each time step. Also, updated interface data are sent to the slaves from the master process using the PVM routines.

Results and Discussion

Speedup and Performance

The virtual machine network used in the present work consisted of three Hewlett-Packard (HP) Apollo workstations: two model 720 and one model 730. The benchmark and performance monitoring for the present Navier–Stokes code using the PVM software was done on this network with no other user processes running on the machines. The workload on the machines was near zero and this gave a realistic figure of speedup obtained by using the PVM software.

The CPU and the elapsed time during each iteration are obtained using the time () function call provided in the Fortran library. The time () function gives the elapsed time rounded to the nearest second. This level of accuracy was adequate for the present application because the computation time required per time step was of the order of 10–20 s. The amount of message passing between the master and the slave processes was about 452 kB of flow data per iteration. The actual communication time taken for this particular packet of data was less than 0.1 s on the present ethernet network. This indicated that the latency of the network was a very small percentage of the total wait time of the processes.

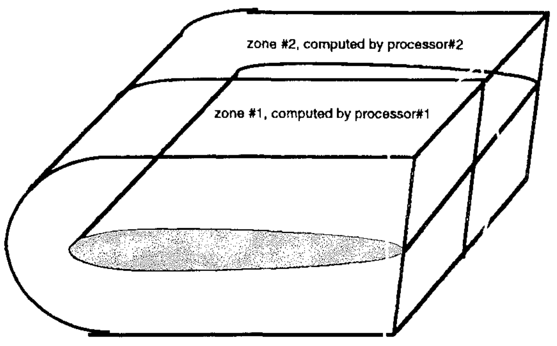
zone #2, computed by processor#2
zone #1, computed by processor#1

Fig. 1 Spanwise division of the domain.

Received Nov. 18, 1994; presented as Paper 95-0575 at the AIAA 33rd Aerospace Sciences Meeting, Reno, NV, Jan. 9–12, 1995; revision received Nov. 15, 1995; accepted for publication Jan. 12, 1996. Copyright © 1996 by the authors. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

*Graduate Research Assistant, School of Aerospace Engineering; currently Postdoctoral Researcher, 233 Hammond Building, Pennsylvania State University, University Park, PA 16802.

†Research Engineer, School of Aerospace Engineering.

‡Professor, School of Aerospace Engineering. Senior Member AIAA.

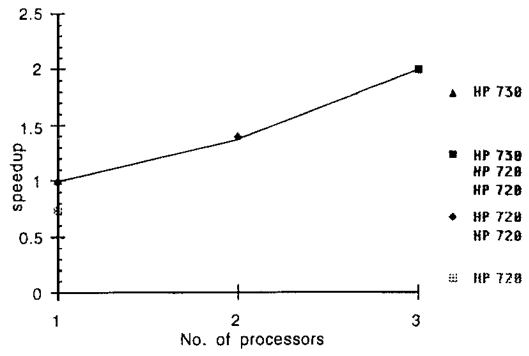

Fig. 2 Speedup relative to the number of processors.

Excellent computation times were achieved when two HP workstations were used to run the PVM Navier–Stokes code. The domain was equally divided among the two processors and near-zero wait times were seen on both the processors. By using all three HP workstations, a better performance was obtained than with two machines but the maximum wait time on the faster processor was of the order of 2–3 s per iteration. Figure 2 shows the variation of speedup with the number of processors.

Load Balancing

One of the primary factors affecting the parallel efficiency of computation on a multiprocessor machine or a network-based parallel computer is load balancing, i.e., minimizing the idle time of each processor. Static and dynamic load balancing schemes were implemented in the present study and are discussed in the following section.

Static Load Balancing

Static balancing is simple and was done at the beginning of the task. An ad hoc approach was used to determine the idle time of each processor at the beginning of the process. The wait time on each processor was obtained initially by running a few iterations interactively on the system. This provided a fairly good idea of the load imbalance at that particular time. The blocks were reallocated, and the workload was transferred from the processor with the least wait time to the processor with the highest wait time to achieve a minimum wait time in all of the processors. This data distribution was done only once, at the beginning of the task, and was frozen during the subsequent calculations. This approach works well when the workload on the machines does not vary significantly during the simulation.

Dynamic Load Balancing

The purpose of a dynamic load balancing algorithm is to minimize dynamically the wait time (idle time) of each worker (slave) in a network environment during the period of computation. In a multiuser network environment, the load of each processor varies because of the addition or deletion of user and/or system processes. Therefore, it is very desirable to redistribute the workload of each worker, depending on the load of the processor.

In the present study, the master program periodically collected the average wait times from each slave processor. The computational domain was redistributed among the slaves by comparing the average wait times of the slaves. The workload of the slave with the highest wait time was increased by adding more grid points to be computed, and the workload on the slave with the least wait time was reduced by taking out the corresponding grid points from its computational domain. Also, grid points were redistributed only when the difference between the highest and the lowest wait times was above or below certain tolerance value, as defined by the user. This was done to reduce the overhead time incurred in the redistribution process. Two types of loading scenarios are discussed here for comparison of wait times between static and dynamic load balancing schemes. First, a lightly loaded virtual machine network, where all of the workstations are dedicated to the particular computation, is discussed. Later, a heavily loaded system, where the loads on machines vary considerably during the computation, is described.

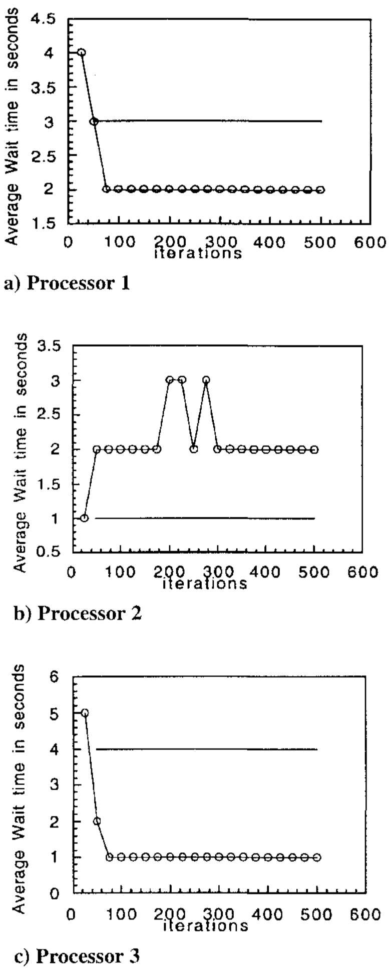

Fig. 3 Comparison of wait times of PVM nodes in a lightly loaded network: o, dynamic balance and ——, static balance.

Lightly loaded virtual machine network. All of the slave processors were lightly loaded, with no other user processes running. Figure 3 shows the variation of average wait times of each processor with the number of time steps. The least count of the measure of wait time is 1 s and is calculated using the time ( ) function. As shown in the plots, there was very little variation of wait times during the computation. In the dynamic balancing case, the wait times varied during the first few iterations and later remained constant because there was no significant load variation on the processors. In the static balancing case, the wait times remained constant during the period of computation as expected. The dynamic balancing wait times of two processors was reduced when compared to the static balancing case. Also, dynamic balancing increased the throughput by about 14%. The overhead time from the redistribution process was small in this case because the loads remained constant during the computation time.

Heavily loaded virtual machine network. All of the slave processors were heavily loaded with one or two other user processes running on the machines. Figure 4 shows the variation of average wait times of each processor with the number of time steps. The wait times in all of the processors varied randomly during the computation because of the varying load. The dynamic balancing algorithm tried to minimize the wait time of each processor. The average wait time of two processors was reduced by using the dynamic redistribution, whereas with one processor, there was an increase in wait time. The throughput with two processors was increased by 33%.

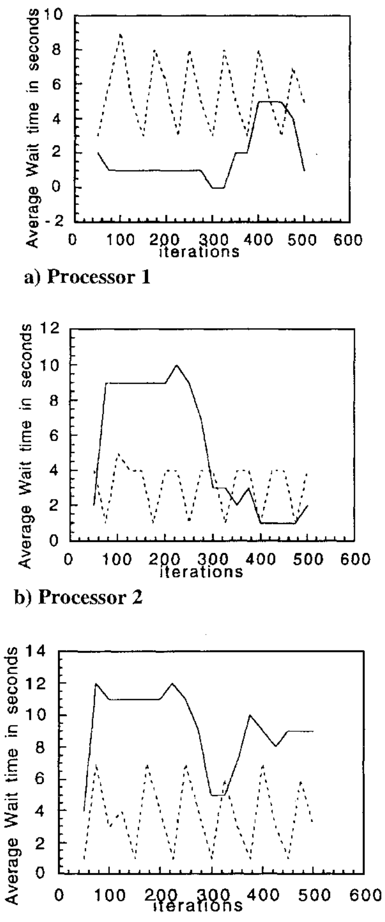

Fig. 4 Comparison of wait times of PVM nodes in a heavily loaded network: - - - -, dynamic balance and ——, static balance.

Concluding Remarks

An existing Navier–Stokes solver has been modified using the PVM software to compute flowfields in parallel using the network-based parallel computer. Excellent speedup in computation time was obtained using the parallel computations on a network of three workstations. The computation speed using the present network parallel computer was about 20% of that of a single processor of a Cray Y–MP supercomputer for the present application and the grid size. A simple dynamic load balancing algorithm was implemented to improve the performance of parallel computation.

Acknowledgments

This work was supported by the U.S. Army Research Office under the Center of Excellence in Rotorcraft Technology program. Thomas Doligalski was the Technical Monitor.

References

[1]Smith, M. H., "Distributed Parallel Flow Solutions Using Overset Grids," *Proceedings of the NASA Workshop on Distributed Computing for Aerosciences Applications*, 1993.

[2]Beguelin, A., Dongarra, J., Geist, A., Manchek, R., and Sunderam, V., "A User's Guide to PVM—Parallel Virtual Machine," Oak Ridge National Lab., ORNL/TM-11826, Oak Ridge, TN, March 1992.

[3]Smith, M., and Pallis, J., "MEDUSA—An Overset Grid Flow Solver for Network-Based Parallel Computer Systems," AIAA Paper 93-3312, July 1993.

[4]Weed, R. A., and Sankar, L. N., "Computational Strategies for Three-Dimensional Flow Simulations on Distributed Computer Systems," AIAA Paper 94-2261, June 1994.

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

[6]Bangalore, A., Tseng, W., and Sankar, L. N., "A Multi-zone Navier–Stokes Analysis of Dynamic Lift Enhancement Concepts," AIAA Paper 94-0164, Jan. 1994.

Bow Shock/Jet Interaction in Compressible Transverse Injection Flowfields

M. R. Gruber* and A. S. Nejad†
U.S. Air Force Wright Laboratory,
Wright–Patterson Air Force Base, Ohio 45433
T. H. Chen‡
Taitech, Inc., Beavercreek, Ohio 45440
and
J. C. Dutton§
University of Illinois at Urbana–Champaign,
Urbana, Illinois 61801

Introduction

TRANSVERSE injection of a gaseous fuel stream into a supersonic flow appears schematically in Fig. 1. This sketch illustrates the features of the flowfield in a plane through the spanwise jet centerline, where a three-dimensional bow shock forms ahead of the jet and interacts with the approaching turbulent boundary layer, resulting in separation. Previous investigations of this flowfield have provided clear visualizations of the large-scale vortices formed at the interface between the freestream and injectant fluids.[1–3] These eddies influenced the position of the bow shock.[2] Some images also showed the region enclosed by the separation shock where the bow shock and turbulent boundary layer interact.[1,2] This region contains locally high wall static pressures as found in both experiments[4] and numerical predictions.[5] Numerical investigations of a reacting transverse hydrogen jet in a supersonic airstream by Takahashi and Hayashi[6] showed relatively high static temperatures occurring within this separated zone. Erosion of the injector wall could occur as a result of this local high temperature zone.

The objective of the present work is to investigate the interaction between the upstream shock structure and the interfacial eddies in the jet fluid. The influence of injector geometry on the separation shock is also of interest. Both issues are addressed using a planar laser-based visualization technique to capture instantaneous images of the interaction created by circular and elliptical injectors fueled with air. Details regarding the facility and imaging technique used are available elsewhere.[7,8] Table 1 shows geometric features of the two injectors studied.

Results and Discussion

The freestream conditions were set such that $M_\infty = 1.98$, $p_{0,\infty} = 317$ kPa, and $T_{0,\infty} = 300$–302 K. Jet flow conditions were set so that the jet-to-freestream momentum flux ratio J,

$$J = \frac{(\rho u^2)_j}{(\rho u^2)_\infty} = \frac{(\gamma p M^2)_j}{(\gamma p M^2)_\infty} \qquad (1)$$

was identical in each case ($J = 2.90$). This parameter controls jet penetration into the crossflow.[1,9] Images were obtained for both circular and elliptical injection using air, cases C1A and E1A, with the major axis of the ellipse aligned with the freestream flow.

Instantaneous images from cases C1A and E1A appear in Figs. 2 and 3, respectively. Freestream fluid (light) flows from left to right, and the jet fluid (dark) enters from the lower edge (jets are centered at $x/d_{\text{eff}} = 0$). All of the images presented show some boundary-layer

Received Jan. 26, 1996; revision received Feb. 28, 1996; accepted for publication March 15, 1996. This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

*Aerospace Engineer, Advanced Propulsion Division, WL/POPT. Member AIAA.

†Senior Research Engineer, Advanced Propulsion Division, WL/POPT.

‡Senior Research Scientist. Senior Member AIAA.

§Professor, Department of Mechanical and Industrial Engineering. Associate Fellow AIAA.