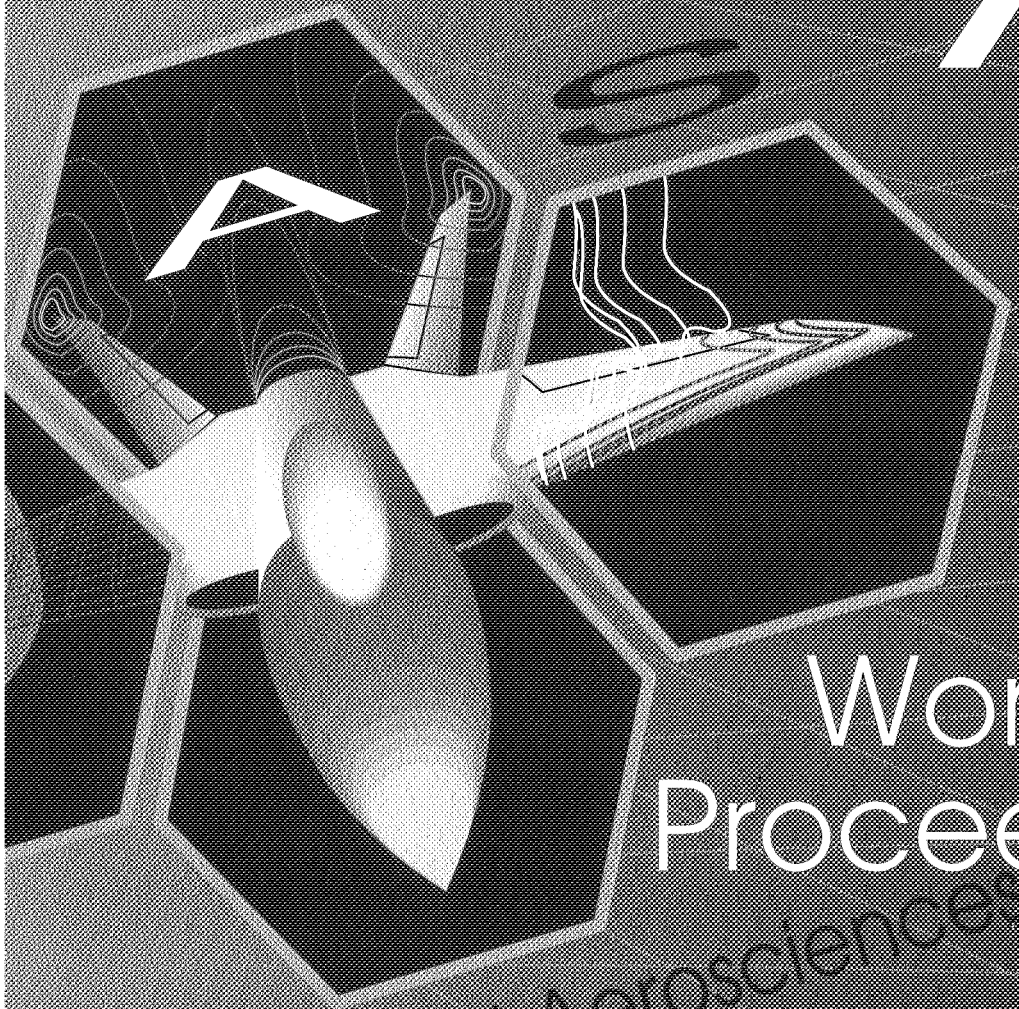
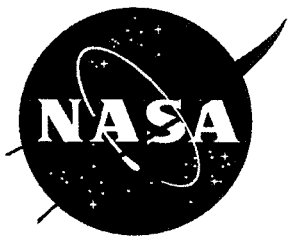
HPCCP/CAS Workshop 98



Workshop Proceedings

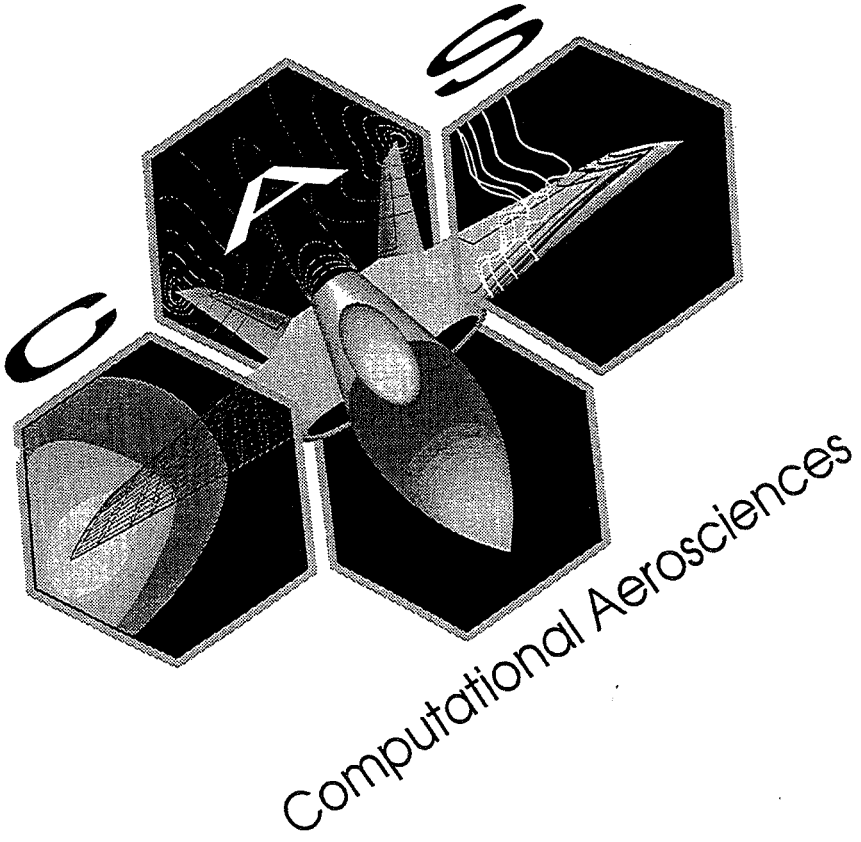
Computational Aerosciences

NASA/CP–1999-208757
January, 1999



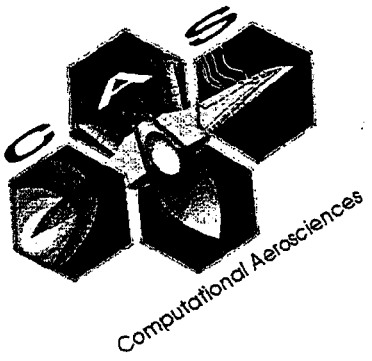
NASA Ames Research Center

HPCCP/CAS Workshop 98



Author:
Catherine Schulbach
NASA Ames Research Center
Editors:
Catherine Schulbach
NASA Ames Research Center
and Ellen Mata, Ratheon ITSS

TABLE OF CONTENTS
NASA HPCCP/CAS Workshop Proceedings



Introduction

This publication is a collection of extended abstracts of presentations given at the HPCCP/CAS Workshop held on August 24–26, 1998, at NASA Ames Research Center, Moffett Field, California. The objective of the Workshop was to bring together the aerospace high performance computing community, consisting of airframe and propulsion companies, independent software vendors, university researchers, and government scientists and engineers. The Workshop was sponsored by the High Performance Computing and Communications Program Office at NASA Ames Research Center.

The Workshop consisted of over 40 presentations, including an overview of NASA's High Performance Computing and Communications Program and the Computational Aerosciences Project; ten sessions of papers representative of the high performance computing research conducted within the Program by the aerospace industry, academia, NASA, and other government laboratories; two panel sessions; and a special presentation by Mr. James Bailey.

Catherine Schulbach

Catherine H. Schulbach
Manager, Computational Aeroscience Project
Workshop Chairperson

NASA'S HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS PROGRAM AND COMPUTATIONAL AEROSCIENCES PROJECT

Catherine H. Schulbach
CAS Project Manager

In 1977, over 21 years ago, 263 computational aerodynamicists and computer scientists met at Ames Research Center to identify the computer requirements for obtaining the desired solutions to their problems and to define the projected capabilities of the general purpose and special purpose processors of the early 1980's. They were motivated by: (1) the promise of an important new technological capability that did not have the limitations of wind tunnels, and (2) the dramatic rate of reduction in the net cost of conducting a simulation. The workshop was one of the cornerstones of NASA's entry into supercomputing. It led to the dedication of the Numerical Aerodynamic Simulation Facility ten years later. It also influenced the NASA's direction when NASA became one of the original participants in the Federal High Performance Computing and Communications (HPCC) Program that was established in 1991.

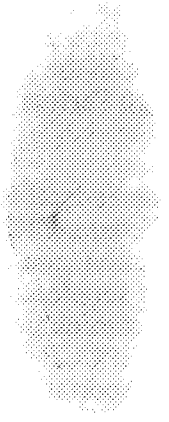
NASA's HPCC Program is an integral part of the Federal multi-agency collaboration in Computing Information and Communications (CIC). The Federal CIC programs invest in long-term research and development to advance computing, information, and communications in the United States. The NASA HPCC Program is aimed at boosting supercomputer speeds by a factor of a thousand to at least one trillion operations per second and communications capabilities by a factor of a hundred or more. The total NASA funding for HPCCP in FY 1998 is $73.8 million. This includes funds from Aeronautics and Space Transportation Technology, Space Science, Earth Science, and Education programs. Through HPCCP, NASA is also a major participant in the Next Generation Internet Initiative, a multi-agency effort that also includes the DOD, the Department of Commerce, the National Science Foundation, and the National Institutes of Health.

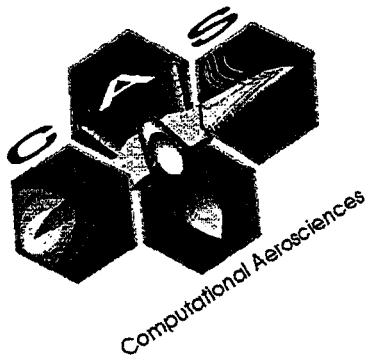
The goals of NASA's HPCC Program are to:
"Accelerate the development, application, and transfer of high-performance computing technologies to meet the engineering and science needs of the U.S. aeronautics, Earth and space science, spaceborne research, and education communities, and to enable Federal implementation of a Next Generation Internet."

The NASA HPCC Program is structured to contribute to broad Federal efforts while addressing agency-specific computational problems that are beyond projected near-term computing capabilities. These computational problems are called "Grand Challenges." NASA selected Grand Challenges in the areas of Computational Aerosciences (CAS), Earth and Space Science (ESS), and Remote Exploration and Experimentation. The Grand Challenge applications were chosen for their potential and direct impact to NASA, their national importance, and the technical challenges they present. The NASA HPCC Program is organized around these Grand Challenges with these three Grand Challenges forming three of the five HPCCP projects. Learning Technologies (LT) Project and the NASA Research and Education Network (NREN) are the two additional projects.

Computational aerosciences remain, 20 years later, an important piece of NASA's HPCC Program through the Computational Aerosciences Project. CAS is a computing and communications technology focused program oriented around the needs of the aeroscience community. Its mission is to: (1) accelerate development and availability of high-performance computing technology of use to the U. S. aerospace community, (2) facilitate adoption and use of this technology by the U. S. aerospace industry, and (3) hasten emergence of a viable commercial market for hardware and software vendors.

As we move into the twenty-first century the CAS Project faces enormous challenges of how to meet ever-increasing needs for computation while the market influence of supercomputing dwindles. Meanwhile, NASA's Strategic Enterprises continue to have bold goals that for achievement require orders-of-magnitude forward leaps in technology. Information systems technology, especially high-performance computing, is key to enabling such breakthroughs. Technology development is not sufficient. Better ways must be found to apply and transfer knowledge about aeronautics to the problem solving process. The problem solving process itself is becoming more and more complex as the result of dramatic improvements in the enabling computer hardware and software. In spite of the obstacles in the past, CAS made significant contributions toward making simulation an integral part of the design process and will approach the new challenges in partnership with industry and academia.





Session 1:

Advanced Computer Algorithms and Methodology

Mustafa Dindar, David Kenwright* and Ken Jansen
Rensselaer Polytechnic Institute
110 8th Street, SCOREC-CII 7011, Troy NY 12180
E-mail: mdindar@scorec.rpi.edu Phone: (518) 276-6795

ADAPTIVE COMPUTATION OF ROTOR-BLADES IN HOVER

Abstract

An adaptive refinement procedure is developed for computing vortical flows encountered in rotor aerodynamics. For this purpose, an error indicator based on interpolation error estimate is formulated and coded into an adaptive finite element framework. It is shown that the error indicator based on interpolation error estimate is effective in resolving the global features of the flow-field. Furthermore, for efficiency and problem size considerations, once the interpolation errors are reduced to acceptable levels, the adaptive refinement is done only in regions affected by the vortical flows. To do this a novel vortex core detection technique is used to capture vortex tubes. The combination of interpolation error estimate and vortex core detection technique proved to be very effective in computing vortical flow-field of rotor blades. Using this two-level adaptive refinement procedure the UH-60A BlackHawk rotor is analyzed in hover flow conditions.

Introduction

The ever increasing demand to understand the complex nature of the flow problems encountered in applied aerodynamics is being met by computational fluid dynamics (CFD). Pioneering work in fixed-wing aircraft aerodynamics also shed light into rotary-wing problems [1][2]. Nevertheless, the accurate prediction of rotor aerodynamics using existing CFD tools still remains a challenge because a rotorcraft often operates in more severe flow conditions than a fixed-wing aircraft. Even an isolated rotor blade system in hover conditions is under the influence of its own wake. Rotor blade performance calculations become a complex task for CFD due to strong blade and tip vortex interactions. Therefore, an accurate representation of the tip vortex is needed to be able to predict the thrust loading at the outboard portions of a rotor blade.

Efficient solutions of flow problems can be achieved by adaptive procedures. In h-adaptive techniques, the main idea is to increase or decrease the mesh resolution in the computational domain based on some measure of the error of the numerical solution procedure. Because of their simplicity, error indicators are usually employed to guide adaptive refinement procedures One possible avenue in designing an error indicator for flow problems is to use the interpolation error estimates. In this paper, we will be describing a simple error indicator derived from the interpolation error estimate for linear finite elements. Furthermore, in rotor-blade calculations, for computational efficiency and minimum problem size considerations, one may wish to augment the existing error indicator such that the adaptation is bracketed only in the vicinity of the tip vortex. Of course, this has to be done only after decreasing the global interpolation errors to acceptable levels over the computational domain. Such a procedure suggests a two-level adaptation technique, the first level resolving main features of the flow by reducing the interpolation errors and a second level that will concentrate on localized features, such as a tip vortex.

---

* MRJ Inc., NASA Ames Research Center

Governing Equations and the Numerical Procedure

Governing equations of an inviscid flow are given by the Euler Equations, which can be written in conservative quasi-linear form as follows:

$$U_{,t} + A_i U_{,i} = R \tag{1}$$

In Eq. (1), $U = [\rho, \rho u, \rho v, \rho w, \rho e]$ is the vector of conservative variables where ρ is mass density, u, v and w are fluid velocity components and e is the total energy per unit mass. A_i is the i^{th} inviscid flux vector and $R = [0, \rho v\Omega, -\rho u\Omega, 0, 0]$ is the source term to account for the rotating frame analysis about the z-axis. The inviscid flux vector, A_i is non-symmetric when written in terms of conservative variables. A symmetrization of inviscid flux vectors can be achieved by performing a transformation using a change of variables U→V [3]

$$V^T = \partial H / \partial U \tag{2}$$

where, $H = -\rho(s - s_o)$ is the generalized entropy function. Details of this transformation procedure can be found in [3]. After this transformation, Eq.(1) can be re-written as

$$\tilde{A}_o V_{,t} + \tilde{A}_i V_{,i} = R \tag{3}$$

where, $\tilde{A}_o = \partial U / \partial V$ and $\tilde{A}_i = A_i \tilde{A}_o$. Note that, $\tilde{A}_o$ is a symmetric positive-definite and $\tilde{A}_i$ is a symmetric matrix. A stabilized finite element formulation of Eq.(3) can be obtained using time-discontinuous Galerkin/least-squares method [3]. All the calculations in this paper are done using a Galerkin/least-squares method, and the GMRES solution technique is used as a linear equation solver.

Error Estimation/Indication

Understanding the flow fields with vortical structures and adaptive mesh refinement requires an understanding of finite element approximation. Interpolation errors exist due to finite dimensional space approximation and depend on the order of the finite element basis. Therefore, an error indicator that utilizes derivatives of velocities in a vortical flow field has to be based on the interpolation error of a particular finite element approximation.

Let us seek L_2-norms of error of the vector field $\boldsymbol{u}$ in a finite element procedure which utilizes linear piecewise polynomials in R^3. The semi-norm of error is given by [4]

$$|e|_{o,2} \le Ch^2 |u|_{2,2} \tag{4}$$

In this case, evaluation of the error requires second partial derivatives of $\boldsymbol{u}$. Let us denote the finite element error indicator by the symbol, θ_i, then a practical error indicator using L_2-norm of the second derivatives of velocity vector is written as

$$\theta_i = h^2 \left(\int_{\Omega^h} \sum_{i+j+k=2} \left| \frac{\partial^2 u}{\partial x_i \partial x_j \partial x_k} \right|^2 d\Omega \right)^{1/2} \tag{5}$$

The second derivatives involved in Eq.(5) are obtained by using a quadratic reconstruction process which requires the solution of a least-squares problem.

Although, in principle one could use the error indicator given in Eq.(5) to drive the adaptation to resolve all the features of a flow problem, because of computational efficiency and storage limitations of current computer architectures, this would not be practical. It is desirable to monitor the global accuracy during the adaptive solution procedure and if permissible resort to more localized adaptation of the mesh for small-scale features of specific interest, such as a tip vortex in rotor-blade calculations.

From the topological point of view, a vortex core contains features that can be used to distinguish it from other regions of the flow-field. First of all a vortex core is a stationary point where flow trajectories spiral in a plane and the vorticity is maximum at the core center. For a real fluid viscous effects cause the core of a vortex to rotate approximately as if it were a rigid body, hence, on the plane of rotation $\boldsymbol{u} \rightarrow 0$ at the vortex core. For inviscid flows with the existence of numerical diffusion, a vortex behaves much like it is in a viscous flow. Therefore, we identify a vortex core to be a stationary point. Also, the velocity-gradient tensor, ∇u, has complex eigenvalues if the stationary point is a vortex. This approach has been used recently to define and extract flow topology information for scientific visualization [5].

In our adaptive solution procedure, we use the eigenvalue extraction method to identify the vortical regions in the flow and modify the definition of error indicator given in Eq.(5) by the following logic

$$\theta^* = \begin{cases} \theta & \text{if a vortex exits on } \Omega^h \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Numerical Results

The UH-60A Blackhawk rotor-blade system is selected to carry out the adaptive hover calculations. An experimental scaled model [6] of the UH-60A blade is used. A summary of computational results are presented here; more details can be found in [7].

UH-60A at zero thrust

The flow-field conditions used for this case are as follows: tip Mach number M_t=0.628, rotor speed Ω = 1427 rpm, thrust coefficient $C_T/\sigma = 0.0$, collective setting $\theta_{.75} = 0.11°$ and coning $\beta_0 = -0.20°$.

Before doing any adaptive solutions, a mesh sensitivity study is done to establish initial element size that can be used as a starting point. First, a mesh containing 760,000 tetrahedral elements (see Fig. 1), is used to calculate the flow-field. For this mesh the average global element size is about 0.125R and it is more or less uniform throughout the domain with the exception of the blade surface which has a finer (down to 0.001R) mesh resolution than the rest of the domain. Second, a finer mesh, which is manually refined near the tip-path-plane of the blade, is generated (see Fig. 2). The initial size of this second mesh is 1,100,400 tetrahedral elements. The average element size in the domain that is refined manually is about 0.002R. A comparison between the flow-field results from the two meshes revealed that calculations with the finer mesh are considerably more accurate than the coarse mesh results. Although the fine mesh does not resolve all the features of the flow-field, it is a good starting point for adaptive refinement.

Next, the adaptive procedure is applied to the fine initial grid results to improve the solution further and to locate the vortex structure. With the adaptive procedure, the mesh is refined in two levels, the first level using the interpolation error estimate and the second level using the vortex core detection technique. At this point, the mesh size reached 2,164,704 elements. Figures 3 and 4 compare computed values of sectional thrust and sectional torque with experiment. Notice that the adapted grid enhances the accuracy of the solution near the tip of the blade. Finally, Figure 5 shows the predicted vortex flow structure which is calculated by the vortex core detection technique. Note that there is an unexpected vortex

tube located between 75%-80% radius of the blade. This vortex tube is independent of the tip vortex. The reason for the existence of a vortex tube at 75% radius is hypothesized to be the differential change in thrust loading from positive to negative as shown in Figure 3. Finally, there is also a vortex tube emanating from the tip of the blade, but it does not seem to have a very strong interaction with the blade tip.

UH-60A at design thrust

The final and the most challenging case analyzed for the UH-60A blade is a design thrust case where the flow conditions are: tip Mach number M_t=0.628, rotor speed Ω = 1425 rpm, thrust coefficient C_T/σ = 0.085, collective setting $\theta_{.75}$ = 10.47°, coning β_o = 2.31°. This particular case offers a stronger tip vortex structure than the zero thrust case.

The computed sectional thrust and torque coefficient distributions are presented in Figures 6 and 7, respectively. The initial mesh, which could not resolve the tip vortex for the first 90 degrees azimuth, resulted in a poor sectional thrust distribution particularly at the outboard portion of the blade. We emphasize here that all of our attempts to capture the correct distribution of sectional thrust before resolving the tip vortex properly failed. Progressive adaptive refinement steps, using both interpolation error estimate and the vortex core detection technique, resulted in a correct prediction of tip vortex structure. The sectional thrust distribution with the final adapted mesh shows a remarkable improvement with respect to initial mesh results. Therefore, it has been concluded that computing at least the first 90 degrees azimuth travel of the tip vortex is essential for this high thrust UH-60A blade case. Figure 7 compares the sectional torque coefficient for the refined and initial meshes against experimental data. Figure 8 shows the change in mesh resolution form initial mesh to adapted mesh at a cross-section taken at 15° behind the blade. Notice that the mesh resolution increased in the vicinity of the tip vortex for the adapted mesh. Finally, figure 9 shows the first 90° azimuth travel of the tip vortex with the adapted mesh.

References

[1] Srinivasan G.R., Beader J.D., Obayashi S., and McCroskey W.J., "Flowfield of a lifting rotor in hover: A Navier-Stokes simulation", AIAA Journal, Vol. 30, No. 10, pp. 2371-2378. Oct. 1992.

[2] Strawn R.C., and Barth T.J., "A finite volume Euler solver for computing rotary-wing aerodynamics on unstructured meshes", AHS 48th Annual Forum Proceedings, pp. 419-428, Jun. 1992.

[3] Hugues T.J.R., Franca L.P. and Hulbert G.M., "A new finite element formulation for fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive equations", Journal of Applied Mechanics, Vol. 73, pp. 173-189, 1989.

[4] Oden J.T., and Reddy J.N., "An introduction to mathematical theory of finite elements", John Wiley & Sons, Inc., 1976.

[5] Kenwright D., and Haimes R., "Vortex identification - application in aerodynamics", IEEE/ACM Proceedings of Visualization '97 ACM Press, Phoenix, AZ, Oct. 1997.

[6] Lorber P.F., Stauter R.C., Pollack M.J., and Landgrebe A.J., "A comprehensive hover test of airloads and airflow of an extensively instrumented model helicopter rotor", Vol I-V, USAAVSCOM TR 91-D-16E, 1991.

[7] Dindar M, Lemnios A.Z., Shephard M.S., Flaherty J.E, Jansen, K., "An adaptive procedure for rotorcraft aerodynamics", AIAA Paper 98-2417, 16th Applied Aerodynamics Conference, Albuquerque, NM, July 1998.

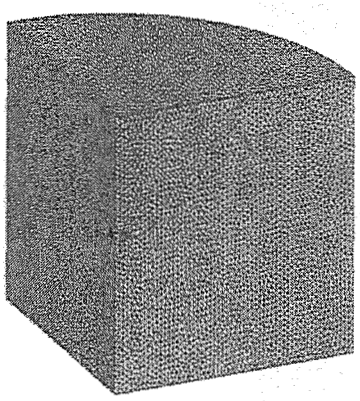


Figure 1: Initial coarse-wake mesh

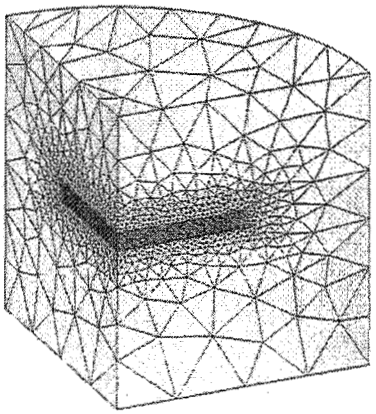


Figure 2: Initial fine-wake mesh

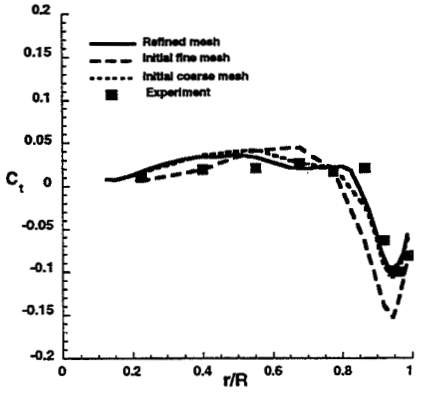


Figure 3: Sectional thrust distribution

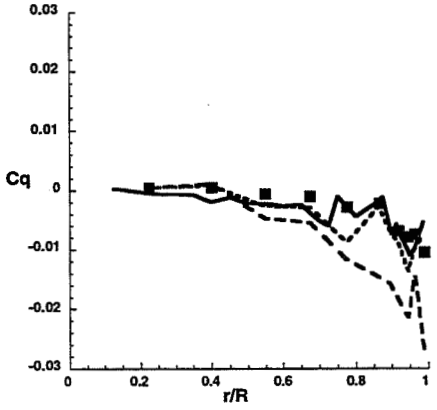


Figure 4: Sectional torque distribution

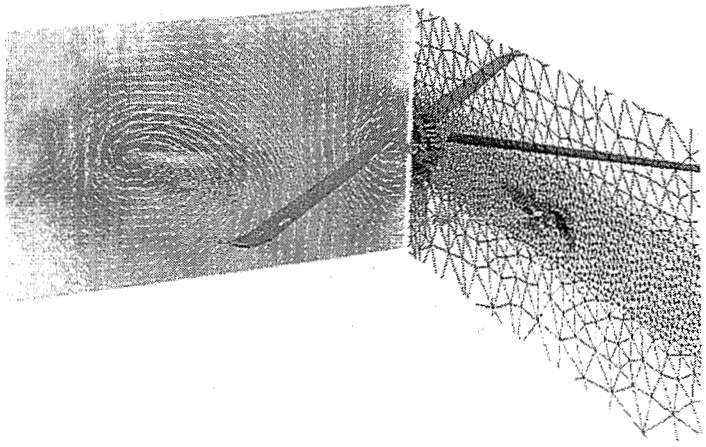


Figure 5: Computed vortex flow structure for the UH-60A blade at zero thrust

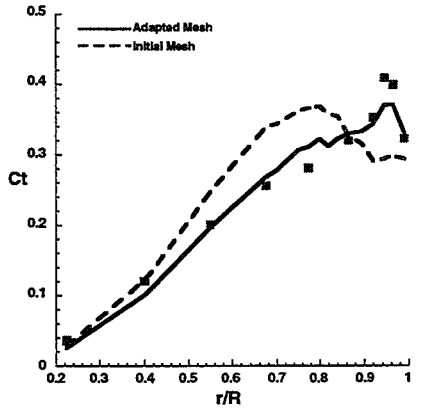


Figure 6: Sectional thrust distribution

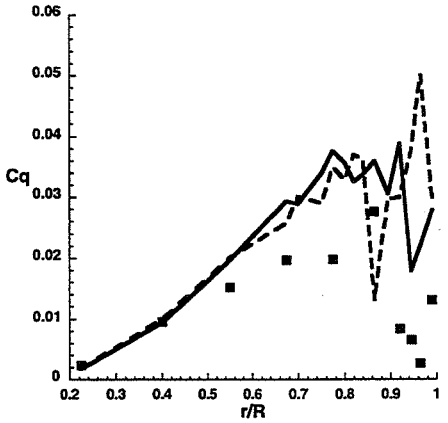


Figure 7: Sectional torque distribution

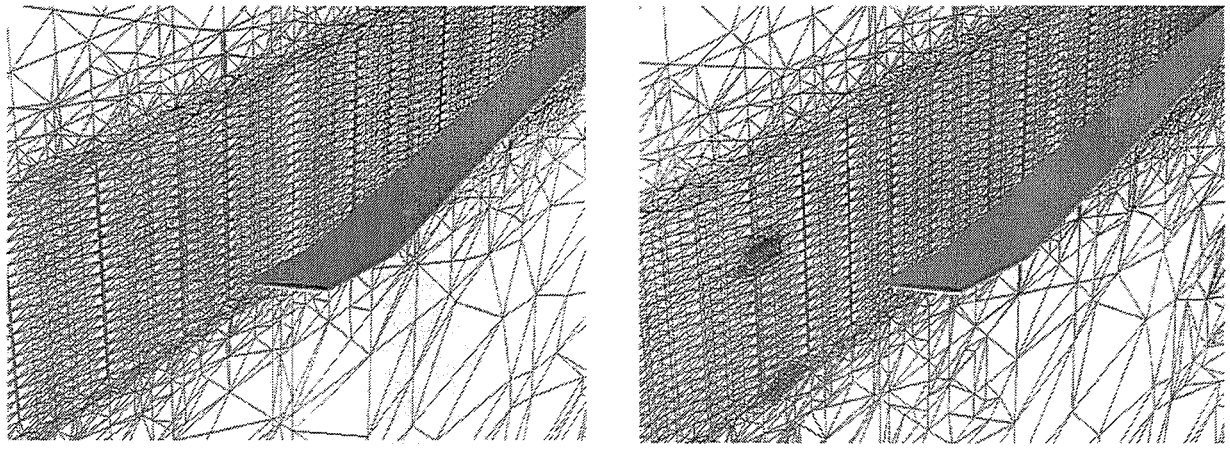


Figure 8: Initial and adapted mesh cross-section at 15° behind the blade

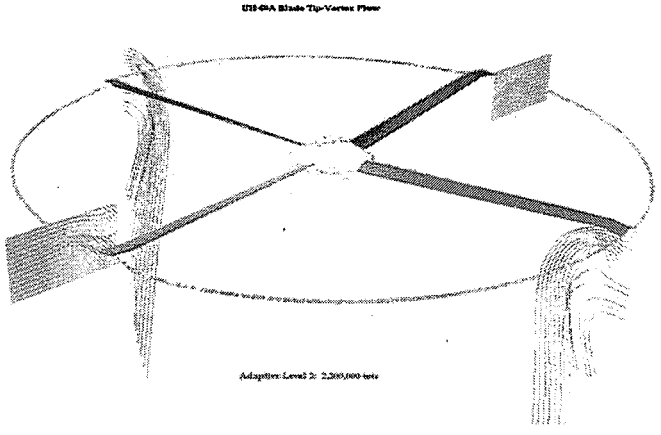


Figure 9: Adaptively resolved tip vortex structure for the UH-60A blade at design thrust

AUTOMATED DEVELOPMENT OF ACCURATE ALGORITHMS AND EFFICIENT CODES FOR COMPUTATIONAL AEROACOUSTICS

John W. Goodrich and Rodger W. Dyson
NASA Lewis Research Center
Cleveland, OH 44135

Abstract

The simulation of sound generation and propagation in three space dimensions with realistic aircraft components is a very large time dependent computation with fine details. Simulations in open domains with embedded objects require accurate and robust algorithms for propagation, for artificial inflow and outflow boundaries, and for the definition of geometrically complex objects. The development, implementation, and validation of methods for solving these demanding problems is being done to support the NASA pillar goals for reducing aircraft noise levels. Our goal is to provide algorithms which are sufficiently accurate and efficient to produce usable results rapidly enough to allow design engineers to study the effects on sound levels of design changes in propulsion systems, and in the integration of propulsion systems with airframes. There is a lack of design tools for these purposes at this time.

Our technical approach to this problem combines the development of new algorithms with the use of Mathematica and Unix utilities to automate the algorithm development, code implementation, and validation. We use explicit methods to ensure effective implementation by domain decomposition for SPMD parallel computing. There are several orders of magnitude difference in the computational efficiencies of the algorithms which we have considered. We currently have new artificial inflow and outflow boundary conditions that are stable, accurate, and unobtrusive, with implementations that match the accuracy and efficiency of the propagation methods. The artificial numerical boundary treatments have been proven to have solutions which converge to the full open domain problems, so that the error from the boundary treatments can be driven as low as is required. The purpose of this paper is to briefly present a method for developing highly accurate algorithms for computational aeroacoustics, the use of computer automation in this process, and a brief survey of the algorithms that have resulted from this work. A review of computational aeroacoustics has recently been given by Lele [11].

1 Linearized Euler Equations

The Linearized Euler Equations [9] can be used for the simulation of sound propagation if a steady flow field is known. In the isentropic case with a constant mean flow, the nondimensionalized equations for the acoustic disturbance are:

$$\begin{aligned} \frac{\partial u_j}{\partial t} + M_i \frac{\partial u_j}{\partial x_i} + \frac{\partial p}{\partial x_j} &= 0, \\ \frac{\partial p}{\partial t} + M_i \frac{\partial p}{\partial x_i} + \frac{\partial u_i}{\partial x_i} &= 0, \end{aligned} \tag{1}$$

where p is the pressure, u_j is the j^{th} velocity component of the disturbance, and M_i is the i^{th} component of the convection field. As an illustrative example we will present a summary of algorithm development for propagation in one space dimension.

1.1 Algorithm Development in 1D

We begin algorithm development by interpolating the known data at t_n about the grid point x_i with order D polynomials in x,

$$\begin{aligned} u(x_i + x, t_n) &\approx u_a(x) = \textstyle\sum_{\delta=0}^{D} \frac{1}{\delta!} u_\delta x^\delta, \\ p(x_i + x, t_n) &\approx p_a(x) = \textstyle\sum_{\delta=0}^{D} \frac{1}{\delta!} p_\delta x^\delta. \end{aligned} \tag{2}$$

The expansion coefficients u_δ and p_δ for the interpolants (2) are obtained from the known data on the grid by the Method of Undetermined Coefficients, and approximate the spatial derivatives of u and p.

In one space dimension, the linearized Euler Equations (1) can be diagonalized and solved by the

Method of Characteristics, with general solution:

$$\begin{aligned} u(x,t) &= +\tfrac{1}{2}u_o(x-(M+1)t) \\ &\quad +\tfrac{1}{2}p_o(x-(M+1)t) \\ &\quad +\tfrac{1}{2}u_o(x-(M-1)t) \\ &\quad -\tfrac{1}{2}p_o(x-(M-1)t), \\ p(x,t) &= +\tfrac{1}{2}u_o(x-(M+1)t) \\ &\quad +\tfrac{1}{2}p_o(x-(M+1)t) \\ &\quad -\tfrac{1}{2}u_o(x-(M-1)t) \\ &\quad +\tfrac{1}{2}p_o(x-(M-1)t), \end{aligned} \tag{3}$$

where $u_o(x) = u(x,0)$ and $p_o(x) = p(x,0)$. If the local spatial interpolants (2) are taken as initial data for the solution (3) of (1), then the global solution is locally approximated in time and space by

$$\begin{aligned} u(x_i+x,t_n+t) &\approx u_A(x_i+x,t_n+t) \\ &= +\tfrac{1}{2}u_a(x-(M+1)t) \\ &\quad +\tfrac{1}{2}p_a(x-(M+1)t) \\ &\quad +\tfrac{1}{2}u_a(x-(M-1)t) \\ &\quad -\tfrac{1}{2}p_a(x-(M-1)t), \\ p(x_i+x,t_n+t) &\approx p_A(x_i+x,t_n+t) \\ &= +\tfrac{1}{2}u_a(x-(M+1)t) \\ &\quad +\tfrac{1}{2}p_a(x-(M+1)t) \\ &\quad -\tfrac{1}{2}u_a(x-(M-1)t) \\ &\quad +\tfrac{1}{2}p_a(x-(M-1)t). \end{aligned} \tag{4}$$

At the new time level $t_{n+1} = t_n + k$, the solution at the grid point x_i is given by

$$\begin{aligned} u_i^{n+1} = u_A(0,k) &\approx u(x_i,t_n+k), \\ p_i^{n+1} = p_A(0,k) &\approx p(x_i,t_n+k). \end{aligned} \tag{5}$$

Equation (5) is a general algorithm form, and represents a family of algorithms with properties that depend upon the interpolant. This form can be recast in various ways, such as an expansion in k, or as a conventional finite difference equation.

The general algorithm form (5) is derived from the exact local solution (4), so that the algorithms in this family will correctly incorporate the local wave dynamics of (1) for the spatial interpolant (2). The order of accuracy in both space and time for any of the algorithm realizations of (5) is the same as the order of interpolation. All of the algorithms based upon the general form (5) are local single step explicit methods, and each is stable if $\frac{k}{h} \leq \frac{1}{1+|M|}$, for any M. Note that there is a change in perspective, away from approximating particular terms in a system of partial differential equations, and toward the local approximation of the solution of the system. Local approximation of the solution in both space and time is the viewpoint of both the Method of Characteristics, and the Cauchy-Kovalevskaya Theorem [2]. Further details are in [3, 4, 5, 6].

1.2 Hermitian Interpolation

We have developed a family of high resolution algorithms with Hermitian interpolation using u, p, and their spatial derivatives up to some order. Hermitian interpolation has been used by Collatz [1], and in more recent work such as [13]. If u and p are interpolated at time t_n by (2), then their first x derivatives can be approximated by

$$\begin{aligned} \tfrac{\partial u}{\partial x}(x_i+x,t_n) &\approx \tfrac{\partial u_a}{\partial x}(x) = \textstyle\sum_{\delta=1}^{D} \frac{1}{(\delta-1)!} u_\delta x^{\delta-1}, \\ \tfrac{\partial p}{\partial x}(x_i+x,t_n) &\approx \tfrac{\partial p_a}{\partial x}(x) = \textstyle\sum_{\delta=1}^{D} \frac{1}{(\delta-1)!} p_\delta x^{\delta-1}, \end{aligned} \tag{6}$$

with similar expressions for other spatial derivatives. Hermitian interpolation is used to obtain the spatial interpolants (2) by the Method of Undetermined Coefficients with constraints for u and p from (2), and for their spatial derivatives from (6). As an example, a cubic interpolant is possible on a two point stencil with data for both u and $\frac{\partial u}{\partial x}$, where two constraints are from (2), and two from (6). The spatial interpolants (2) are approximations for u and p, and their x derivatives are obtained from (6) with the same expansion coefficients.

The derivatives are evolved along with u and p by algorithms which are consistent with the algorithm form (5). The spatial derivatives satisfy a system of equations which is obtained by differentiating (1), and the derivatives of (3) are exact solutions for u_x and p_x. The spatial derivatives of the local solution approximation (4) provide local approximates of u_x and p_x in space and time which are consistent with (4). The local derivative approximations are used to obtain algorithms for u_x and p_x just as (4) is used to obtain the algorithms (5) for u and p. The propagation algorithms with derivative data and Hermitian interpolation on central stencils are unstable. Stable, diffusive algorithms can be developed on alternating grids with a half time step $\frac{k}{2}$. Half time step algorithms can be composed to obtain algorithms for a full time step k. Further details are in [4, 5, 6].

1.3 Algorithms in 2D and 3D

Algorithm development for (1) in any number of space dimensions combines a local spatial interpolant with a propagator. Genuinely multidimensional spatial interpolation is used. As an example, a biquadratic interpolant for p on a symmetric three

by three stencil can be written as

$$p_a(x,y) = \sum_{\alpha,\beta=0}^{2} \frac{1}{\alpha!\beta!} p_{\alpha,\beta} x^\alpha y^\beta. \qquad (7)$$

Notice that the biquadratic interpolant (7) has coefficients representing cross derivatives up to the fourth order, even though differentiation in either x or y only goes up to the second order. Many interpolants are possible, with properties that depend upon the choice of stencil and data. The Method of Undetermined Coefficients can be used to obtain the expansion coefficients from the known data on the interpolation stencil.

The Linearized Euler Equations are non diagonalizable in multiple space dimensions, with information propagating along characteristic surfaces, so that the Method of Characteristics cannot be used to find an exact general solution for (1). A local polynomial expansion in space and time can be found by applying the method used in the proof of the Cauchy-Kovalevskaya Theorem [2], with time derivatives expressed in terms of space derivatives. A Cauchy-Kovalevskaya process has also been used in [8]. If a finite degree polynomial expansion in space is used as the initial data, then it is possible to obtain a finite degree polynomial which is an exact solution for the linear system (1). As in the one dimensional case, multidimensional algorithms are obtained from the polynomial solution forms in space and time by using the local spatial interpolants as initial data.

Artificial boundary conditions are a significant issue in multiple space dimensions, but cannot be addressed in any significant detail here. Recent work has provided new radiation boundary conditions that can provide any required degree of accuracy, and that have algorithmic implementations which are completely analogous to the propagation algorithms. See [6, 7] for more details.

2 Automated Development

We have developed and used algorithms up to the 29^{th} order in space and time, in one, two and three space dimensions. These are all local, single step, explicit algorithms. One family of algorithms interpolates and propagates only the solution variables on central stencils. and the stencil widths increase with the order of the algorithm. Wide stencils lead to a variety of problems that limit the utility of these methods. A second family of our algorithms interpolates and propagates the solution variables and some of their spatial derivatives. These algorithms use alternating offset grids, with high order derivative data on stencils that vary from two to six grid points wide, where the derivative order increases with the order of the algorithm. Algorithm complexity can arise in either the spatial interpolation or the time evolution, and the complexity increases both with the accuracy of the algorithm and with the number of space dimensions.

2.1 Algorithm Complexity

In multiple space dimensions, our algorithms use tensor product interpolants such as (7). An order D algorithm in N space dimensions requires the solution of a linear interpolation problem for $C = (D+1)^N$ expansion coefficients by inverting a $C \times C$ matrix. The number of expansion coefficients for the pressure or any of the velocity components is tabulated by the algorithm order and the spatial dimension in Table 1. Solving high order

Table 1: Number of Coefficients

Order	1D	2D	3D
2	3	9	27
4	5	25	125
8	9	81	729
10	11	121	1331
12	13	169	2197
14	15	225	3375
16	17	289	4913

interpolation problems in three dimensions can tax the capabilities of current symbolic manipulators on commonly available computer systems. The tensor product interpolants can reformulate this problem to with dimensional recursion as a nested sequence of one dimensional problems for $D+1$ coefficients. The one dimensional problems are easy to solve and code, so that the multidimensional tensor interpolants are readily implemented with succinct, validated codes. It is well known [12] that the precision of computer arithmetic can have a dramatic impact on higher order difference methods. The use of 64-bit arithmetic effectively limits useful finite difference methods to less than 20^{th} order.

Local Cauchy-Kovalevskaya expansions are used to derive our algorithms, and the resulting propagation methods are equivalent to high order series expansions in time. In N space dimensions, an order D with a local exact polynomial solution to (1) and a tensor interpolant will have terms up to order $N \times D$ both in the interpolant as cross derivatives, and in the time evolution propagator. Explicitely

developing these high order propagator forms can also tax currently available software and hardware. Recurrence relationships from the evolution equations can simplify the propagator. As an example, in one space dimension, system (1) gives

$$\begin{aligned} \frac{\partial}{\partial t}(\frac{\partial^\delta u}{\partial x^\delta}) &= -M\frac{\partial^{\delta+1} u}{\partial x^{\delta+1}} - \frac{\partial^{\delta+1} p}{\partial x^{\delta+1}}, \\ \frac{\partial}{\partial t}(\frac{\partial^\delta p}{\partial x^\delta}) &= -M\frac{\partial^{\delta+1} p}{\partial x^{\delta+1}} - \frac{\partial^{\delta+1} u}{\partial x^{\delta+1}}, \end{aligned} \tag{8}$$

for the first time derivatives of the δ^{th} x derivatives of u and p, with the recursion relationship

$$\begin{aligned} \frac{\partial^{\kappa+1}}{\partial t^{\kappa+1}}(\frac{\partial^\delta u}{\partial x^\delta}) &= -M\frac{\partial^{\delta+\kappa+1} u}{\partial t^\kappa \partial x^{\delta+1}} - \frac{\partial^{\delta+\kappa+1} p}{\partial t^\kappa \partial x^{\delta+1}}, \\ \frac{\partial^{\kappa+1}}{\partial t^{\kappa+1}}(\frac{\partial^\delta p}{\partial x^\delta}) &= -M\frac{\partial^{\delta+\kappa+1} p}{\partial t^\kappa \partial x^{\delta+1}} - \frac{\partial^{\delta+\kappa+1} u}{\partial t^\kappa \partial x^{\delta+1}}. \end{aligned} \tag{9}$$

Similar forms are available in multiple space dimensions. In the case of algorithms which use Hermitian data, the solution variables and some of their spatial derivatives must all be evolved in time, so that the sub-calculations for the recursion relationships produce time evolution terms that can be used more than once in evolution equations. This results in a relative computational efficiency for methods which use higher order derivative information, since the time evolution terms are used more often, and for methods in higher space dimensions, since the time evolution terms are more complex.

2.2 Automation

Reducing the algorithms to simpler recursive forms with repeatable and extensible patterns has allowed for the automated development of complex algorithms which do not exceed the capabilities of available compilers. Computer algebra packages such as Mathematica or Maple provide the capabilities that can be used for the automation. We have used Mathematica in this work. A wide variety of algorithms have been developed and studied in this work, and we have used FORTRAN shells that were simply written into Mathematica for code development. Mathematica is used to develop the various algorithms, and to convert them into FORTRAN segments that are inserted into the FORTRAN shells. Algorithms are broken into a series of subroutines which are contained in separate Mathematica routines, and results from the Mathematica and FORTRAN routines are compared for validation. This process can be used to easily and quickly generate and validate a complex new algorithm and large code implementation. We have been successful with this approach for both propagation methods and radiation boundary conditions, and are currently working on high order geometry definition. The automated development of algorithms and codes can be extended to other systems besides the Linearized Euler Equations, and we are currently working on problems with variable coefficients and nonlinear terms.

3 Typical Numerical Results

The tensor product algorithms have performance which is relatively independent of the space dimension. Typical numerical experiments for the Linearized Euler Equations (1) will be presented in one space dimension, with the periodic solution

$$\begin{aligned} p(x,t) &= \tfrac{1}{2}(\sin[\pi(x-t)] + \sin[\pi(x+t)]), \\ u(x,t) &= \tfrac{1}{2}(\sin[\pi(x-t)] - \sin[\pi(x+t)]), \end{aligned} \tag{10}$$

for $x \in (-1, 1]$. Note that both the wavelength and period are 2, and that we are assuming $M = 0$, with no mean convection. Grid refinement data will be reported from various algorithms for mesh sizes $h = 2^{-m}$. The grid ratio is always $\lambda = \frac{k}{h} = \frac{8}{10}$. The algorithm names that end in *ex* correspond to central explicit methods, and the *cn* prefix to their names refers to a stencil of width n, and an algorithm of order $n-1$. The algorithm names that end in *s2* correspond to Hermitian methods on staggered grids, the *cn* prefix to their names refers to a stencil of width $n-1$, and the *od* central part of their names implies the use of derivative data up to order d. These algorithms have order $(d+1) \times (n-1) - 1$.

Figure 1 shows the maximum absolute error in p or u at $t = 10$ plotted against the grid density. The maximum absolute error represents the accumulated error in both space and time over the course of the entire simulation. Note that $t = 10$ corresponds to five periods of wave propagation. The vertical axis in Figure 1 is the $\log_{10}$ of the maximum error, and the horizontal axis is the $\log_2$ of the number of grid points per wavelength. The slopes of the plot lines in Figure 1 corroborate the orders of accuracy for the corresponding methods. The sequence of plot lines from upper right to lower left is roughly the same as the sequence of the orders of the methods. A very interesting feature of Figure 1 is the error data at four grid points per wavelength. At this grid resolution most of the methods have $O[1]$ or $O[10^{-1}]$ error, but the errors of the fifth order *c3o2s2* and seventh order *c5o1s2* methods are $O[10^{-2}]$, while the errors of the seventh order *c3o3s2* and eleventh order *c5o2s2* methods are $O[10^{-4}]$ and $O[10^{-6}]$, respectively. The high resolution of these methods at four grid points per wavelength produces lower errors at greater grid

densities than higher order methods with poor resolving power. If this simulation is conducted with 8 grid points per wavelength out to $t = 100,000$, or for 50,000 periods, then the maximum absolute error for the $c5o1s2$, $c3o3s2$, and $c5o2s2$ methods are $O[10^{-1}]$, $O[10^{-3}]$, and $O[10^{-6}]$, respectively.

The performance of the Hermitian algorithms is quite good with respect to grid density, but these algorithms are more complex than the conventional $cno0ex$ methods, so that it is natural to ask which algorithms are more efficient, or how their error levels compare with respect to the total number of FLOPS. Figure 2 presents the error data from Figure 1 replotted against the $\log_{10}$ of the total number of multiplications required to compute from $t = 0$ to $t = 10$. The number of multiplications is counted since modern computers have a multiply and add instruction. The essential point in Figure 2 is that the efficiency of the algorithms increases with the order of the algorithm, irrespective of the complexity of the algorithm. Note also that there is a range of at least four orders of magnitude in the number of FLOPS that are required to attain the modest error level of $O[10^{-4}]$ at $t = 10$. Kreiss and Oliger [10] have shown that the relative efficiency of higher order methods increases as the error level is lowered, and as t is increased. This means that the range of required effort will increase across the spectrum of algorithms as the error level is lowered, or as the time is increased. We have also observed that the relative efficiency of our higher order methods increases with the number of space dimensions.

4 Conclusions

We have produced an automated method for algorithm development that leads to exceptional numerical algorithms. Our algorithms are local, explicit methods with the same order of accuracy in both space and time, and are easily parallelized. We have developed and used algorithms up to the 29^{th} order in one, two, and three space dimensions. The class of algorithms that uses Hermitian interpolation has truly spectral like resolution. The efficiency of our algorithms improves as their complexity increases, in terms of both FLOPS and required memory. The relative efficiency of our higher order methods is several orders of magnitude greater than conventional algorithms. The only limit to performance appears to be machine accuracy. Local, explicit, high order, and high resolution methods that are capable of efficiently maintaining accuracy over large times and distances are particularly suitable for large scale scientific computing on high end parallel systems.

References

[1] L. Collatz, *The Numerical Treatment of Differential Equations*, (Springer, Berlin-New York, 1960).

[2] P. Garabedian, *Partial Differential Equations*, (Wiley, New York, 1964).

[3] J. W. Goodrich, "An Approach to the Development of Numerical Algorithms for first Order Linear Hyperbolic Systems in Multiple Space Dimensions: The Constant Coefficient Case," NASA TM 106928, 1995.

[4] J. W. Goodrich, "Accurate Finite Difference Algorithms," NASA TM 107377, 1996.

[5] J. W. Goodrich, "High Accuracy Finite Difference Algorithms for Computational Aeroacoustics," AIAA 97-1584, *The 3rd AIAA/CEAS Aeroacoustics Conference*, Atlanta, GA, May, 1997.

[6] J. W. Goodrich and T. Hagstrom, "Accurate Algorithms and Radiation Boundary Conditions for Linearized Euler Equations," AIAA 96-1660, *The 2nd AIAA/CEAS Aeroacoustics Conference*, State College, PA, May, 1996.

[7] J. W. Goodrich and T. Hagstrom, "A Comparison of Two Accurate Boundary Treatments for Computational Aeroacoustics," AIAA 97-1585, *The 3rd AIAA/CEAS Aeroacoustics Conference*, Atlanta, GA, May, 1997.

[8] A. Harten, B, Engquist, S. Osher and S. R. Chakravarthy, "Uniformly High Order Accurate Essentially Non-oscillatory Schemes, III," *J. Comput. Phys.*, **71**, 231 (1987).

[9] H. O. Kreiss and J. Lorenz, *Initial Boundary Value Problems and the Navier Stokes Equations*, (Academic Press, New York, 1989).

[10] H. O. Kreiss and J. Oliger, "Comparison of Accurate Methods for the Integration of Hyperbolic Equations," *Tellus,* **24**, p199, 1972.

[11] S. K. Lele, "Computational Aeroacoustics: A Review," AIAA 97-0018, *The 35th Aerospace Sciences Meeting*, Reno, NV, January, 1997.

[12] M.J.D. Powell, *Approximation Theory and Methods*, (Cambridge University Press, Cambridge, 1981).

[13] H. Takewaki, A. Nishiguchi, and T. Yabe, *J. Comput. Phys.*, **61**, p261, 1985.

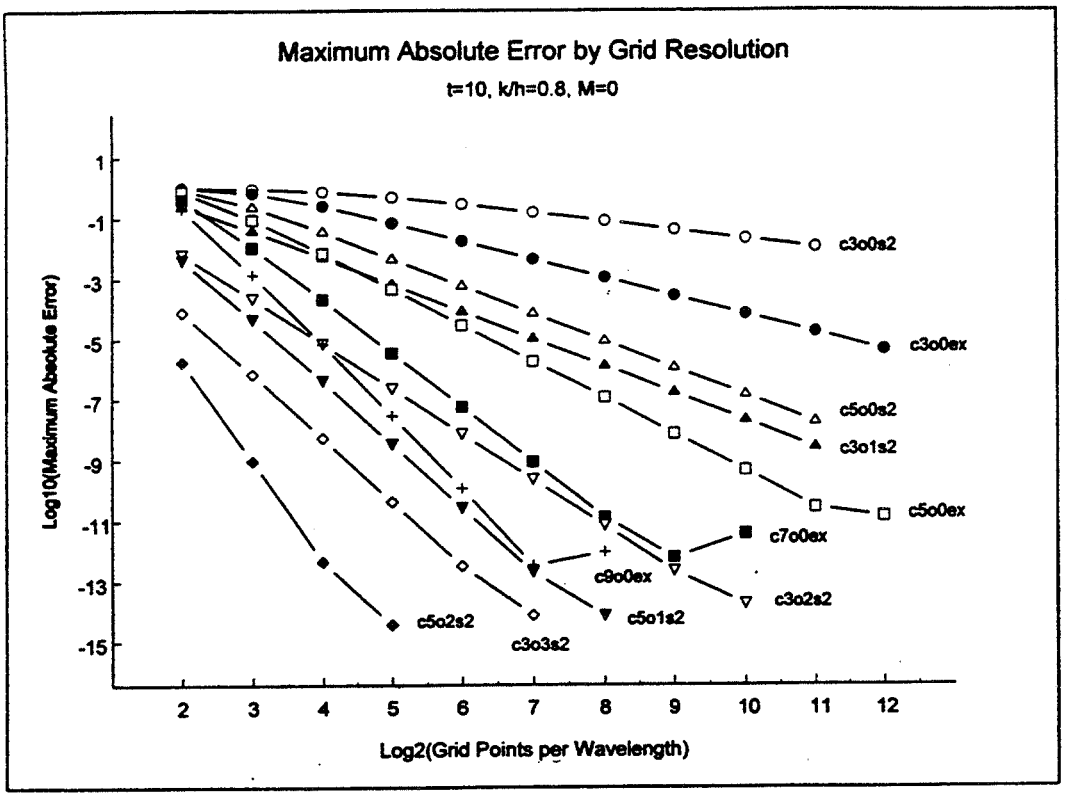


Figure 1: Maximum Absolute Error by Grid Points per Wavelength: 1D

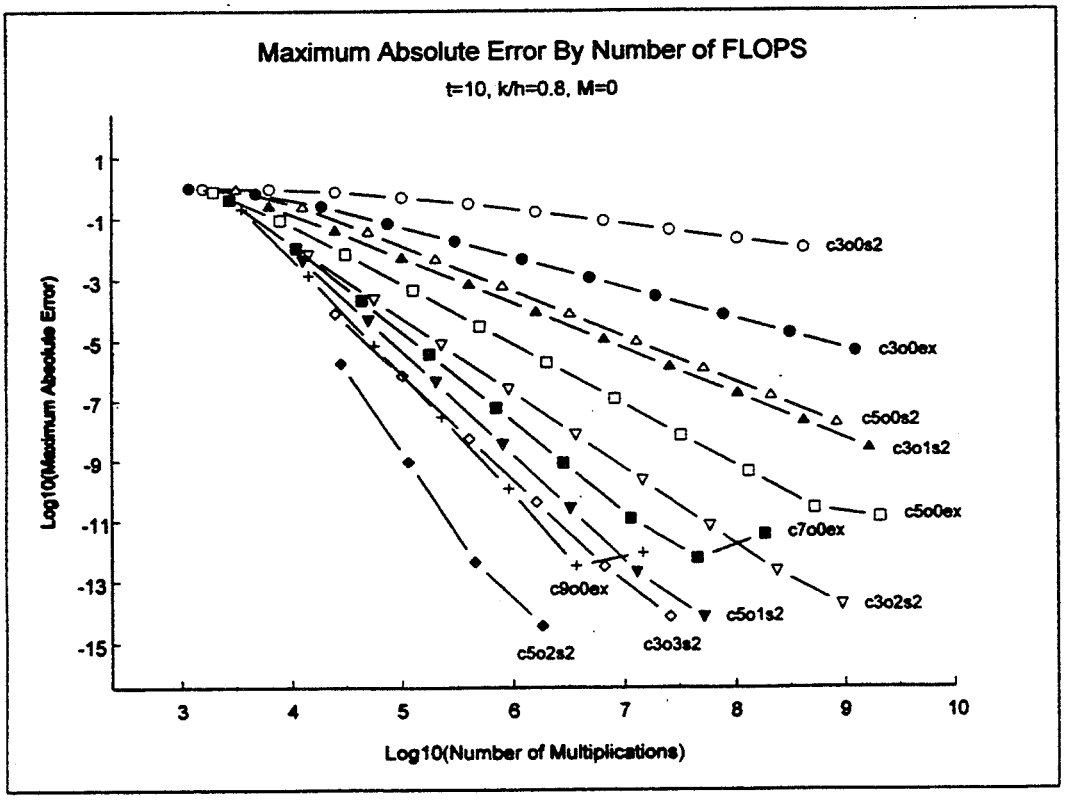


Figure 2: Maximum Absolute Error by Total FLOPS: 1D

Performance Analysis of Large-Scale Applications Based on Wavefront Algorithms

Adolfy Hoisie, Olaf Lubeck, and Harvey Wasserman
<hoisie, oml, hjw> @lanl.gov

Scientific Computing Group, Los Alamos National Laboratory
Los Alamos, NM 87545
505-667-5216

1. Introduction

Wavefront techniques are used to enable parallelism in algorithms that have recurrences by breaking the computation into segments and pipelining the segments through multiple processors [1]. First described as "hyperplane" methods by Lamport [2], wavefront methods now find application in several important areas including particle physics simulations [3], parallel iterative solvers [4], and parallel solution of triangular systems of linear equations [5-7].

Wavefront computations present interesting implementation and performance modeling challenges on distributed memory machines because they exhibit a subtle balance between processor utilization and communication cost. Optimal task granularity is a function of machine parameters such as raw computational speed, and inter-processor communication latency and bandwidth. Although it is simple to model the computation-only portion of a single wavefront, it is considerably more complicated to model multiple wavefronts existing simultaneously, due to potential overlap of computation and communication and/or overlap of different communication or computation operations individually. Moreover, specific message passing synchronization methods impose constraints that can further limit the available parallelism in the algorithm. A realistic scalability analysis must take into consideration these constraints.

Much of the previous parallel performance modeling of software-pipelined applications has involved algorithms with one-dimensional recurrences and/or one-dimensional processor decompositions [5-7]. A key contribution of this paper is the development of an analytic performance model of wavefront algorithms that have recurrences in multiple dimensions and that have been partitioned and pipelined on multidimensional processor grids. We use a "compact application" called SWEEP3D, a time-independent, Cartesian-grid, single-group, "discrete ordinates" deterministic particle transport code taken from the DOE Accelerated Strategic Computing Initiative (ASCI) workload. Estimates are that deterministic particle transport accounts for 50-80% of the execution time of many realistic simulations on current DOE systems; this percentage may expand on future 100-TFLOPS systems. Thus, an equally-important contribution of this work is the use of our model to explore SWEEP3D scalability and to show the sensitivity of SWEEP3D to per-processor sustained speed, and MPI latency and bandwidth on future-generation systems.

Efforts devoted to improving performance of discrete ordinates particle transport codes extended recently to massively-parallel systems [8-12]. Research has included models of performance as a function of problem and machine size, as well as other characteristics of both the simulation and the computer system under study. In [3] a parallel efficiency formula that considered computation only is presented, while [9] a model specific to CRAY T3D communication is developed. These previous models had limiting assumptions about the computation and/or the target machines.

2. Description of Discrete Ordinates Transport

Although much more complete treatments of discrete ordinates neutron transport have appeared elsewhere [12-15], we include a brief explanation here to make clear the origin of the wavefront process in SWEEP3D. The basis for neutron transport simulation is the time-independent, multigroup, inhomogeneous Boltzmann transport equation, in which the unknown quantity is the flux of particles at the spatial point **r** with energy E traveling in direction **Ω**.

Numerical solution involves complete discretization of the multi-dimensional phase space defined by **r**, **Ω**, and E. Discretization of energy uses a "multigroup" treatment, in which the energy domain is partitioned into subintervals in which the dependence on energy is known. In the discrete ordinates approximation, the angular-direction **Ω** is discretized into a set a quadrature points. This is also referred to as the S_N method, where (in 1D) N represents the number of angular ordinates used. The discretization is completed by differencing the spatial domain of the problem on to a grid of cells.

The numerical solution to the transport equation involves an iterative procedure called a "source iteration" [13]. The most time-consuming portion is the "source correction scheme," which involves a transport sweep through the entire grid-angle space in the direction of particle travel. In Cartesian geometries, each octant of angles has a different sweep direction through the mesh, and all angles in a given octant sweep the same way.

Each interior cell requires in advance the solution of its three upstream neighboring cells – a three-dimensional recursion. This is illustrated in Figure 1 for a 1-D arrangement of cells and in Figure 2 for a 2-D grid.

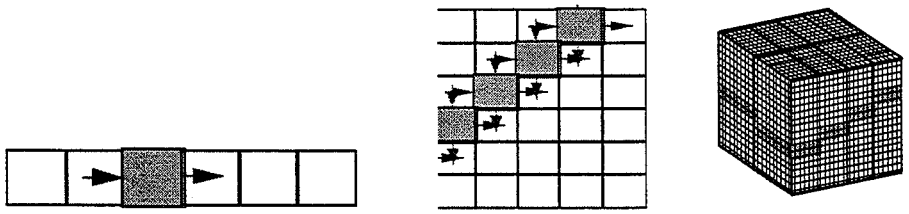


Figure 1. Dependences for a 1-D and 2-D (diagonal) Transport Sweep (left, middle). Illustration of the 2-D Domain decomposition on eight processors with 2 k-planes per block. The transport sweep has started at top of the processor in the foreground. Concurrently-computed cells are shaded. (right)

3. Parallelism in Discrete Ordinates Transport

The only inherent parallelism is related to the discretization over angles. However, reflective boundary conditions limit this parallelism to, at most, angles within a single octant. The two-dimensional recurrence may be partially eliminated because solutions for cells within a diagonal are independent of each other (as shown in Figure 1).

Diagonal concurrency can also be the basis for implementation of a transport sweep using a decomposition of the mesh into subdomains using message passing to communicate the boundaries [12], shown in Figure 1. The transport sweep is performed subdomain by subdomain in a given angular direction. Each processor's exterior surfaces are computed by, and received in a message from, "up-

stream" processors owning the subdomains sharing these surfaces.

However, as pointed out in [9] and [3], the dimensionality of the S_N parallelism is always one order lower than the spatial dimensionality.

Parallel efficiency would be limited if each processor computed its entire local domain before communicating information to its neighbors. A strategy in which blocks of planes in one direction (k) and angles are pipelined through this 2-D processor array improves the efficiency, as shown in Figure 1. Varying the block sizes changes the balance between parallel utilization and communication time.

4. A Performance Model for Parallel Wavefronts

This section describes a performance model of a message passing implementation of SWEEP3D. Our model uses a pipelined wavefront as the basic abstraction and predicts the execution time of the transport sweep as a function of primary computation and communication parameters. We use a two-parameter (latency/bandwidth) linear model for communication performance, which is equivalent to the LogGP model [16]. The demonstrations are sketched only due to space limitations in this abstract.

4.1 Pipelined Wavefront Abstraction

An abstraction of the SWEEP3D algorithm partitioned for message passing on a 2-D processor domain (ij plane) is described in Figure 2. The inner-loop body of this algorithm describes a wavefront calculation with recurrences in two dimensions. Multiple waves initiated by the octant, angle-block and k- block loops are pipelined one after another as shown in Figure 3, in which two inner loop bodies (or "sweeps") are executing on a *Px* by *Py* processor grid. Using this abstraction, we can build a model of execution time for the transport sweep. The number of steps required to execute a computation of N_{sweep} wavefronts, each with a pipeline length of N_s stages and a repetition delay of d is given by equation (1).

$$Steps = N_s + d(N_{sweep} - 1), \qquad (1)$$

The pipeline consists of both computation and communication stages. The number of stages of each kind and the repetition delay per wavefront need to be determined as a function of the number of processors and shape of the processor grid.

```
FOR EACH OCTANT DO
  FOR EACH ANGLE-BLOCK IN OCTANT DO
          FOR EACH K-BLOCK DO
                  IF (NEIGHBOR_ON_EAST) RECEIVE FROM EAST (BOUNDARY DATA)
                  IF (NEIGHBOR_ON _NORTH) RECEIVE FROM NORTH (BOUNDARY DATA)
                        COMPUTE_MESH (EVERY I,J DIAGONAL; EVERY K IN K-BLOCK;
                  EVERY ANGLE IN ANGLE-BLOCK)
                  IF (NEIGHBOR_ON_WEST) SEND TO WEST(BOUNDARY DATA)
                  IF (NEIGHBOR_ON_SOUTH) SEND TO SOUTH(BOUNDARY DATA)
          END FOR
  END FOR
END FOR
```

Figure 2. Pseudo Code for the wavefront Algorithm

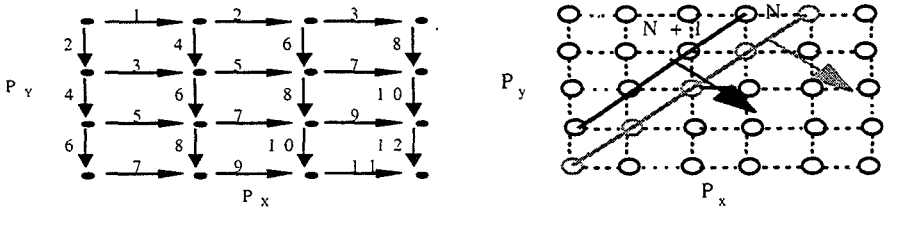


Figure 3. Communication (left) and Computation Pipelines.

4.2 Computation Stages

Figure 3 shows that the number of computation stages is simply the number of diagonals in the grid. A different number of processors is employed at each stage but all stages take the same amount of time since processors on a diagonal are executing concurrently. Equation (2) gives the number of computation steps in the pipeline,

$$N_s^{comp} = P_x + P_y - 1 \quad (2) \qquad T_{cpu} = \left(\frac{N_x}{P_x} + \frac{N_y}{P_y} + \frac{N_z}{K_b} + \frac{N_a}{A_b}\right)\frac{N_{flops}}{R_{flops}} \quad (3)$$

and Equation 3 gives the cost of each step with N_x, N_y, and N_z being the number of grid points in each direction; K_b is the size of the k-plane block; A_b is the size of the angular block; N_{flops} is the number of floating-point operations per gridpoint; and R_{flops} is a characteristic floating-point rate for the processor. The next sweep can begin as soon as the first processor completes its computation so the repetition delay, d^{comp}, is 1 computational step (i.e., the time for completing one diagonal in the sweep).

4.3 Communication Stages

In Figure 6 edges labeled with the same number are executed simultaneously and the graph shows that it takes 12 steps to complete one communication sweep on a 4 x 4 processor grid. One can generalize the number of stages to a grid of P_x by P_y :

$$N_s^{comm} = 2(P_y - 1) + 2(P_x - 1) \quad (4) \qquad T_{msg} = t_0 + \frac{N_{msg}}{B} \quad (5)$$

The cost of any single communication stage is the time of a one-way, nearest neighbor communication given (5). Latency (t_0) and bandwidth (B), are defined above.

The repetition delay for the communication pipeline, d^{comm}, is 4 because a message sent from the top-left processor (processor 0) to its east neighbor (processor 1) on the second sweep cannot be initiated until processor 1 completes its communication with its south neighbor from the first sweep (Figure 3).

4.4 Combining Computation and Communication Stages

We can summarize the discussion so far in two equations that give the separate contributions of computation and communication:

$$T^{comp} = [(P_x + P_y - 1) + (N_{sweep} - 1)] * T_{cpu} \quad (6) \qquad T^{comm} = [2(P_x + P_y - 2) + 4(N_{sweep} - 1)] * T_{msg} \quad (7)$$

The major remaining question is whether the separate contributions, T^{comp} and T^{comm}, can be summed to derive the total time. We have demonstrated that the total time is the sum of eqns. (6) and (7), where T_{cpu} is given by eqn. (3) and T_{msg} is given by eqn. (5). The proof is not presented in this abstract.

5. Validation of the Model

The model was validated with performance data from SWEEP3D on three different machines (SGI Origin 2000, IBM SP2 and Cray T3E), with up to 500 processors, over the entire range of the various model parameters. Inspection of eqns. (6) and (7) leads to identification of the following validation regimes:

$N_{sweep} = 1$: This case validates the number of pipeline stages in T^{comp} and T^{comm}, as functions of $(P_x + P_y)$.

$N_{sweep} \sim (P_x + P_y)$: Validation of a case where the contributions of the $(P_x + P_y)$and N_{sweep} are comparable.

$N_{sweep} >> (P_x+P_y)$: This case validates the repetition rate of the pipeline.

For each of these three cases, we analyze problem sizes chosen in such a way as to make:

$T^{comp} >> T^{comm}$; (validate eqn. (6) only). $T^{comp} = 0$; (validate eqn. (7) only). $T^{comp} \sim T^{comm}$; (validate the sum of eqns. (6) and (7)).

The agreement of the model with the measured data is very good in all cases (not presented here).

6. Applications of the Model. Scalability Predictions.

ASCI is targeting a 100-TFLOPS system in the year 2004, with a workload defined by specific engineering needs. In this section we apply our model to predict the machine parameters under which the runtime goal might be met. We assume a 100-Tflops-peak system composed of about 20,000 processors (based on an extrapolation of Moore's law).

Three sources of difficulty with such a prognosis are (1) making reasonable estimates of machine performance parameters for future systems; (2) managing the SWEEP3D parameter space (i.e., block sizes); and (3) estimating what problem sizes will be important. We handle the first by studying a range of values covering both conservative and optimistic changes in technology. We handle the second by reporting results that correspond to the shortest execution time (i.e., we use block sizes that minimize runtime). We handle the third as follows. For particle transport, one ASCI target problem involves $O(10^9)$ mesh points, 30 energy groups, $O(10^4)$ time steps, and a runtime goal of about 30 hours. With 5,000 unknowns per grid point, this requires about 40 TBytes total memory. On 20,000 processors the resulting subgrid size is approximately 6 x 6 x 1000. In a different ASCI scenario, particle transport problem size is determined by external factors. Based on [17], such computations will involve smaller grid sizes (20 million cells) on the full machine. The 20 million-cell problem would utilize a 2 x 2 x 250 subgrid.

6.1. The 1 billion-cell problem

Plots showing dependence of runtime with sustained processor speed and MPI latency are shown in Figure 4 for several k-plane block sizes, using optimal values for the angle-block size. Table 1 collects some of the modeled runtime data for a few important points: sustained processor speeds of 10% and 50% of peak, and MPI latencies of 0.1, 1, and 10 microseconds. Our model shows that the dependence on bandwidth is small, and as such no sensitivity plot based on ranges for bandwidth is presented. All results assume 400 Mbytes/s MPI bandwidth [18].

We note that runtime under the most optimistic technological estimates in Table 1 is larger than the 30-hour goal by a factor of two. The runtime goal could be met if, with these values of processor speed and MPI latency, we used what we believe to be an unrealistically high bandwidth value of 4 GBytes/s.

Assuming a more realistic sustained processor speed of 10% of peak, Table 1 shows that we miss the goal by about a factor of six even when using 0.1 μs MPI latency. With the same assumption for processor speed, but with a more conservative value for latency (1 μs), the model predicts that we are a factor of 6.6 off. Our results show that the best way to decrease runtime is to achieve better sustained processor performance. This is a result of the relatively low communication/computation ratio that our model predicts. For example, using values of 1 μs and 400 MB/sec for the MPI latency and bandwidth, and a sustained processor speed of 0.5 GFLOPS, the communication time will only be 20% of the total runtime.

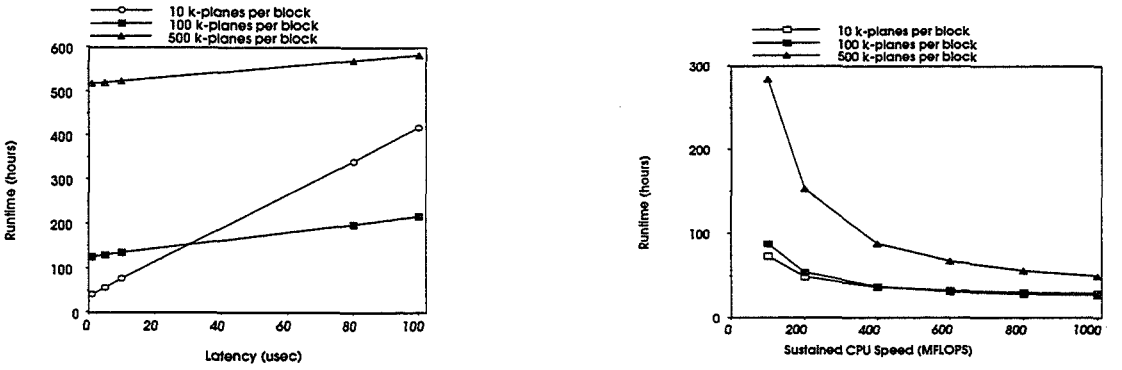


Figure 4. Left: Model-projected sensitivity of the billion-cell problem to MPI latency on a 100-Tflops system. Sustained CPU speed = 500 Mflops, B = 400 Mbytes/s. Right: Model-projected sensitivity of the billion-cell runtime to sustained processor speed on a hypothetical 100-Tflops. Latency=15us, B=400 MB/s.

Table 1. Estimates of SWEEP3D Performance on a Future-Generation System as a Function of MPI Latency and Sustained Per-Processor Computing Rate

	10% of Peak		50% of Peak	
MPI Latency	*Runtime (hours)*	*Amount of Communication*	*Runtime (hours)*	*Amount of Communication*
0.1 μs	180	16%	56	52%
1.0 μs	198	20%	74	54%
10 μs	291	20%	102	58%

6.2. The 20 million-cell problem

The model predicts that communication time ranges from one-half the total time to two-thirds of the total time depending on specific values for the latency and processor speed. The contribution of the bandwidth to the communication cost is, again, negligible. For this problem size latency and processor speed are equally important in decreasing the runtime. Actual plots are not presented here.

7. Summary

We introduced a performance model for parallel, multidimensional, wavefront calculations with machine performance characterized using the LogGP framework. The model accounts for overlap in the communication and computation components. The agreement with experimental data is very good under a variety of model sizes, data partitionings, blocking strategies, and on three different parallel architectures. Using our model, we analyzed performance of a deterministic transport code on a hypothetical 100 Tflops future parallel system of interest to ASCI.

8. Acknowledgements.

We acknowledge the use of resources at the Advanced Computing Laboratory, LANL, and support from the U.S. DOE under Contract No. W-7405-ENG-36. We thank SGI/CRAY for a grant of computer time on the CRAY T3E system. We acknowledge the use of the IBM SP2 at the LLNL.

9. References.

Please contact any of the authors for the complete reference list.

Virtual Petaflop Simulation: Parallel Potential Solvers and New Integrators for Gravitational Systems *

George Lake[‡†] Thomas Quinn[‡] Derek C. Richardson[‡] Joachim Stadel[‡]

Abstract

The orbit of any one planet depends on the combined motion of all the planets, not to mention the actions of all these on each other. To consider simultaneously all these causes of motion and to define these motions by exact laws allowing of convenient calculation exceeds, unless I am mistaken, the forces of the entire human intellect.
—Isaac Newton 1687

Epochal surveys are throwing down the gauntlet for cosmological simulation. We describe three keys to meeting the challenge of N-body simulation: adaptive potential solvers, adaptive integrators and volume renormalization. With these techniques and a dedicated Teraflop facility, simulation can stay even with observation of the Universe.

We also describe some problems in the formation and stability of planetary systems. Here, the challenge is to perform accurate integrations that retain Hamiltonian properties for 10^{13} timesteps.

1 COSMOLOGICAL N-BODY SIMULATION

Simulations are required to calculate the nonlinear final states of theories of structure formation as well as to design and analyze observational programs. Galaxies have six coordinates of velocity and position, but observations determine just two coordinates of position and the line-of-sight velocity that bundles the expansion of the Universe (the distance via Hubble's Law) together with random velocities created by the mass concentrations (see Figure 1). To determine the underlying structure and masses, we must use simulations. If we want to determine the structure of a cluster of galaxies, how large must the survey volume be? Without using simulations to define observing programs, the scarce resource of observing time on $2Billion space observatories may be mispent. Finally, to test theories for the formation of structure, we must simulate the nonlinear evolution to the present epoch.

This relationship to observational surveys defines our goal for the next decade. The Sloan Digital Sky Survey (SDSS) (Gunn and Knapp 1992) will produce fluxes and sky positions for 5×10^7 galaxies with redshifts for the brightest 10^6. Our ambitious observational

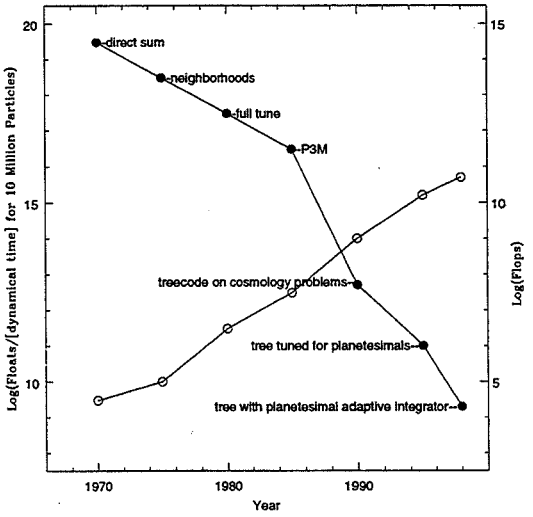


Figure 1: Gains in hardware and algorithms are compared for the N-body simulations. Algorithms are shown as filled points with the scale to the left, while hardware is open points with the sclale on the right. The final algorithmic point should be considered a hopeful projection for 1999.

colleagues have cut steel and ground glass to survey a "fair volume" that *we must simulate*, but we need $N = 10^{12}$ to do this. Direct summation of the gravitational forces using fixed timesteps would take 10^{10} Teraflop-years.

We will explain why this is a unique time to survey the Universe as well as describing the technical breakthroughs required to create a better survey of the cosmos. We will then present the three keys to a realistic float count: 1) spatially adaptive potential solvers, 2) temporally adaptive integrators and 3) volume renormalizations. Another goal of this paper is to define "high quality simulations" and the niche science that can be done with $N \sim 10^8$.

2 THE PROGRESS OF SIMULATIONS

Over the last 20 years, the N of our simulations has increased as: $log_{10}N = 0.3 * (Year - 1973)$. Figure 1 shows the relative contributions of hardward and algorithms. We can't wait to simulate 10^{12} particles,

*Support by NASA HPCC/ESS, IRP and ATP
†NASA HPCC/ESS Project Scientist
‡Department of Astronomy, University of Washington

we have to invent the algorithms that are a thousand times faster! The power of computers has doubled every 8 months (open circles, log scale to the right) with algorithmic advances keeping the same pace (closed circles, log scale to the left). Together, the doubling time in power is 8 months, accumulating to a trillion-fold increase in less than 3 decades. We can't wait to simulate 10^{12} particles, we have to invent the algorithms that are a thousand times faster!

There are two constraints on our choice of N. The cost of computing a full cosmological simulation is $\sim 10^{5.7}N^{4/3}$ floats (the scaling with $N^{4/3}$ arises from the increased time resolution needed as inter-particle separation decreases). The memory needed to run a simulation is $\sim 10^2 N$ bytes. If we fix N by filling memory, the time to run a simulation is 10 days $\times$ (bytes/flop rate$(N/30Million)^{1/3}$). Current machines are well balanced for our Grand Challenge simulations. With Gigaflops and Gigabytes, we can perform simulations with $N \sim 10^{7.5}$. With Teraflops and Terabytes, we can simulate 10^{10} particles. Simulations with $N \sim 10^{12}$ lie in the nether world of Petaflops and Petabytes.

There are a variety of problems where $N \sim 10^6$ represents a minimum ante. For example, clusters of galaxies are extremely important for determining cosmological parameters such as the density of the Universe. Within a cluster, the galaxies are 1-10% of the mass, and there are roughly 10^3 of them. If the galaxies have fewer than 10^3 particles, they dissolve before the present epoch owing to two-body relaxation in the tidal field of the cluster. To prevent this, we need $N > 10^7$ per cluster. Scaling to the Sloan Volume yields $N \sim 10^{12}$.

There are $\sim 10^{20}$ solar masses within the SDSS volume, so even 10^{12} is a paltry number as each particle would represent 10^8 solar masses. We nead a ten-fold more to represent the internal structure of galaxies. N will always be far smaller than the true number of particles in the Universe and will compromise the physics of the system at some level. We can only make sure that: 1) the physics being examined has not been compromised by discreteness effects owing to N-deprivation and 2) gravitational softening, discrete timesteps, force accuracy and simulation volume don't make matters even worse. N is not the figure of merit in most reported simulations—it should be! *The N-body Constitution* (Lake *et al.* 1995) provides a set of *necessary but not sufficient* guidelines for N-body simulation.

The main physical effect of discreteness is the energy exchange that results from two body collisions. Gravity has a negative specific heat owing to the negative total energy (sum of gravitional binding and kinetic energy) of a bound ensemble, like a star cluster. As a star cluster evolves, stars are scattered out by collisions leaving with positive energy. The remaining stars remaining have greater negative energies, the cluster shrinks, the gravitational binding energy increases and the stars move faster. In galaxies and clusters of galaxies, the timescale for this to occur is 10^3 to 10^6 times the age of the Universe. In many simulations, the combination of discreteness in mass, time and force evaluation can make the timescale much shorter leading to grossly unphysical results. So, we must use N sufficient that physical heating mechanisms dominate over numerical or the numerical heating timescale is much longer than the time we simulate. We inventoried all the physical heating mechanisms experienced by galaxies in clusters and discovered a unique new phenomena we call "galaxy harassment" .

3 PARALLEL SPATIALLY/TEMPORALLY ADAPTIVE N-BODY SOLVERS WITH "VOLUME RENORMALIZATION"

Performance gains of the recent past and near future rely on parallel computers that reduce CPU–years to wall-clock–days. The challenge lies in dividing work amongst the processors while minimizing the latency of communication.

The dynamic range in densities demands that spatially and temporally adaptive methods be used. Our group has forsaken adaptive mesh codes to concentrate on tree–codes (Barnes and Hut 1986) that can be made fully spatially and temporally adaptive. The latter use multipole expansions to approximate the gravitational acceleration on each particle. A tree is built with each node storing its multipole moments. Each node is recursively divided into smaller subvolumes until the final leaf nodes are reached. Starting from the root node and moving level by level toward the leaves of the tree, we obtain a progressively more detailed representation of the underlying mass distribution. In calculating the force on a particle, we can tolerate a cruder representation of the more distant particles leading to an $O(N \log N)$ method. We use a rigorous error criterion to insure accurate forces.

As the number of particles in a cosmological simulation grows, so do the density contrasts and the range of dynamical times ($\propto 1/\sqrt{\text{density}}$). If we take the final state of a simulation and weight the work done on particles inversely with their natural timesteps, we find a potential gain of of ~ 50.

The leapfrog time evolution operator, $D(\tau/2)$

$K(\tau)D(\tau/2)$, is the one most often used:

$$\begin{aligned} \textit{Drift, } D(\tau/2); \quad \underline{\mathrm{r}}_{n+1/2} &= \underline{\mathrm{r}}_n + \frac{1}{2}\tau\underline{\mathrm{v}}_n, \\ \textit{Kick, } K(\tau); \quad \underline{\mathrm{v}}_{n+1} &= \underline{\mathrm{v}}_n + \tau\underline{\mathrm{a}}(\underline{\mathrm{r}}_{n+1/2}), \\ \textit{Drift, } D(\tau/2); \quad \underline{\mathrm{r}}_{n+1} &= \underline{\mathrm{r}}_{n+1/2} + \frac{1}{2}\tau\underline{\mathrm{v}}_{n+1}. \end{aligned}$$

where $\underline{\mathrm{r}}$ is the position vector, $\underline{\mathrm{v}}$ is the velocity, $\underline{\mathrm{a}}$ is the acceleration, and τ is the timestep. This operator evolves the system under the Hamiltonian

$$H_N = H_D + H_K + H_{err} = \frac{1}{2}\underline{\mathrm{v}}^2 + V(\underline{\mathrm{r}}) + H_{err},$$

where H_{err} is of order τ^2 (Saha and Tremaine 1994). The existence of this surrogate Hamiltonian ensures that the leapfrog is symplectic—it is the exact solution of an approximate Hamiltonian. Errors explore the ensemble of systems close to the initial system rather than an ensemble of non-Hamiltonian time evolution operators near the desired one.

Leapfrog is a second-order symplectic integrator requiring only one costly force evaluation per timestep and only one copy of the physical state of the system. These properties are so desirable that we have concentrated on making an adaptive leapfrog. Unfortunately, simply choosing a new timestep for each leapfrog step evolves $(\underline{\mathrm{r}}, \underline{\mathrm{v}}, \tau)$ in a manner that may not be Hamiltonian, hence it is neither symplectic nor time-reversible. The results can be awful (Calvo and Sanz-Serna 1993). Time reversibility can be restored (Hut, Makino and McMillan 1994) if the timestep is determined implicitly from the state of the system at both the beginning and the end of the step. This requires backing up timesteps, throwing away expensive force calculations and using auxiliary storage. However, we can define an operator that "adjusts" the timestep, A, yet retains time reversibility and only calculates a force if it is used to complete the timestep (Quinn *et al.* 1997). This is done by choosing A such that it commutes with K, so that $DAKD$ is equivalent to $DKAD$. Since K only changes the velocities, an A operator that depends entirely on positions satisfies the commutation requirement. The "natural definition" of timestep, $\propto 1/\sqrt{\text{density}}$, is ideal but it is difficult to define when a few particles are close together. Synchronization is maintained by choosing timesteps that are a power-of-two subdivision of the largest timestep, τ_s. That is, $\tau_i = \frac{\tau_s}{2^{n_i}}$, where τ_i is the timestep of a given particle. We are currently experimenting with this approach and encourage others to look at variants.

"Volume Renormalization" uses a large scale simulation with modest resolution to identify regions of particular interest: sites of galaxy/QSO formation, large clusters of galaxies, etc. Next, initial conditions are reconstructed using the same low-frequency waves but adding higher spatial frequencies. We have achieved a resolution of we can achieve 10^3 parsec resolution within a cosmological volume of size 10^8 parsec to study the origin of quasars (Katz *et al.* 1994).

4 SIMULATING THE SLOAN VOLUME

Our proposed program to simulate the Sloan Volume before the millenia is as follows:

- Simulate the entire volume (800 Mpc)^3 with $N = 10^{10}$, each with a mass of $10^{10.5}\mathrm{M}_\odot$.
- "Renormalize" dozens of groups, clusters, etc. and simulate with 10^8–10^9 particles.

The total cost for the first simulation is roughly a Teraflop-year and requires a machine with a Terabyte of memory. The second sequence of simulations should be designed to have roughly equal computational cost, but will require less memory.

5 THE FATE OF THE SOLAR SYSTEM

Advances in hardware and numerical methods finally enable us to integrate the solar system for its lifetime. Such an integration is a 1,000 fold advance on the best longest accurate integration ever performed (Laskar, Quinn and Tremaine 1992) and can address numerous questions:

Is the Solar System stable? Do all the planets remain approximately in their current orbits over the lifetime of the Solar System, or are there drastic changes, or perhaps even an ejection of a planet?

What is the affect of orbital changes on the planetary climates? According to the Milankovich hypothesis, climate variations on the Earth are caused by insolation changes arising from slow oscillations in the Earth's orbital elements and the direction of the Earth's spin (Berger *et al.* 1984). Remarkably, the geophysical data (primarily the volume of water locked up in ice as determined by the $^{18}O/^{16}O$ ratio in seabed cores) covers a longer time than any accurate emphemeris.

How does weak chaos alter the evolution of the Solar System? Why does the solar system appear stable if its Lyapunov time is so short?

What is the stability of other planetary system? How are the giant planets related to terrestrial planets in the "inhabitable zone" between boiling and freezing of water by the central star? Without such a cleansing of planetesimals from the solar system by giant planets (Duncan and Quinn 1993), the bombardment of the Earth by asteroids would be steady and frequent throughout the main sequence lifetime of the Sun (Wetherill 1994). The chaos produced by the Jupiter and Saturn may have played a role in insuring that plan-

etesimals collided to form the terrestrial planets [1], but too much chaos will eject planets in the habitable zone. While a search for giant planets is the only technically feasible one today, it may be the ideal way to screen systems before searching for terrestrial planets.

6 INTEGRATING NINE PLANETS FOR 10^{11} DYNAMICAL TIMES

When Laplace expanded the mutual perturbations of the planets to first order in their masses, inclinations and eccentricities, he found that the orbits could be expressed as a sum of periodic terms—implying stability. Poincaré (1892) showed that these expansions don't converge owing to resonances. Using the KAM theorem, Arnold (1961) derived contraints on planet masses, eccentricities, and inclinations sufficient to insure stability. The solar system does not meet his stringent conditions, but this does not imply that it is unstable.

Laskar (1989) tested the quasi-periodic hypothesis by numerically integrating the perturbations calculated to second order in mass and fifth order in eccentricities and inclinations, ~150,000 polynomial terms. Fourier analysis of his 200 million year integration reveals that the solution is not a sum of periodic terms and implyies an instability that is surprisingly short, just 5 Myr.

The second method for attacking the stability problem is to explicitly integrate the planets' orbits (Table 1). As early as 1965, Pluto's behaviour was suspicious. In the last ten years, it has become clear that the solar system is chaotic. However, the source of the chaos is unclear as the system of resonances is complex and the the Lyapunov exponent appear to be sensitive to fine details of initial conditions.

Nonetheless, the Solar System is almost certainly chaotic. Laskar (1994) looked at the fate of Mercury and estimates the chance of ejection in the next few billion years approaches 50%. Our belief in the apparent regularity of the solar system may owe to our inability to know that before the last few ejections, there were 10, 11 or even 12 planets a few billion years ago. At the very least, the chaotic motion leads to a horizon of predictability for the detailed motions of the planets. With a divergence timescale of 4-5 Myr time, an error as small as 10^{-10} in the initial conditions will lead to a 100% discrepancy in 100 Myr. Every time that NASA launches a rocket, it can turns winter to spring in a mere 10 Myr.[2].

We have started a 9 Gyr integration—4.5 Gyr into the past when the solar system was formed and 4.5 Gyr into the future when the Sun becomes a red giant. One basic requirement is a computer with fast quad precision to overcome roundoff problems. The IBM Power 2 series is the current machine of choice, evolving the solar system at $\sim 10^9$ times faster than "real time", this is 1–3 orders of magnitude faster than other available cpus. To understand any chaoitic, we will need to see it by an independent means and devise methods to determine its underyling source.

Table 1: Solar System Integration History

Year Ref	Length (Myr)	# Planets	GR?	Earth's Moon?
1951 Eckert...	0.00035	5	no	no
1965 Cohen...	0.12	5	no	no
1973 Cohen...	1.	5	no	no
1986 Applegate...	217.	5	no	no
	3.	8	no	no
1986 Nobili	100.	5	yes	no
1988 Sussman...	845.	5	no	no
1989 Richardson...	2.	9	no	no
1991 Quinn...	6.	9	yes	yes
1992 Sussman...	100.	9	yes	yes
1999 us	10,000	9	yes	yes

A Parallel Method doesn't seem promising since there are only nine planets to distribute among processors. We employ a different form of parallelism—the "time–slice concurrency method" (*TSCM*) (Saha, Stadel and Tremaine 1997). In this method, each processor takes a different time–slice; processor 2's initial conditions are processor 1's final conditions and so on. The trick is to start processor 2 with a good prediction for what processor 1 will eventually output, and iterate to convergence. This is analogous to the waveform relaxation technique used to solve some partial differential equations (Gear 1991). However, Kepler ellipses are a good guess to the orbits for a timescale that is proportional to the ratio of the Sun's mass to Jupiter's. Tests show that it is extremely efficient to iterate to convergence in double precision (typically 14 iterations each costing 10-15% of a quad iteration), then peform just two iterations to get convergence in quad. In this way, the total overhead pf the full 16 iterations can be less than a factor of 4. There are still many algorithmic issues to be addressed.

For long-term integrations, TSCM has been formulated in a way that preserves the Hamiltonian structure

[1] In Ancient Greek, chaos was "the great abyss out of which Gaia flowed".

[2] Don't let this go beyond this room, environmental impact statements are already tough enough! Are the integrations meaningful given this sensitivity to the initial conditions? We investigate Hamiltonian systems that are as close to the solar system as possible. KAM theory tells us that the qualitative behavior of nearby Hamiltonians should be similar. While the exact phasing of winter and spring is uncertain after millions of years, the severity of winter or spring owing to changes in the Earth–Sun distance and the obliquity are predictable.

and exploits the nearness to an exactly soluble system; otherwise errors grow quadratically with time. *TSCM* will enable us to integrate ~ 0.5 Gyr per day on a 512 node SP2—a speed–up over real–time of 10^{11}. This will make it feasible to study the stability of other solar systems. Detailed development and implementation will be much more challenging than for previous methods, and our high quality serial integration will be required for comparison and validation.

Finally, we will use a new technique to gauge the origin of instabilities (the "tangent equation method") (Saha 1997). In the past, it was common to integrate orbits from many slightly different initial conditions. While that works, it is more rigorous and also more economical to integrate the the linearized or tangent equations—the equations for *differences* from nearby orbits. We will integrate the tangent equations along with the main orbit equations.

7 COSMOLOGY MEETS COSMOGONY: PLANETARY SYSTEM FORMATION

Theories of Solar System formation are traditionally divided into four stages (Lissauer 1993): collapse of the local cloud into a protostellar core and a flattened rotating disk (Nebular Hypothesis); sedimentation of grains from the cooling nebular disk to form condensation sites for planetesimals; growth of planetesimals through binary collision and mutual gravitational interaction to form protoplanets (Planetesimal Hypothesis); and the final assembly to planets with the remaining disk cleansed by ejections from chaotic zones.

Our cosmology code is ideal for the third stage of Solar System formation, particularly in the inner regions where gas was not a primary component and gravitational interactions dominated the evolution. The first stage entails magnetohydrodynamics, the complicated small-particle physics and gas dynamics of the second stage is still not well understood, and the fourth is the purview of long-term stability codes.

All that is required for a detailed simulation of the third stage is a model of the collisional physics and a code capable of dealing with a large number of particles. However, previous direct simulations of the planetesimal stage (summarized in Table 1) fall far short of capturing the full dynamic range of the problem. Our cosmology code has the potential to treat as many as 10^7 particles simulataneously for 10^7 dynamical times, a ten-million-fold improvement that makes us enthusiastic! Only statistical methods (Wetherill and Stewart 1989) employing prescriptions for the outcomes of gravitational encounters have been used to peek at this regime.

We reach an important threshold at $N \sim 10^7$ in our ability to follow planetesimal evolution. At early times, the relative velocities between planetesimals are small and inelastic physical collisions lead to "runaway" growth of planetary embryos (Beaugé and Aarseth 1990). Eventually gravitational scattering increases the planetesimal eccentricities to such an extent that collisions result in fragmentation, not growth. The embryos will continue to grow owing to their large mass, but at a slower rate as their "feeding zones" are depleted (Ida and Makino 1993). The total mass of our planetary system is $448 M_\oplus$ or $3.6 \times 10^4 M_{lunar}$, while the inner planetismal disk amenable to simulation had a mass $\sim 10^2 M_{lunar}$. To capture both growth and fragmentation (Wetherill and Stewart 1989) requires a minimum particle mass of $10^{-5} M_{lunar}$, leading to our target $N \sim 10^7$.

ref	N	t (yr)	Δa (AU)	Col?	G?
Lecar....	200	6×10^4	0.5–1.5	a	n
Beaugé...	200	6×10^5	0.6–1.6	abcf	n
Ida...	800	5000	0.3	–	n
Aarseth...	400	1.2×10^4	0.04	ab	n
Chambers...	100	10^8	0.5–2.0	a	y
Kokubo...	5000	2×10^4	0.4	a	n
Richardson...	10^5	10^3	1.2–3.6	a	y
goal	10^7	10^7	0.5–2.0	abcf	y

Table 1: Highlights of advances in direct simulations of the formation of the inner planets. N is the maximum number of planetesimals used in the simulation, t is the longest integration time, and Δa is either the width of the simulation region at 1 AU or the actual range in orbital distance. If collisions are included in the simulations, details are noted by: a=agglomeration; b=bouncing; c=cratering; f=fragmentation. The final column shows whether perturbations from one or more giant planets are included.

A detailed direct simulation of planet formation can address a variety of important questions, including: Was there runaway growth of a few embryos, or a continuously evolving homogeneous mass distribution? How does the primordial surface density alter the evolution? What fixes the spin orientation and period of the planets—uniform spin-up from planetesimal accretion (Lissauer and Safronov 1991), or a stochastic process dominated by the very last giant collisions (Dones and Tremaine 1993)? Is it feasible that the Earth suffered a giant impact late in its growth that led to the formation of the Moon (Benz *et al.* 1986)? How much radial mixing was there and can it explain observed compositional gradients in the asteroid belt (Gradie, Chapman and Tedesco 1989)? Finally, what is the dominant physical mechanism that drives the late stages of growth—

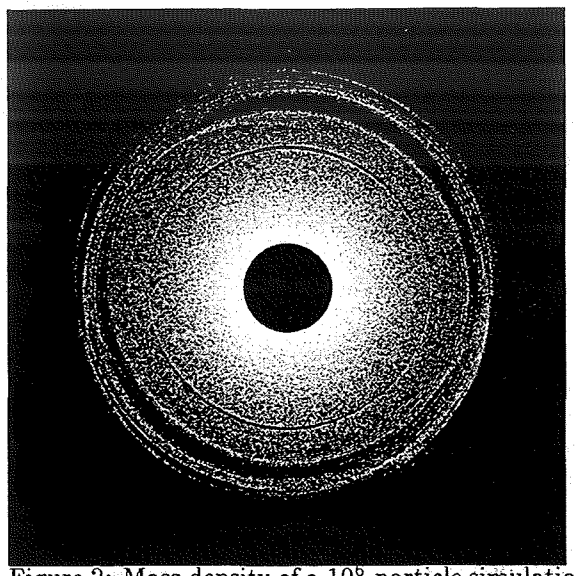


Figure 2: Mass density of a 10^6-particle simulation after 250 yr. Bright shades represent regions of high density. The dot at the top right is Jupiter. The disk extends from 0.8 AU, just inside Earth's present-day orbit, to 3.8 AU, near the outer edge of the asteroid belt. The gaps and spiral structures in the disk are associated with Jupiter mean-motion resonances.

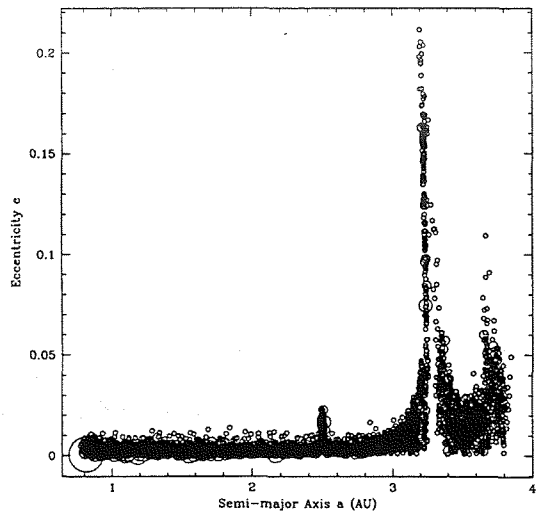


Figure 3: Plot of semi-major axis a *vs.* eccentricity e for the 10^6-particle simulation, showing only every 100^{th} particle to prevent overcrowding. The peaks in e correspond to mean-motion resonances with Jupiter; there are similar features in plots of a *vs.* inclination i (not shown). The circles are scaled by mass for emphasis.

are intrinsic gravitational instabilities between embryos sufficient, or are perturbations by the giant gas planets required? This last point is of key importance to future searches for terrestrial planets. We strongly suspect that the end result of our research may be the assertion that one should concentrate searches for terrestrial planets in those systems that have giant planets.

We have begun to address these issues with a modified version of the cosmology code. Collisions are detected (rather than "softened away") and the outcomes are determined by the impact energy, the lowest energies generally leading to mergers and the highest energies leading to fragmentation (presently merging and bouncing are implemented). Integrations are carried out in the heliocentric frame and may include the giant planets as perturbers. Auxilliary programs are used to generate appropriate initial conditions and to analyze the results of the simulation, but the main work is performed by the modified cosmology code.

Figures 2 and 3 show the mass density and a *vs.* e, respectively, at the end of a 250-yr run that began with 10^6 identical cold planetesimals in a disk from 0.8 to 3.8 AU with surface density proportional to $r^{-3/2}$. The present-day outer planets were included in the calculation. The simulation took 60 hours to finish on a Cray T3E with 128 dedicated processors using a fixed timestep of 0.01 yr. The effect of Jupiter on the disk, which extends well into the present-day asteroid belt, can be seen clearly in the density plot: there is a large density gap at the 2:1 resonance at 3.2 AU and a narrow groove at the 3:1 at 2.5 AU along with spiral wave patterns and other telltale features. There are corresponding features in Fig. 3 which show how Jupiter stirs up planetesimals at the mean-motion resonances. Note that conservation of the Jacobi integral accounts for the slight bending of the e peaks toward smaller a. Meanwhile, planetesimal growth has proceeded unmolested in the inner region of the disk (under the assumption of perfect accretion). The largest planetesimal at the end of the run is 8 times its starting size. As far as we are aware, this is the largest simulation of a self-gravitating planetesimal disk that has ever been attempted.

The figures show however that to get to the regime of runaway growth ($\sim 10^4$–10^5 yr), a new timestepping approach is needed. We are currently developing a technique to exploit the near-Keplerian motion of the planetesimals. For weakly interacting particles, we divide the Hamilonian into a Kepler component, implemented using Gauss' f and g functions, and a perturbation component owing to the force contributions of all the other particles. In this regime, timesteps can be of order the dynamical (*i.e.* orbital) time, resulting in compuational speedups of 10–100. For strongly interacting particles (defined as particles with overlapping Hill spheres), the Hamiltonian is factored into the standard kinetic and

potential energy components, with the central force of the Sun as an external potential. In this regime, particles are advanced in small steps, which allows for the careful determination of collision circumstances. It also allows the detection of collisions in the correct sequence even if a single particle suffers more than one collision during the interval.

The challenge is to predict when particles will change between the regimes of weak and strong interaction. One method we are considering is to construct a new binary tree ordered by perihelion and aphelion. Those particles with orbits separated by less than a Hill sphere are flagged for further testing. This screening has a cost of $N \log N$ and is only performed once per long Kepler step. Flagged pairs of particles with phases that are certain to stay separated over the integration step are reset. The remaining particles are tested by solving Kepler's equation in an elliptical cylindical coordinate system to determine the time of actual Hill sphere overlap. Switching between Hamiltonians is not strictly symplectic, but it occurs infrequently enough for any given particle that it is not a concern. Dissipating collisions are inherently non-symplectic anyway. Once particles separate beyond their Hill spheres (or merge), they are returned to the Kepler drift scheme.

Although much work remains to be done, the reward will be the first self-consistent direct simulation of planetesimals evolving into planets in a realistic disk. The results can be used to study related problems, such as the formation of planetary satellites, orbital migration of giant planets in a sea of planetesimals, and ultimately the ubiquity and diversity of extra-solar planetary systems.

8 Summary: Virtual Petaflops

Past planetesimal simulations used codes with an algorithmic complexity that would be similar to the point labeled "full tune" in Fig. 1 and computers with speeds of $\sim$ 10 Mflops. (Special purpose "GRAPE" hardware of $\sim 10^6$ Mflops has been used, but such implementations involve sums over all interactions so they are closer to the direct sum case in floating point cost [*cf.* Kokubo and Ida 1998].) Our algorithms result in a collective speed-up $\gtrsim 10^8$ for simulations with $N \sim 10^7$ (with rough accounting for the reduction in N over the course of the simulation). It must be emphasized that to attain the desired performance, both hardware *and* algorithm improvements are required. Figure 1 shows that the speed-up factor from algorithms vastly exceeds that of the hardware. It is not sufficient to simply wait for computers to get better, nor does it seem to pay to build special hardware. McMillan *et al.* (1997) asserted that if Grape-6 is built, it could use its Petaflop speed to follow 10^6 planetesimals by the year 2000. They claimed that this would be a seven-year advantage over General Purpose Computers that would only be able to follow 10^4 particles by the year 2000. Our test simulations *without the new integrator* are already 10 years ahead of their projections. We argue that our approach will beat efforts that rely on special purpose hardware with encoded algorithms for at least the next decade.

Sir Isaac would love to see the enhancement of *"the entire human intellect"* by high performance computing.

REFERENCES

Aarseth, S. J.; D. N. C. Lin; and P. L. Palmer. 1993. *Evolution of planetesimals. II. Numerical simulations.* Ap.J., 403, 351–76.

Applegate, J. H.; M. R. Douglas; Y. Gursel; G. J. Sussman and J. Wisdom. 1986. *The outer solar system for 210 million years.* A.J., 92, 176–94.

Arnold, V. I. 1961. *Small Denominators and the Problem of Stability in Classical and Celestial Mechanics.* In Report to the IVth All-Union Math. Congress, Leningrad.

Barnes, J. 1986. *An efficient N-body algorithm for a fine-grain parallel computer.* In The Use of Supercomputers in Stellar Dynamics, P. Hut and S. McMillan, eds., Springer Verlag, New York, 175–80.

Barnes J. and P. Hut. 1986. *A Hierarchical O(NlogN) Force-Calculation Algorithm.* Nature, 324, 446–50.

Beaugé, C. and S. J. Aarseth. 1990. *N-body simulations of planetary formation.* M.N.R.A.S., 245, 30–9.

Benz, W.; W. L. Slattery and A. G. W. Cameron. 1986. *The origin of the Moon and the single-impact hypothesis. I.* Icarus, 66, 515–35.

Berger, A.; J. Imbrie; J. Hayes; G. Kukla and B. Saltzman. 1984. *Milankovitch and climate: understanding the response to astronomical forcing.* Reidel. Dordrecht.

Calvo, M. P. and J. M. Sanz-Serna. 1993. *The development of variable-step symplectic integrators with application to the two-body problems.* SIAM J. Sci. Comput., 14, 936–52.

J. E. Chambers and G. W. Wetherill, *N-body simulations of the formation of the inner planets*, B.A.A.S., 28, (1996), 1107.

Cohen, C. J. and E. C. Hubbard, 1965. *Libration of the Close Approaches of Pluto to Neptune.* A.J., 70, 10–22.

Cohen, C. J.; E. C. Hubbard and C. Oesterwinter. 1973. *Elements of the outer planets for a million years.* Astron. Pap. Amer. Ephem., 22, 1–5.

Dones L. and S. Tremaine. 1993. *On the origin of planetary spins.* Icarus, 103, 67–92.

Duncan M. and T. Quinn. 1993. *The Long-Term Dynamical Evolution of the Solar System.* Ann. Rev. Astr. Ap., 31, 265–89.

Eckert, W. J.; D. Brouwer and G. Clemence. 1951. *Coordinates of the five outer planets 1653–2060.* Astron. Pap. Amer. Ephem., 12, 1–47.

Gear, C. 1991. *Waveform methods for space and time parallelism.* J. Comp. and Appl. Math., 38, 137–47.

Gradie, J. C.; C. R. Chapman and E. F. Tedesco. 1989. *Distribution of taxonomic classes and the compositional structure of the asteroid belt.* In Asteroids II, R. P. Binzel *et al.* eds., Univ. Arizona Press, Tucson, 316-35.

Gunn J. E. and G. R. Knapp. 1992. *The Sloan Digital Sky Survey.* In A.S.P. conf., #43, 267–79.

Hut, P.; J. Makino and S. McMillan. 1994. *Building a better leapfrog.* Ap.J.Lett., 443, L93–6.

Ida, S. and J. Makino, 1992a. *N-body simulation of gravitational interaction between planetesimals and a protoplanet. I. Velocity distribution of planetesimals,* Icarus, 96, pp. 107–20.

Ida, S. and J. Makino, 1992b. *N-body simulation of gravitational interaction between planetesimals and a protoplanet. II. Dynamical friction,* Icarus, 98, pp. 28–37.

Ida S. and J. Makino. 1993. *Scattering of Planetesimals by a Protoplanet: Slowing Down of Runaway Growth.* Icarus, 106, 210–27.

Katz, N.; T. Quinn; E. Bertschinger and J. M. Gelb. 1994. *Formation of quasars at high redshift.* M.N.R.A.S., 270, L71–4.

Kokubo, E. and S. Ida, 1996, *On runaway growth of planetesimals,* Icarus, 123, pp. 180–91.

Kokubo, E. and S. Ida, 1998, *Oligarchic growth of planetesimals,* Icarus, 131, pp. 171–78.

Lake, G.; N. Katz; T. Quinn and J. Stadel. 1995. *"Cosmological N-body Simulation".* Proc 7th SIAM Conf. on Parallel Processing for Sci. Comp., 307–12

Laskar, J. 1989. *A numerical experiment on the chaotic behaviour of the Solar System.* Nature, 338, 237–8.

Laskar, J. 1994 *Large-scale chaos in the solar system.* A.A., 287, L9–12

Lecar, M. and S. J. Aarseth, 1986 *A simulation of the formation of the terrestrial planets,* Ap.J., 305, pp. 564–79.

Lissauer, J. J. 1993*Planet formation.* Ann. Rev. Astr. Ap., 31, 129–74.

Lissauer, J. J. and V. S. Safronov. 1991. *The random component of planetary rotation.* Icarus, 93, 288–97.

Laskar, J.; T. Quinn and S. Tremaine. 1992 *Confirmation of resonant structure in the solar system.* Icarus, 95, 148–52.

McMillan, S., P. Hut, J. Makino, M. Norman, and F. J. Summers. 1997 *Design studies on petaflops special purpose Hardware for particle simulation.* in *Frontiers: 6^{th} Symposium on the Frontiers of Massively Parallel Computing.*

Moore, B.; N. Katz; G. Lake; A. Dressler and A. Oemler. 1996. *Galaxy harassment and the evolution of clusters of galaxies.* Nature, 379, 613–16.

Nobili, A. M. 1986. *LONGSTOP and the masses of Uranus and Neptune.* In Solid Bodies of the Outer Solar System. ESTEC. Noordwijk, 9-13.

Poincaré, H. 1892. *Les methodes nouvelles de la mechanique celeste.* Gauthiers-Villars. Paris.

Quinn, T., N. Katz, J. Stadel and G. Lake. 1997. *Time stepping N-body simulations.* In preparation.

Quinn, T.; S. Tremaine and M. J. Duncan. 1991. *A three million year integration of the Earth's orbit.* A.J., 101, 2287–305.

Richardson, D. L. and C. F. Walker. 1987. *Multivalue integration of the planetary equations over the last one-million years.* Astrodynamics 1987, Soldner et al, eds. Univelt. San Diege. 1473–56.

Richardson, D. L. and C. F. Walker. 1989. *Numerical simulation of the nine-body planetary system spanning two million years.* J. Astronautical Sci., 37, 159–82.

Richardson, D. C., G. Lake, T. Quinn, and J. Stadel, *Direct simulation of planet formation with a million planetesimals: A progress report,* B.A.A.S., 30, (1998), 765.

Saha P. and S. Tremaine. 1992. *Symplectic integrators for solar system dynamics.* A.J., 104, 1633-40.

Saha, P. and S. Tremaine. 1994. *Long-term planetary integration with individual time steps.* A.J., 108, 1962–9.

Saha, P., J. Stadel and S. Tremaine. 1997. *A Parallel Integration Method for Solar System Dynamics.* A.J., 114, 409–15.

Saha, P. 1997. *The Use of Tangent Equations to Detect Chaos in Solar System Dynamics.* in preparation.

Stadel, J. and T. Quinn. 1998. *A generalization of Ewald summation to arbitrary multipole order.* In preparation.

Sussman, G. J. and J. Wisdom. 1988. *Numerical evidence that the motion of Pluto is chaotic.* Science, 241, 433–7.

Sussman, G. J. and J. Wisdom. 1992. *Chaotic Evolution of the Solar System.* Science, 257, 56–62.

Wetherill, G. W. 1994. *Possible consequences of absence of "Jupiters" in planetary systems.* Ap. Sp. Sci., 212, 23–32.

Wetherill G. W. and G. R. Stewart. 1989. *Accumulation of a swarm of small planetesimals.* Icarus, 77, 330–57.

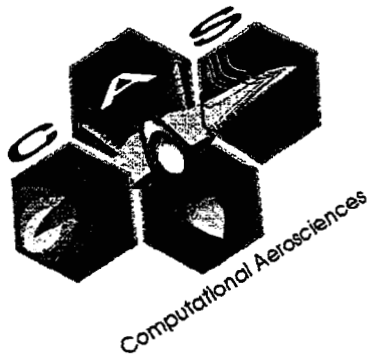
The Kalman Filter and High Performance Computing at NASA's Data Assimilation Office (DAO)

Peter M. Lyster
NASA Data Assimilation Office (DAO)
and University of Maryland Earth System Science Interdisciplinary Center (ESSIC)

Atmospheric data assimilation is a method of combining actual observations with model simulations to produce a more accurate description of the earth system than the observations alone provide. The output of data assimilation, sometimes called ``the analysis", are accurate regular, gridded datasets of observed and unobserved variables. This is used not only for weather forecasting but is becoming increasingly important for climate research. For example, these datasets may be used to assess retrospectively energy budgets or the effects of trace gases such as ozone. This allows researchers to understand processes driving weather and climate, which have important scientific and policy implications. The primary goal of the NASA's Data Assimilation Office (DAO) is to provide datasets for climate research and to support NASA satellite and aircraft missions.

This presentation will: (i) describe ongoing work on the advanced Kalman/Lagrangian filter parallel algorithm for the assimilation of trace gases in the stratosphere, and (ii) discuss the Kalman filter in relation to other presentations from the DAO on Four Dimensional Data Assimilation at this meeting. Although the designation ``Kalman filter" is often used to describe the overarching work, the series of talks will show that the scientific software and the kind of parallelization techniques that are being developed at the DAO are very different depending on the type of problem being considered, the extent to which the problem is mission critical, and the degree of Software Engineering that has to be applied.

A discussion of the scientific and computational complexity of atmospheric data assimilation may be found at http://dao.gsfc.nasa.gov/DAO_people/lys, and for general information on the Data Assimilation Office view http://dao.gsfc.nasa.gov.



Session 2:

Advanced Computer Algorithms and Methodology

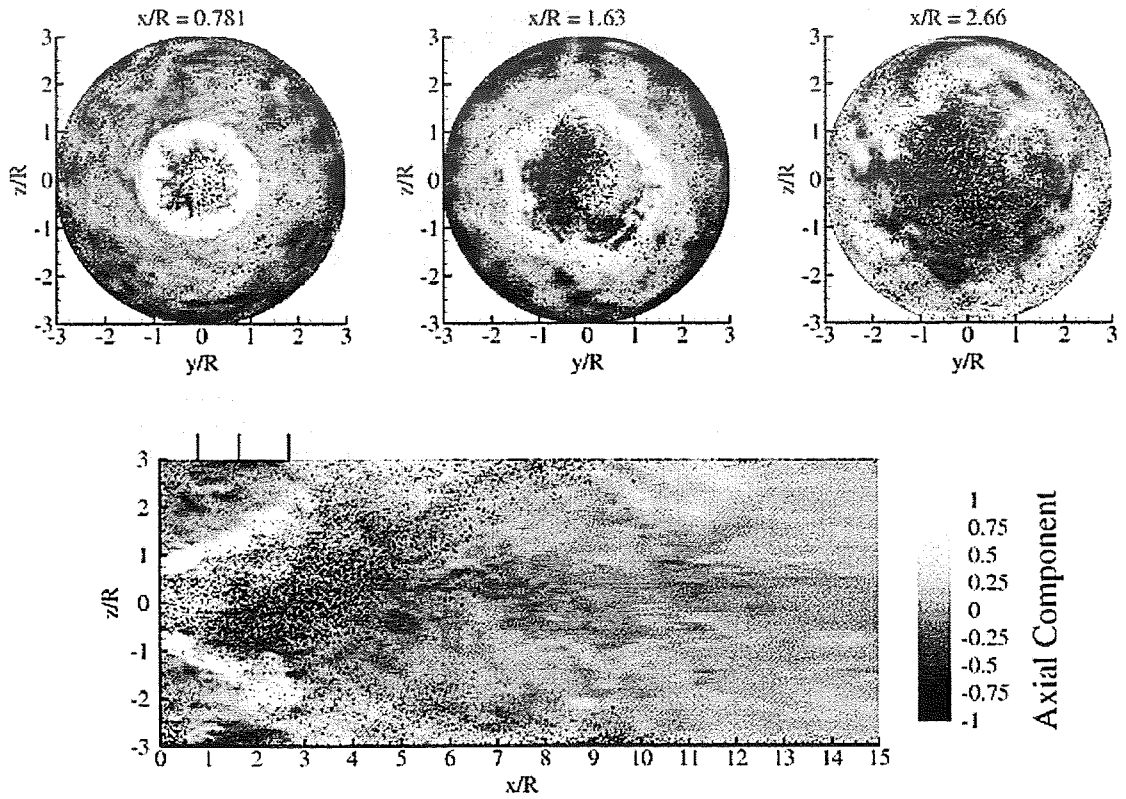
TOWARD THE LARGE EDDY SIMULATION OF GAS TURBINE SPRAY COMBUSTION PROCESSES

Joseph C. Oefelein

Center for Integrated Turbulence Simulations
Department of Mechanical Engineering
Stanford University, Stanford, California 94305-3030
Phone: (650) 723-4220, Fax: (650) 725-3525
Email: jco@stanford.edu

As part of the Department of Energy Accelerated Strategic Computing Initiative (ASCI), an effort is currently underway to develop improved numerical methods and subgrid-scale models suitable for performing highly resolved large-eddy-simulations of multiphase combustion processes in gas turbine combustors. The objective is to provide simulation methodologies that will enable a new paradigm for the design of advanced systems in which turbulent multiphase combustion plays a controlling role. Key components of the program include the development of general theories and models, model validation, and the concurrent development of high-performance parallel algorithms which support the implementation of massively parallel large-scale simulations. Beginning in the point-particle ideal limit, a hierarchy of three fundamental systems are being investigated in swirling coannular dump combustor configurations. These are: 1) particle-laden systems; 2) binary fuel-air systems; and 3) reacting multicomponent systems. The focus of respective investigations are: 1) momentum coupling and subgrid-scale modulation effects; 2) mass and energy coupling and subgrid-scale scalar mixing; and 3) multiphase combustion models. Here, an overview of the objectives, approach, current status, and future directions will be given with emphasis placed on model development, accuracy, and parallel implementation.

A result from work in progress is given in Figure 1. This figure shows a simulation of the Sommerfeld experiment (J. Fluids Eng., Vol. 114, 1992, p. 648) which provides detailed measurements of swirling particle-laden flow in a cylindrical dump combustor configuration. Cross-sections of the instantaneous particle distribution are shown superimposed on the corresponding turbulent velocity field. The primary jet extends a radial distance of 0.5 dimensionless units from the centerline and is laden with glass beads to obtain a mass loading of 0.034. The annular jet extends over a radial interval from 0.59 to 1 dimensionless units and is injected with a swirling azimuthal velocity component to obtain a swirl number of 0.47. The Reynolds number (based on the total volume flow rate and outer radius of the annular jet) is 26200. Results were obtained using the large-eddy-simulation technique and a cylindrical 1.6 million node multiblock grid, with Lagrangian point-particle models employed to handle the dispersed phase. The established results, and those that follow, will serve as benchmarks to validate the accuracy of new and existing models and to assess the performance of improved parallel algorithms. In future studies validated assessments of models used to simulate fully-coupled spray combustion processes will be provided using data acquired from an experimental combustor configuration currently under development at Stanford and similar databases established elsewhere.



Cross-sections of the instantaneous particle distribution superimposed on the corresponding turbulent velocity field in the cylindrical dump combustor configuration employed by Sommerfeld et al. (J. Fluids Eng.}, Vol. 114, 1992, p. 648).

Parallelization of ADI Solver FDL3DI Based on New Formulation of Thomas Algorithm

A.Povitsky

ICASE, NASA Langley Research Center, Hampton, VA 23681-0001

e-mail:aeralpo@icase.edu, phone: (757) 864-4746

M. Visbal

CFD Research Branch, Air Force Research Laboratory, Wright-Patterson AFB, OH 45433-7913

e-mail:visbal@fim.wpafb.af.mil, phone: (937) 255-7127

Introduction. Efficient solution of directionally split banded matrix systems is essential to compact and implicit solvers. When the ADI schemes are applied to multi-dimensional problems, the operators are separated into one-dimensional components and the scheme is split into two (for 2-D problems) or three (for 3-D problems) steps, each one involving only the implicit operations originating from a single coordinate [1]. A direct solver, known as the Thomas algorithm, which is a version of Gauss Elimination method for banded matrix systems, is used for solution of these systems. Parallelization of the Thomas algorithm is hindered by global spatial data dependencies due to the recurrence of data within a loop. Consequently, processors become idle at the switch from the forward to the backward step of the Thomas algorithm and at the beginning of the computations in the next spatial step. Thus, straightforward parallel implementation of the Thomas algorithm [2] is of the pipelined type and is denoted here as the basic pipelined Thomas algorithm (PTA). There are following reasons for its poor parallelization efficiency: (i) there is no completed data for other computational tasks while processors stay idle; (ii) communications control computational tasks as either the forward step coefficients or the backward step solution must be obtained from the neighboring processors for the beginning of the forward or the backward step computations.

In order to avoid pipelining, some parallel Thomas algorithms include the reduction of an $O(N_{tot})$ system of equations on P "slave" processors, the solution of the reduced system of size $O(P)$ on the "master" processor, broadcast this solution to the "slaves", and simultaneous computation of the final solution on P processors. This algorithm includes global communications of "slave-master" type and additional computations on each "slave" processor [3].

Implementation of internal boundary conditions eliminates far-field data dependencies, allowing band matrix systems to be solved independently on each processor. However, modification of either the finite-difference approximation or the implicitness of the scheme due to interface boundary conditions can deteriorate accuracy, stability and convergence properties

relative to the original serial method [4].

We propose a way to reduce parallelization penalty of the basic PTA where the numerical algorithm remains the same as the serial one and only order of computations is changed. To overcome the first problem with the basic PTA (see above), a new pipelined Thomas algorithm has been developed [5]. This algorithm is designed for parallel solution of banded matrix linear systems. We called it the Immediate Backward pipelined Thomas Algorithm (IB-PTA). This algorithm provides exactly the same solution as the serial Thomas algorithm. The advantage of the IB-PTA over the basic PTA is that some lines has been completed by the backward step of the Thomas algorithm before the processors are idle. In non-linear and multi-dimensional problems the IB-PTA may be used for other computational tasks while processors are idle from the Thomas algorithm computations in the current direction.

To overcome the second problem with the basic PTA and to make the IB-PTA feasible, the scheduling algorithm has been developed [5]. The static schedule of computations and communication has been assigned before processors run 3-D ADI solver [6]. The advantage of "control by schedule" over "control by communications" is that processors do not wait to receive necessary data. Instead, processors compute other tasks and switch to receive data only when these data are available on neighboring processors and necessary in a current processor. Thus, the 3-D ADI solver runs on processors in a time-staggered manner without the idle time of global synchronization, i.e., the first processor completes its computational tasks first at each spatial step. In turn, the optimal number of solved lines per message is greater than that for the basic PTA and, consequently, the overall latency time is reduced.

A theoretical model of parallelization efficiency of the 3-D ADI code based on the proposed algorithm is presented in [6]. Our model is based on the idealized multicomputer parallel architecture model [7]. Low-level hardware details such as memory hierarchies and the topology of the interconnection network are not introduced in the model. This model is used to define the optimal number of solved lines per message, to provide asymptotic analysis, to estimate parallelization efficiency for a large number of processors which are not available yet, and to compare the proposed algorithm with the basic one. First, this model is used for a cubic global domain (equal number of grid nodes in all directions). Then an optimal partitioning for a global domain with unequal number of grid nodes in different directions is obtained by this model. To provide a unified approach for various MIMD computers, results are presented in terms of non-dimensional ratios between communication latency and transfer times to the computation time per grid node.

Here we implement this methodology to parallelization of the target code FDL3DI [8], which is based on the ADI method, so as to provide exactly the same solution as the original serial solver and keep a low parallelization penalty.

Methodology. Consider a non-linear partial differential equation

$$\frac{dU}{dt} = S(U)U + Q, \tag{1}$$

where U is solution vector, t is the time, $S(U) = S_x(U)+S_y(U)+S_z(U)$ is a spatial differential

operator and Q is a source term. The high order ADI scheme applied to the above Eq. is solved in three steps as a succession of one-dimensional finite-difference penta-diagonal systems:

$$a_{i,j,k}U_{i-2,j,k} + b_{i,j,k}U_{i-1,j,k} + c_{i,j,k}U_{i,j,k} + d_{i,j,k}U_{i+1,j,k} + e_{i,j,k}U_{i+2,j,k} = f_{i,j,k}, \qquad (2)$$

where $i = 1, ..., N_x$, $j = 1, .., N_y$, $k = 1, ..., N_z$ are spatial grid nodes, coefficients a, b, c, d, e are functions of U^n and/or time. This system of $N_xN_yN_z$ equations is considered as N_yN_z systems of N_x equations, where each system of N_x equations corresponds to $j, k = const$. The above system of linear equations corresponds to the first spatial step; the similar banded linearized systems must be solved for the second and the third spatial steps of the ADI.

The algorithm denoted as the Immediate Backward PTA (IB-PTA) is described here. First, lines are computed by the forward step of the Thomas algorithm till the first portion of lines is completed on the last processor. Then the backward step computations of the Thomas algorithm for each portion of lines start immediately after the completion of the forward step computations for the corresponding lines. Each processor switches between the forward and backward steps of the Thomas algorithm and communicates with its neighbors to get necessary data for beginning of either the forward or backward computations for a next portion of lines. Finally, remaining lines are computed by the backward step computations and there is no available lines for the forward step computations. It is shown in [5], that the idle time of both the IB-PTA and the PTA is equal to $2L + 2(P - 1)$, where L is the number of portions of lines and P is the number of processors. The idle time of the p^{th} processor is equal to $2(P - p)$. The advantage of the IB-PTA over the PTA is that part of lines has been completed by the Thomas algorithm before processors become idle, and the idle processors can perform other data-dependent computational tasks while processors are idle. Additionally, these tasks might be manifold, and the idle processor times are different for the different processors.

The static scheduling of processors is adopted in this study, i.e., the communication and computations schedule of processors is assigned before numerical computations are executed. The recursive scheduling algorithm is presented in [5]. The schedule of processors is stored in two arrays, where the first array contains the order of computational tasks on each processor and the second one contains the order of communication with the processor's neighbors.

The additional (penalty) time required for a single time step due to communication and idle time of processors is composed of the following main contributions: the communication time due to the transfer of the forward step coefficients and the backward step solution of the Thomas algorithm; the idle time due to waiting for communication with the neighboring processor and the communication time due to the transfer of the values of the main variables between neighboring subdomains. The optimal number of lines solved per message in forward direction is given as follows:

$$\begin{cases} \sqrt{N(1+\rho)\gamma/\rho} & \text{if } N \geq N_{cr} \\ \dfrac{N^2}{2(N_d - 1) + \lceil \rho \rceil} & \text{if } N'_{cr} \leq N \leq N_{cr} \end{cases} \qquad (3)$$

$$\frac{1}{\rho}\sqrt{\frac{N(1+\rho)\gamma}{2(N_d-1)+\lceil\rho\rceil}} \quad \text{if } N < N'_{cr}$$

Here a cubic domain is divided regularly into $N_d \times N_d \times N_d$ cubic subdomains with the $N \times N \times N$ grid nodes each one, γ is the ratio between the communication latency and the characteristic computational time per grid node and ρ is the ratio between the forward and the backward step computational times. The values of N_{cr} and N'_{cr} are defined as follows:

$$N_{cr} = \left[\frac{(2(N_d-1)+\lceil\rho\rceil-1)^2(1+\rho)\gamma)}{\rho^2}\right]^{1/3}, N'_{cr} = \frac{N_{cr}}{(2(N_d-1)+\lceil\rho\rceil-1)^{1/3}}. \quad (4)$$

Derivation of the above formulae and other results of the model are presented in [6]. The asymptotic order of the penalty function in terms of the overall number of nodes and the overall number of subdomains (processors) is shown in [6]. Non-linear Poisson equation is taken as a test case. Processors compute non-linear coefficients $S(U)$ while they are idle from the Thomas algorithm computations. Results of multiprocessor runs are presented in Table 1.

Parallelization of the target code. Recommendations with regard to the processor scheduling are applied to the code FDL3DI as follows. This code includes three dimensional solution of a system of five Euler or Navier-Stokes PDE. The version considered in this study is based on the diagonalization technique of Pulliam and Chaussee [9] leading to the decoupling of variables and solution of five penta-diagonal scalar systems in each direction. For the scalar penta-diagonal version, one may use processors for local computations while they are idle from the Thomas algorithm. These local computations include data-independent computations of discretized coefficients and data-dependent multiplications of intermediate ADI functions by transformation metrics.

The legacy code FDL3DI is based on plane-by-plane solution of banded systems. To reduce the number of pipelines and to remove local computations from pipelines the code has been changed as follows. The dimension of arrays of forward step coefficients is increased to store these coefficients for the entire computational volume. The FDL3DI solves separately three scalar penta-diagonal systems with the same coefficient matrices and two scalar penta-diagonal systems with different coefficient matrix. The data fluxes originated from these sub-routines are merged. The array governing the order of computational tasks is placed in corresponding sub-routines via the COMMON block. Calls of MPI communication procedures are done from the main routine (i.e., separated from computations) and are governed by the schedule array. To compute local discretization coefficients, the values of main variables on two near-boundary grid planes of subdomains are transfered to neighboring processors after each time step. The sample multiprocessor computations were performed on 64 processors of CRAY T3E MIMD computer. The number of grid nodes per subdomain varies from 15^3 to 25^3 (see Table 2).

Future research. The scheduling algorithm will be developed for multi-block parallel computations. In this case a processor solves banded systems originated from different grids and receive data from neighboring processors non-regularly. Therefore, each processor can treat different sub-domains and each sub-domain can be mapped to several processors to overcome the problem of load imbalance.

This methodology will be implemented to compact explicit schemes where independent banded systems are solved in spatial directions. Finally, we will implement this method of parallelization to the multi- block aeroacoustic code which combines the FDL3DI aerodynamic solver and a compact aeroacoustic solver.

References

[1] Ch. Hirsch, Numerical computation of internal and external flows, Vol. 1: Fundamentals of numerical discretization, John Wiley and Sons, 1994.

[2] Naik, N.H., Naik, V.K. and Nicoules, M., Parallelization of a Class of Implicit Finite Difference Schemes in Computational Fluid Dynamics. Int. Journ. of High Speed Computing, Vol 5, No 1, (1993), pp. 1-50.

[3] Gustafson, F.G. and Gupta, A. A new Parallel Algorithm for Tridiagonal Symmetric Positive Definite Systems of Equations, Proceedings of the third International Workshop of Applied Parallel Computing, PARA'96, pp. 341-349.

[4] Povitsky,A. and Wolfshtein, M. Multi-domain Implicit Numerical Scheme, International Journal of Numerical Methods in Fluids, 25, 1997, pp. 547-566.

[5] A. Povitsky. Parallelization of the pipelined Thomas algorithm. Submitted as ICASE Report, http://www.icase.edu/ $\sim$ aeralpo

[6] A. Povitsky. Parallel Directionally Split Solver Based on Reformulation of Pipelined Thomas Algorithm. Submitted as ICASE Report, http://www.icase.edu/ $\sim$ aeralpo

[7] Ian Foster. Designing and Building Parallel Programs, Addison-Wesley, 1995, http://www-fp.mcs.anl.gov/division/people/.

[8] Visbal, M and Gaitonde, D., "High-Order Accurate Methods for Unsteady Vortical Flows on Curvilinear Meshes," AIAA Paper 98-0131, January 1998.

[9] T. H. Pulliam and D. Chaussee, A diagonal form of an implicit approximate-factorization algorithm, Journal of Computational Physics, 39 (1981), pp. 347-363.

Table 1: Speedup for the benchmark problem on the CRAY T3E multiprocessor system. N_{tot} - total number of nodes, N - number of grid nodes per subdomain in a single direction, K_x, K_y and K_z - number of lines solved by backward step per message, Method - method of computation of K_x, K_y and K_z values, respectively (1,2 and 3 correspond to the first, second and third line of Eq (4)), Pn - measured penalty, PnM - theoretical value of penalty

		Proposed algorithm						Basic algorithm				
N_{tot}	N	K_x	K_y	K_z	Method	Pn, %	PnM, %	K_x	K_y	K_z	Pn, %	PnM, %
partitioning 3 x 3 x 3												
27000	10	51	51	26	112	52.41	39.90	22	22	22	108.35	96.74
91125	15	63	63	62	112	23.80	19.97	27	27	27	53.60	49.93
216000	20	74	74	74	111	15.01	13.56	31	31	31	33.86	31.33
partitioning 4 x 4 x 4												
64000	10	42	42	20	222	58.71	43.94	18	18	18	132.4	114.53
216000	15	63	63	47	112	31.10	21.34	22	22	22	59.23	57.46
512000	20	74	74	74	111	15.19	13.56	26	26	26	39.46	36.67
partitioning 5 x 5 x 5												
125000	10	35	35	17	222	67.29	54.54	16	16	16	152.20	131.61
421875	15	63	63	38	112	33.84	22.69	19	19	19	70.17	66.33
1000000	20	74	74	67	112	16.12	13.01	22	22	22	45.08	41.59
partitioning 10 x 10 x 10												
1000000	10	19	19	10	222	-	92.66	11	11	11	-	186.33
3375000	15	40	40	19	222	-	29.84	13	13	13	-	93.51
8000000	20	74	74	35	222	-	13.80	15	15	15	-	59.09
partitioning 8 x 4 x 2												
64000	10	22	42	40	222	56.74	42.37	12	18	31	136.83	119.59
216000	15	51	63	63	211	23.12	20.89	15	22	38	61.23	59.77
512000	20	74	74	74	111	15.11	13.56	17	26	44	39.23	37.71

Table 2: Parallelization penalty of the FDL3DI target code

Problem size		Measured parallelization penalty, %	
N_{tot}	N	Basic algorithm	Proposed algorithm
1000000	25	33.45	18.18
512000	20	36.35	19.46
216000	15	99.29	22.32

A COMPACT HIGH-ORDER UNSTRUCTURED GRIDS METHOD FOR THE SOLUTION OF EULER EQUATIONS

R. K. Agarwal, Wichita State University, Wichita, KS, 67260, 316-978-5226, agarwal@niar.twsu.edu
D. W. Halt, Ford Motor Company, Dearborn, MI, (313) 322-1036, Email: dhalt@ford.com

ABSTRACT

Two compact higher-order methods are presented for solving the Euler equations in two dimensions. The flow domain is discretized by triangles. The methods use a characteristic-based approach with a cell-centered finite-volume method. Polynomials of order 0 through 3 are used in each cell to represent the conservation flow variables. Solutions are demonstrated to achieve up to fourth order accuracy. Computations are presented for a variety of fluid flow applications. Numerical results demonstrate a substantial gain in efficiency using compact higher-order elements over the lower-order elements.

INTRODUCTION

A compact higher order polynomial reconstruction technique is developed that allows for higher order characteristic-based numerical solutions to the Euler equations on unstructured grids. This reconstruction requires extended sets of Euler equations that include as dependent variables not only the associated physical variables but also their spatial derivatives. These additional dependent variables are used in a compact polynomial reconstruction process within a finite-volume framework. The development presented follows some of the ideas in the computational fluid dynamics (CFD) literature using continuous and discontinuous piecewise polynomial approximations. Two methods are presented for extending the Euler equations. The first method solves the spatial derivatives of the governing integral equations. The second method follows the work of Allmaras [1], where the spatial moments of the governing equations are solved. Allmaras developed an approach with linear reconstructions on structured grids by using the first moment equations. This paper extends his approach to higher order and unstructured grids. Numerical results are presented for the transonic shockless Ringleb flow [2] and transonic flow past a sinusoidal bump, NACA 0012 airfoil and NLR 7301 airfoil. For Ringleb flow, a matrix of results is compared with the exact hodograph solution to establish accuracy levels for a two-dimensional transonic shockless flow. A study is also performed to ascertain the relative efficiencies of various orders of compact reconstruction.

GOVERNING EQUATIONS

The governing equations for an unstructured two-dimensional grid are written in integral equation form as follows:

$$\frac{\partial}{\partial t}\int\int_{v} Q dV + \int_{s} F\cdot\hat{n}dS = 0 \quad , \quad F = G\hat{i} + H\hat{j} \tag{1}$$

$$Q=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho e_0\end{pmatrix}, \quad G=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uh_0\end{pmatrix}, \quad H=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vh_0\end{pmatrix}, \quad and \quad p=\rho(\gamma-1)\left[e_0-\frac{u^2+v^2}{2}\right], \tag{2}$$

where ρ is the density, u and v are the velocity components, e_0 is the total energy per unit mass, h_0 is the total enthalpy per unit mass, and p is the static pressure.

METHOD 1 – DERIVATIVE EQUATIONS

The first method considers the spatial derivatives of the governing equation to be an extended set of governing equations:

$$\frac{\partial^m}{\partial x^m}\frac{\partial^n}{\partial y^n}\left[\frac{\partial}{\partial t}\iint_v Q dV+\int_s F\cdot\hat{n}dS\right]=0, \tag{3}$$

where $0 \le m \le k$, $0 \le n \le k$, and $0 \le m+n \le k$. They are solved simultaneously to level k to achieve the compact form of the reconstruction. The number of simultaneous equations necessary for a level k compact reconstruction in two dimensions is

$$neq_k = \frac{(k+1)(k+2)}{2}. \tag{4}$$

The unknowns are the coefficients of the Taylor series expansion of the conservation variables.

Compact Reconstruction

A compact reconstruction polynomial is applied at each cell locally. The dependent variables of each cell represent the average values that the reconstructed polynomial must have over each cell. For example, given the average value of the dependent variables Q_{mn} of cell i, a polynomial of degree $k = 1$ is reconstructed as follows:

$$Q(x,y)_i = Q_{00} + (x-x_c)Q_{10} + (y-y_c)Q_{01}, \tag{5}$$

where $Q_{mn} = (1/V)\int\int_v (\partial^m/\partial x^m)(\partial^n/\partial y^n)Q dV$, *and* (x_c, y_c) is the centroid of cell i. Polynomials of higher order than $k = 1$ require the use of calculated moments of inertia for each cell. The reconstruction polynomial is used to determine the left and right states at all Gaussian quadrature points on the shared cell edge. Flux values are evaluated at each Gauss point by a characteristic analysis of the left and right states. Numerical integration of the fluxes through each edge is accomplished by Gaussian quadrature. One Gauss point on the center of each edge is used with $k = 0$ and $k = 1$ reconstructions. This is sufficient for integrations that provide first- and second-order spatial accuracy, respectively. Two Gauss points on each edge are used with $k = 2$ and $k = 3$ reconstructions and are sufficient for third- and fourth-order spatial accuracy, respectively.

Characteristic-Based Flux Model

An appropriate flux vector is determined by approximating a pseudo one-dimensional Riemann solution between the left and right states at any Gauss point as follows:

$$F_{mn} = \tfrac{1}{2}\left[F_{mnR} + F_{mnL} - S|\Delta|\Delta^{-1}S^{-1}(F_{mnR} - F_{mnL})\right], \tag{6}$$

where S is the similarity matrix for diagonalizing the Jacobian $\partial F_{mn}/\partial Q_{mn}$, Δ is the eigenvalue diagonal matrix, and $|\Delta|$ refers to the absolute value of Δ obtained by taking the absolute values of each eigenvalue in the matrix. Note that the Jacobian is identical for all m and n, i.e.,

$$\partial F_{\infty}/\partial Q_{\infty} = \partial F_{mn/\partial Q_{mn}}. \tag{7}$$

The last term in the flux formula is often referred to as the dissipation term in the numerical method. It is interesting to note that this term exists for each equation in the extended set; however, the order of accuracy is not diminished by this. It will be shown for the Ringleb flow problem that the order of accuracy of $k + 1$ results from reconstruction polynomials of order k. The integration of the fluxes through the cell faces is accomplished by the method of Gaussian quadrature [3]. Given the discrete flux information at certain positions (referred to as Gauss points) along each cell face, a high-order integration can be performed.

Numerical Method

The computational domain is discretized by a novel cell-centered finite volume method using an approximate Riemann solver. The dependent variables Q_{mn} are updated by the Jacobi algorithm for each cell i as follows:

$$\frac{Q_{mni}^{n+1} - Q_{mni}^{n}}{\Delta t} V_i + \sum_{j=1}^{nfaces} \left(F_{mnj} \cdot \hat{n}_j\right) A_j = 0, \tag{8}$$

where A_j is the area of face j, V_i is the volume of cell i and Δt is the time step. The indexing or pointer system is based on two primary pointer arrays for each face. The first array points to the two cells adjacent to each face. The second array identifies the vertices of each face. The metrics can be efficiently computed from the pointer array defining the vertices of each cell face. The flow solver initially computes the left and right states for each face. The residuals are accumulated for each cell by looping through the faces and adding/subtracting flux contributions of the face from the associated left and right cells. The scheme then updates the dependent variables for every cell, and the iteration process is repeated for a desired number of iterations.

METHOD 2 – MOMENT EQUATIONS

The second method requires the solution of the moment equations

$$\iint_v x^m y^n \left[\frac{\partial}{\partial t} Q + \nabla \cdot F\right] dV = 0 \,, \tag{9}$$

where $0 \le m \le k$, $0 \le n \le k$, $0 \le m + n \le k$, and the (x,y) origin is shifted to the centroid of each cell locally. Equation (3) also gives the number of simultaneous equations for the second method. The equation for $m = n = 0$ is the same as Eq. (1) for both methods. All higher order equations are different for the two methods. The number of equations necessary for a level k compact reconstruction are the same as Eq. (4). To cast Eq. (9) in a form where spatial derivatives can be updated in time, Eq. (5) is substituted into Q in Eq. (9). Using the fact that first moment of inertias about a centroid are zero, the first moment equations reduce to

$$A_{xx} Q_{xt} + A_{xy} Q_{yt} + \int_s xF \cdot \hat{n} dS - \iint_v G dV = 0 \tag{10}$$

and

$$A_{xx} Q_{xt} + A_{yy} Q_{yt} + \int_s xF \cdot \hat{n} dS - \iint_v H dV = 0 \,, \tag{11}$$

where A_{xx}, A_{xy}, and A_{yy} are the second moments of inertia. The evaluation of line integrals in Eqs. (10) and (11) is performed by the Gaussian quadrature [3]. The integration of the flux G and H over the cell volume is done numerically using the efficient quadrature formulas for the triangle by Dunavant [4]. The characteristic-based flux model for this method is simpler than the first method and can use standard flux-split formulas such as those of Roe, van Leer, or Osher and Chakravarthy.

BOUNDARY CONDITIONS

The boundary conditions for the Ringleb flow problem come from the hodograph solution. The Ringleb boundary values are specified from the exact solution [2] as the outer state for the cell edges along the outer boundary. For problems where the outer states are not known, a farfield, symmetry, and Neumann condition are imposed. In the first method, it is quite difficult to properly impose these conditions because derivatives of mass, momentum, and energy are involved in the implementation. The second method uses conventional boundary conditions and it is much simpler to implement proper boundary conditions. Flow tangency is imposed for the surface Neumann condition by subtracting the normal velocity component from velocity vector

at Gauss points. The left and right states are set equal here so that zero mass and energy fluxes are insured through the surface boundary. Characteristic boundary conditions are applied to the farfield boundary by using the appropriate values of freestream and flowfield Riemann variables.

COMPUTATIONAL TEST CASES

Ringleb Flow

The Ringleb flow is chosen as a model problem because it is a transonic flow and an exact solution exists for comparison. A triangular region is selected from the Ringleb flowfield as the model region shown in Fig. 1. A curved region is not selected here because a series of straight-line segments along the curved boundary would introduce a truncation error from modeling the boundary shape. A matrix of results for various grid sizes is compared with the exact solution so that the accuracy and efficiency of the compact higher order methods can be demonstrated.

Method 1: A set of computational results is shown in Table 1. In Table 1, k denotes the degree of polynomial reconstruction. For each k, successive grids are generated by subdividing each cell into four smaller cells of equal size. L2 error refers to the rms error between the reconstructed and exact density at the vertices of each cell. SP.RAD refers to the average spectral radius over the first six orders of magnitude of residual reduction. CPU refers to the number of microseconds of CPU time per iteration per cell used by a single CRAY YMP processor. The time steps used were 0.1, 0.05, 0.025, and 0.015, respectively, for the 16, 64, 256, and 1024 cell cases and were near optimal for the point Jacobi scheme. One Gauss point is used on each edge when $k = 0$ or 1, whereas two Gauss points are used when $k = 2$ or 3. The order of accuracy is determined to be approximately $k + 1$ for each k in Table 1. As expected, the spectral radius increases as the number of cells increases. The CPU time and memory requirement also increase as k increases because more equations are solved per cell. The accuracy of the compact scheme is compared with the noncompact scheme of Barth [5] in Fig. 2. The first method is nearly a half of an order of magnitude more accurate than the noncompact method for $k = 3$ and furthermore shows a promising slope for extension to $k > 3$.

Method 2: For the second method, a set of computational results is shown in Table 2. Solutions from 1 cell through 1024 cells are cross-tabulated for constant ($k = 0$) to cubic ($k = 3$) polynomial reconstructions. The errors are significantly lower than corresponding cases in Table 1 when $k > 0$. However, the spectral radii and CPU times per iteration per cell are higher. The CPU times are higher primarily because of the increased number of Gauss points needed for numerical integration. The CPU times are lower than those in Table 1 for the $k = 0$ cases because of more efficient programming techniques used for the second method. The memory needed is roughly the same for both methods. The results for the cases with 256 cells are compared with the first method and the noncompact method of Barth [5] in Fig. 2. The L2 error is substantially smaller for the second compact method. For $k = 3$ solutions, the second method is two orders of magnitude more accurate than the noncompact result and an order and a half more accurate than the first compact method.

Sinusoidal Bump

The flow over a sinusoidal bump at freestream Mach number of 0.3 was selected as a test case to assess the accuracy of the curved edge model. An unstructured grid of 243 grid points and 425 cells was generated using the advancing front grid generator and is shown in Fig. 3 for $k = 0$ through $k = 3$ reconstruction. Method 2 is used to perform the Euler calculations. An accuracy and efficiency comparison is performed for this case between CFL2D [6] and the second

compact higher order method. CFL2D is used for three grid sizes of 61 by 21, 121 by 41 and 241 by 81. The compact higher order method is used with the order of reconstruction ranging from $k=1$ to $k=3$. Figure 4 compares the efficiency of each result. Plotted along the x axis is the total CRAY-YMP CPU time used to converge the L2 norm of the continuity equation residual by three orders of magnitude. The deviation in the peak pressure coefficient is plotted for each Euler solution along the y axis. The results show the compact higher order method to be much more efficient in reducing the peak pressure error. The $k=2$ solution is about an order of magnitude more accurate than the CFL2D solution on the 241 by 81 grid and furthermore uses an order of magnitude less CPU time.

NLR 7301 Airfoil

The NLR 7301 airfoil was run at Mach 0.721 and –0.194 degrees angle of attack using Method 2. The unstructured grid shown in Fig. 5 is much coarser in comparison to those in [7], yet accurate results are obtained. The grid is composed of 2379 grid points and 4496 triangles with 234 points on the airfoil surface. The far field is a rectangular box which extends 10 chordlengths from the airfoil surface. A point vortex was added to the far field boundary condition to better model the circulation in the far field. The compact higher order solution is shown in Fig. 6 in comparison to the hodograph computed pressures [7]. The pressures match the hodograph solution very well.

NACA 0012 Airfoil

The standard AGARD [7] 2D Euler test case of flow over a NACA 0012 airfoil at a freestream Mach number of 0.8 and angle of attack of 1.25 degrees was chosen to test the shock capturing ability of the second compact high-order method. A high-order solution was computed on the unstructured grid of 848 grid points and 1586 triangles with 134 points on the airfoil surface shown in Fig. 7. The results of Jameson and Schmidt [7] using a structured grid of 320 by 64 quadrilaterals are shown for comparison in Fig. 8. A close comparison in surface pressures can be seen everywhere except near the shocks on both upper and lower surfaces.

CONCLUSIONS

The quadratic and cubic polynomial reconstructions for a given number of cells are demonstrated to provide much more accurate solutions than lower order reconstructions. A significant gain in efficiency is also demonstrated over a wide range of accuracy levels since fewer cells are needed by the higher order methods for maintaining the same level of accuracy. The larger cell size accommodates larger time steps, which in turn results in quicker convergence rates. This effect seems to outweigh the burden of extra work needed for the higher order methods. The higher order solutions were computed an order of magnitude faster (and with less memory) at moderate error levels than a lower order solution, which needs far more cells. Method 2 (the moment method) appears more robust than the first method (the derivative method) primarily due to the simpler boundary conditions.

REFERENCES

1. Allmaras, S. R., "A Coupled Euler/Navier-Stokes Algorithm for 2-D Unsteady Transonic Shock/Boundary-Layer Interaction," Ph.D. Thesis, Massachusetts Inst. Of Technology, Dept. of Aeronautics and Astronautics, Cambridge, MA, March 1989.
2. Chiocchia, G. (ed.), "Exact Solutions to Transonic and Supersonic Flows," AGARD Advisory Rept. AR-211, May 1985.

3. Stroud, A. H., *Numerical Quadrature and Solution of Ordinary Differential Equations*," Springer-Verlag New York Inc., 1974.
4. Dunavant, D. A., "High Degree Efficient Symmetrical Gaussian Quadrature Rules for the Triangle," *International Journal for Numerical Methods in Engineering*, Vol. 21, No. 6, 1985, pp. 1129-1148.
5. Barth, T. and Frederickson, P., "Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction," AIAA Paper 90-0013, Jan. 1990.
6. Rumsey, C., Taylor, S., Thomas, J., and Anderson, W., "Application of an Upwind Navier-Stokes Code to Two-Dimensional Transonic Airfoil Flow," AIAA Paper 87-0413, Jan. 1987.
7. Chiocchia, G., "Test Cases for Inviscid Flow Field Methods," AGARD Advisory Rept. AR-211, May 1985.

Table 1 – Ringleb Efficiency Results Using Derivative Method.

k	CELLS	L2 ERROR	SP.RAD	CPU	MEM
0	16	7.80×10^{-2}	0.917	11	1.0
0	64	4.08×10^{-2}	0.946	6	2.6
0	256	2.09×10^{-2}	0.977	4	9.0
0	1024	1.06×10^{-2}	0.988	4	34.4
1	16	2.15×10^{-2}	0.931	24	1.5
1	64	6.05×10^{-3}	0.964	13	4.6
1	256	1.65×10^{-3}	0.984	11	16.9
1	1024	4.46×10^{-4}	0.989	10	66.0
2	16	6.99×10^{-3}	0.937	84	2.3
2	64	2.17×10^{-3}	0.968	47	7.7
2	256	3.16×10^{-4}	0.979	38	28.7
2	1024	4.37×10^{-5}	0.990	35	112.7
3	16	2.06×10^{-2}	0.952	158	3.4
3	64	9.70×10^{-4}	0.974	90	11.7
3	256	1.63×10^{-4}	0.990	75	44.3
3	1024	8.71×10^{-6}	0.994	70	174.9

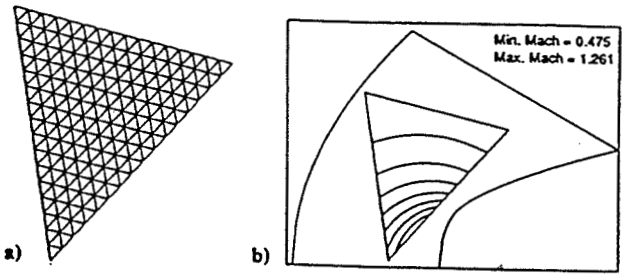


Figure 1 – Two Hundred and Fifty-Six Cell Case:
a) Unstructured Grid and
b) Cellwise Continuous Mach Contours for k=3.

Table 2 – Ringleb Efficiency Results Using Moment Method.

k	CELLS	L2 ERROR	SP.RAD	CPU	MEM
0	1	1.97×10^{-1}	0.740	85	0.05
0	4	1.05×10^{-1}	0.855	22	0.16
0	16	7.80×10^{-2}	0.906	7	0.56
0	64	4.08×10^{-2}	0.946	3	2.1
0	256	2.09×10^{-2}	0.977	3	8.3
0	1024	1.06×10^{-2}	0.988	2	32.7
1	1	7.45×10^{-2}	0.923	284	0.08
1	4	4.86×10^{-2}	0.964	82	0.27
1	16	1.07×10^{-2}	0.965	33	1.0
1	64	2.99×10^{-3}	0.980	22	4.0
1	256	7.82×10^{-4}	0.986	20	15.7
1	1024	2.04×10^{-4}	0.992	20	62.4
2	1	4.39×10^{-2}	0.951	670	0.13
2	4	9.54×10^{-3}	0.958	202	0.5
2	16	1.91×10^{-3}	0.982	87	1.9
2	64	3.29×10^{-4}	0.989	63	7.6
2	256	4.62×10^{-5}	0.992	59	30.3
2	1024	6.46×10^{-6}	0.996	57	120.7
3	1	3.07×10^{-2}	0.971	1502	0.24
3	4	3.85×10^{-3}	0.983	444	0.9
3	16	5.25×10^{-4}	0.997	184	3.6
3	64	6.52×10^{-5}	0.996	130	14.4
3	256	5.49×10^{-6}	0.998	121	57.2

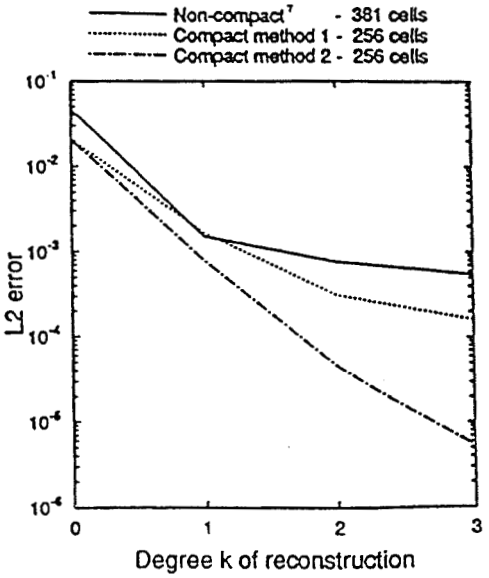


Figure 2 – Comparison of L2 Errors Between Compact and Noncompact Schemes.

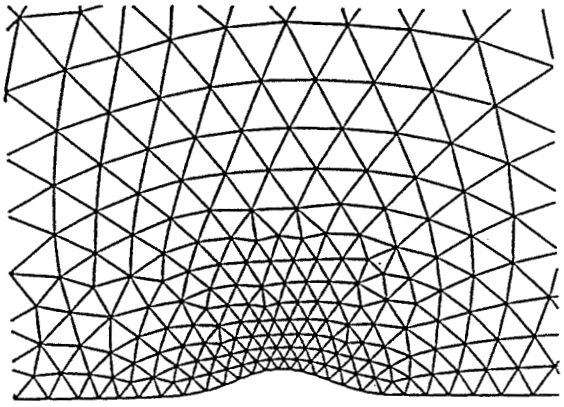


Figure 3 – Unstructured Grid for Sinusoidal Bump Case.

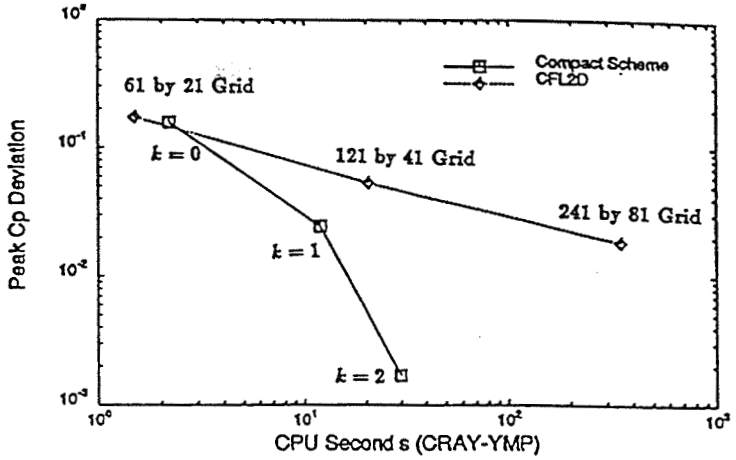


Figure 4 – Code Efficiency Comparison for Bump Case.

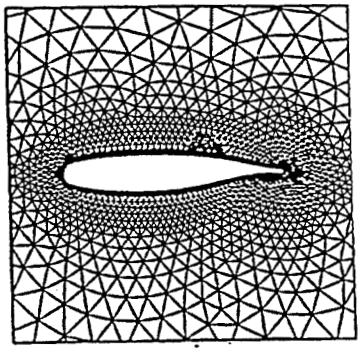


Figure 5 – Unstructured Grid for NLR 7301 Airfoil Case.

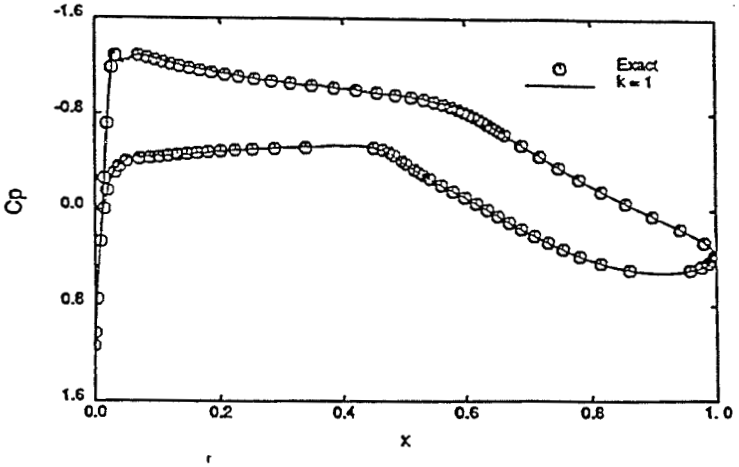


Figure 6 – Surface Pressure Coefficient Distribution for NLR Airfoil Case.

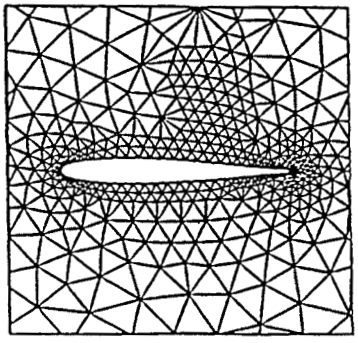


Figure 7 – Unstructured Grid for NACA 0012 Airfoil Case.

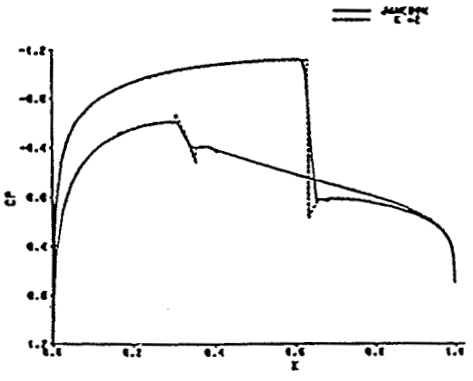


Figure 8 – Surface Pressure Coefficient Distribution for NACA Airfoil Case.

A NEURAL NETWORK AERO DESIGN SYSTEM FOR ADVANCED TURBO-ENGINES

Jose M. Sanz
NASA Lewis Research Center

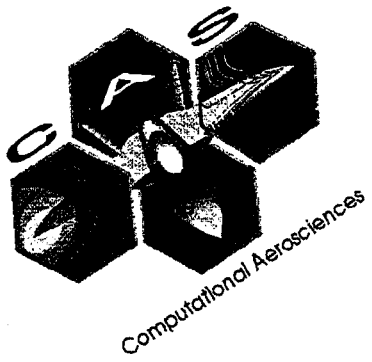
An inverse design method calculates the blade shape that produces a prescribed input pressure distribution. By controlling this input pressure distribution the aerodynamic design objectives can easily be met. Because of the intrinsic relationship between pressure distribution and airfoil physical properties, a Neural Network can be trained to choose the optimal pressure distribution that would meet a set of physical requirements.

Neural network systems have been attempted in the context of direct design methods. From properties ascribed to a set of blades the neural network is trained to infer the properties of an `interpolated' blade shape. The problem is that, specially in transonic regimes where we deal with intrinsically non linear and ill posed problems, small perturbations of the blade shape can produce very large variations of the flow parameters. It is very unlikely that, under these circumstances, a neural network will be able to find the proper solution.

The unique situation in the present method is that the neural network can be trained to extract the required input pressure distribution from a database of pressure distributions while the inverse method will still compute the exact blade shape that corresponds to this `interpolated' input pressure distribution. In other words, the interpolation process is transferred to a smoother problem, namely, finding what pressure distribution would produce the required flow conditions and, once this is done, the inverse method will compute the exact solution for this problem.

The use of neural network is, in this context, highly related to the use of proper optimization techniques. The optimization is used essentially as an automation procedure to force the input pressure distributions to achieve the required aero and structural design parameters. A multilayered feed forward network with backpropagation is used to train the system for pattern association and classification.

OMIT THIS PAGE



Session 3:

Parallel System Software Technology

THE SGI/CRAY T3E: EXPERIENCES AND INSIGHTS

Lisa Hamet Bernard
NASA Goddard Space Flight Center
Code 931
Greenbelt, MD 20771
Lisa.Bernard@gsfc.nasa.gov
301-286-9417

1. Background

The focus of the HPCC Earth and Space Sciences (ESS) Project is *capability computing* – pushing highly scalable computing testbeds to their performance limits. The drivers of this focus are the Grand Challenge problems in Earth and space science: those that could not be addressed in a *capacity computing* environment where large jobs must continually compete for resources. These Grand Challenge codes require a high degree of communication, large memory, and very large I/O (throughout the duration of the processing, not just in loading initial conditions and saving final results). This set of parameters led to the selection of an SGI/Cray T3E as the current ESS Computing Testbed.

In 1996, the ESS Project entered into a series of NASA Cooperative Agreements, one with a scalable testbed vendor (SGI/Cray) and nine with Earth and space science Grand Challenge Investigator teams. All ten awardees were competitively selected. The cooperative agreements are performance-based, wherein payments are triggered by achievement of milestones. Furthermore, each Grand Challenge Investigator team agreement includes the ESS Project milestones of achieving 10, 50, and 100 GFLOPS sustained performance on a code key to the team's research. These performance milestones are to be met on testbeds provided by SGI/Cray, which has the same milestones: to enable the teams to achieve their performance goals. This arrangement led the way for SGI/Cray to place a 512 processing element (PE) T3E at NASA/Goddard Space Flight Center in March 1997. The system is named jsimpson, in honor of pioneering meteorologist Dr. Joanne Simpson. Fifty percent of the resources are allocated to the nine Grand Challenge research teams, 20 percent to the general NASA science community, 15 percent to the NASA Computational Aerosciences Project, 10 percent to the vendor, and the remaining five percent to system software research. In March 1998, the NASA Earth Science Enterprise purchased an additional 512 PE's and accompanying disk for the NASA Seasonal to Interannual Prediction Project (NSIPP). The two "halves" of the system must be managed independently, though they share key system components. Consequently, HPCC and NSIPP system requirements must be carefully coordinated to enable success for both projects. It is under these diverse requirements that jsimpson is managed.

2. T3E Architecture and Configuration

The T3E at GSFC, model T3E-600 LC1024/128, is a liquid-cooled, distributed memory system composed of 1,088 DEC Alpha EV5 chip PE's, each with 128 MB of local DRAM memory and a peak performance of 600 MFLOPS. Each PE contains an 8 KB primary cache and a 96 KB three-way set associative secondary cache. The PE's are connected by a low latency, high bandwidth bi-directional 3-D torus. Interprocessor data payload communication rates are 480 MB per second in every direction through the torus. Each PE contains a C-chip to enable streams, a latency hiding feature. When the circuitry detects fetches of contiguous addresses in local memory, the system assigns a stream buffer (one of six) to this load sequence and performs prefetching. Another hardware feature which can be used to hide the latency of local or remote memory accesses is 512 off-chip memory-mapped E-registers.[1] Note that E-registers bypass cache, and therefore it is possible to access a memory location simultaneously through stream buffers and E-registers, a condition that can cause a PE or system hang. This potential situation is blocked through hardware

in later models of the T3E (T3E-900 and T3E-1200) and is prevented from occurring through software checks in the PVM and MPI communication libraries.[2] However, users striving for optimal performance prefer the SGI/Cray-proprietary shared memory (SHMEM) communication library, which has no guard against this conflict. Furthermore, there is no automated mechanism to determine if a code is "streams-safe". Consequently, jsimpson users are prohibited by default from using streams (via system parameters) and must have each code hand-checked by on-site Cray application staff before use of streams is allowed.

The T3E I/O topology consists of multiple GigaRings. Data travel across these counter-rotating, 32-bit dual-ring channels at rates up to one GB per second. There is one channel per 16 PE's. All channels are accessible and controllable from all PE's. All disk and tape controllers and network interfaces are GigaRing-attached. There are nine GigaRings in jsimpson. All disks (1,470 GB) are RAIDed (RAID-5) Fibre Channel and average 25 to 30 MB per second writes. Note that disk I/O performance is highly application-dependent and can be improved by using larger block sizes.

The T3E runs the UNICOS/mk operating system, a serverized, microkernel-based version of UNICOS, SGI/Cray's operating system for vector machines. It is derived from the UNIX System V operating system. UNICOS/mk provides a single-system image; each PE is not configured individually. Process-specific requests, such as memory allocation and message passing, are handled locally by the microkernel. Requests for global services are handled by servers which reside on dedicated PE's, as described in the next paragraph. This operating system is scalable because the number of servers is determined by the total number of PE's and scales accordingly.[3] The Network Queuing System (NQS) is the batch queuing system layered on top of UNICOS/mk. The global resource manager (GRM) acts as an interface between the operating system and NQS, assigning PE's to requesting processes.[4]

The 1,088 PE's are designated as application, command, or operating system (OS) PE's. Application PE's are used by all parallel processes (any process requiring more than one PE). The GSFC T3E is always configured with at least 1,024 application PE's. Command PE's handle all single PE processes, including user login shells, editing sessions, and compiles. Processes are distributed among the command PE's by a load-balancing algorithm, and these PE's, unlike application PE's, are time-shared. On average, there are 30 command PE's on jsimpson. However, this is also the pool of spare PE's, which are drawn upon when an application PE fails and must be "mapped out" (process described in section 4) until it can be physically replaced. The remaining PE's are designated OS PE's, which handle all global system-level functions such as file space allocation, scheduling and I/O management. A PE's designation may be dynamically changed between application and command PE's; adding or subtracting OS PE's requires a reboot.

A Storage Technologies, Inc. Powderhorn silo is directly attached to the GSFC T3E via eight SCSI-2 fast and wide controllers. The silo contains eight Timberline (linear 36-track, 800 MB tape capacity) drives, two per controller, and four Redwood (helical 10, 25, and 50 GB tape capacity) drives, each on a separate controller. The silo behaves as virtual disk via the SGI/Cray Data Migration Facility (DMF) software. One jsimpson filesystem is the "front-end" of the silo, and when that filesystem approaches capacity, files are automatically migrated to tape. When a user accesses a migrated file, it is automatically copied back to disk. File size determines the type of tape to which a file is migrated. The vast majority of files are small (tens to hundreds of MB), so they are initially migrated to 800 MB tapes, which have the lowest positioning times. Files that have not been accessed for 90 days are transferred to the larger tapes to conserve silo tape capacity.

3. System Constraints

The T3E architecture presents a unique challenge in maximizing system utilization and throughput while preserving the capability computing environment. Due to the focus on high speed communication between PE's, the T3E requires PE's to be allocated contiguously per job. Even

more constraining is the fact that, unlike in the conventional shared-memory vector environment, jobs cannot execute if the user- and/or code-specified number of PE's is not available. Application PE timesharing is possible but impractical for the ESS Project because context switching degrades system efficiency, and with no virtual memory, many codes require the full amount of available physical memory. A checkpoint/restart facility is available, though very large jobs take a considerable amount of time (up to 20 minutes), and therefore checkpointing is used only when the system must be taken down. Consequently, once a job begins, it has a set number of PE's and associated memory allocation until job completion or reaching the queue runtime limit, whichever comes first. With a highly varied job mix in both size and runtime of jobs, the resulting scenario is PE fragmentation and an inability to achieve near 100 percent utilization.

4. System Management Tools

SGI/Cray has implemented several tools to help mitigate the effects of the system constraints described in the previous section[5]. The political scheduler has many features which give the system administrator more control over the allocation of resources to particular jobs and users. One key feature used on jsimpson is the load balancer. At regular intervals, currently running jobs are migrated within the two predefined PE regions (HPCC and NSIPP) to pack the jobs and move all available PE's in each region into a contiguous block. This addresses the fragmentation problem and improves throughput for larger PE jobs. Each invocation of the migrate command halts execution of the effected job for the migration period, which lasts from a few to approximately 20 seconds, depending on the size of the migrating job and the location of the target PE's. (If there is overlap between the currently allocated PE's and the target PE's, then all PE's cannot be moved concurrently; hence the longer delay.) The caveat with this tool is not to run it too often. First, it introduces system overhead, which can be considerable. Second, in a scenario with many small jobs (both in number of PE's and runtime), the load balancer clashes with the resources manager, and both try allocating the same free block of PE's, one for a new job and one for a migrated job. This does not jeopardize currently running jobs, but it puts the system in thrashing mode. On jsimpson, NASA has settled on migration frequencies of five minutes and 10 minutes for the HPCC and NSIPP regions, respectively. Furthermore, once a job is migrated, it will not be considered for another migration for 10 minutes (HPCC) or 20 minutes (NSIPP).

The global resource manager is the mechanism by which a system administrator sets attributes for particular PE's. The diagram at the end of this section is a portion of the output from the grmview command. A key attribute used on jsimpson is the PE label. There are both hard and soft labels, distinguished by the first letter of the label (H or S). To use a labeled PE, the user's executable must also be tagged with the same label. With a hard label, the executable will not run if the exact number of PE's with the matching label are not available. With a soft label, PE's with a matching label are preferred, but the executable will run on other PE's if those are not available. In the case of jsimpson, hard labels are used to isolate the NSIPP-dedicated PE's from the HPCC PE's. Any application PE's beyond the 1,024 total (24 currently) are soft-labeled for NSIPP, so that both groups may access them. Since there is no mechanism to map queues to PE regions, the use of labels is the method NASA has found to effectively "divide" the PE's. Note that NASA has also labeled six command PE's as S256. Those PE's have 256 MB of memory, so users doing large builds can set the preference to have that job execute on a larger memory PE.

One final system management tool of note is SGI/Cray's recent implementation of the warmboot feature. If a single PE dies, that PE can be warmbooted without affecting any other PE or running job. This does not always work, depending on the severity of the error that caused the PE failure. Nonetheless, it is a large step forward from the previous requirement of a full system reboot to bring a failed PE back online. Since PE's must be allocated contiguously, an inconveniently located failed PE severely limits the maximum size of runnable jobs. If a PE fails on a hardware error, it can be mapped out to defer the time-consuming task of physically replacing the PE. To map out a PE, a spare PE is assigned the virtual PE address of the one being replaced. In this

instance, high-communication codes may see slight performance degradation due to the longer physical distance between virtual neighboring PE's. This is a short-term situation, corrected at the next scheduled maintenance period.

Sample output from grmview:

PE Map: 1088 (0x440) PEs configured

		Ap. Size		Number	Aps.	Abs.		<<<< Lists >>>>>			
Type	PE	min	max	running	limit	limit	Label	svc	uid	gid	acid
+ APP	0	2	1048	0	1	2	-	-	-	-	-
511 identical PEs skipped											
+ APP	0x200	2	1048	1	1	2	HNSIPP	-	-	-	-
255 identical PEs skipped											
+ APP	0x300	2	1048	0	1	2	HNSIPP	-	-	-	-
255 identical PEs skipped											
+ APP	0x400	2	1048	0	1	2	SNSIPP	-	-	-	-
23 identical PEs skipped											
+ CMD	0x418	1	1	0	unlim	unlim	-	-	-	-	-
17 identical PEs skipped											
+ CMD	0x42a	1	1	1	unlim	unlim	-	-	-	-	-
5 identical PEs skipped											
+ CMD	0x430	1	1	0	unlim	unlim	S256	-	-	-	-
5 identical PEs skipped											
+ OS	0x436	0	0	0	0	0	-	-	-	-	-
8 identical PEs skipped											
+ OS	0x43f	0	0	0	0	0	-	-	-	-	-

5. Performance and Utilization

Due to the milestone requirements of the Grand Challenge Investigators on the GSFC T3E, large job throughput is key to timely debugging and optimization. Therefore, the queues must be configured such that large PE jobs have priority over small PE jobs, and short jobs requiring 512 PE's may run at any time of the day. This eliminates the option of running long, production queues on a portion of the system during the primetime window (Mon-Fri 8AM to 8PM ET). Such a large development time is needed to accommodate users located across the U.S. This scenario sacrifices high system utilization (capacity computing) for capability computing needs and tends to lead to more PE idle time during the development window. Our monthly utilization averages 70 percent, which is high for T3E systems not running in operational mode. We categorize the unused 30 percent time as either "unusable" or "idle". "Unusable" is defined as time accrued by PE's that are idle because jobs waiting in active queues cannot fit in the contiguous blocks available. "Idle" is the time that no jobs are waiting to run. During primetime, 36 percent is idle and 8 percent is unusable. During nonprimetime, 8 percent is idle and 12 percent is unusable. The combined average is 18 percent idle and 12 percent unusable. Utilization is based on system availability, which is also high. SGI/Cray achieved 95 percent availability from May 1, 1997 through April 30, 1998 to meet one of their cooperative agreement milestones. Only six hours per month are permitted for system maintenance (and therefore not counted against the availability time), so this achievement reflects excellent system stability and reliability.

The GSFC T3E has demonstrated excellent performance numbers. The Linpack benchmark achieved 448.6 GFLOPS on the full system, placing jsimpson fifth in the world on the Top 500

Supercomputers List[6]. More impressive are the sustained performance numbers achieved on real science applications. A team studying Rayleigh-Benard-Marangoni Problems in a Microgravity Environment under Principal Investigator Dr. Graham Carey of the University of Texas at Austin achieved 118.7 GFLOPS[7]. A team studying Turbulent Convection and Dynamos in Stars under Principal Investigator Dr. Andrea Malagoli of the University of Chicago achieved 114, 101 and 104 GFLOPS on three different codes, respectively, within that research effort[8]. And a team studying Multiscale Modeling of the Heliosphere under Principal Investigator Dr. Tamas Gombosi of the University of Michigan achieved 67 GFLOPS using just 512 PE's. This code scales linearly, and achieved well over 200 GFLOPS on 1,024 PE's of a T3E-1200 (1.2 GFLOPS peak performance per PE)[9]. With the exception of this final example, all of these timings were accomplished on 1,024 PE's of jsimpson, when it was reconfigured solely for this purpose. The full machine is not generally available to the user community. Note that NASA did not require the 50 and 100 GFLOPS milestones to be met on the 512 PE HPCC T3E. This system only needed to achieve 25 GFLOPS; larger configurations could be used to attain the higher performance.

6. Experiences

As a distributed memory system, the T3E must be programmed with explicit message passing, using either PVM, MPI or the SGI/Cray-proprietary SHMEM library. Although HPF (High Performance Fortran) was designed to remove this limitation, it has not yet proven to be a viable option. As reported by C. Ding[10], HPF averages 2 to 4 times slower than message passing codes for many applications and does not scale as well. The T3E was not designed for any other programming style, and the execution of other types of codes leads to unexpected and undesirable operating system and scheduler behaviors. NASA's experience with jsimpson bears that out.

Two different users ported codes designed to run on workstations which used the programming model of forking processes. One code forked many single-processor threads. On the T3E, any process requiring only one processor automatically runs on command PE's. The consequence was that the command PE's, which also run all user shells and editing sessions, were saturated with processes, and response time for all users dropped dramatically. In another case, a parallel code was designed to periodically fork processes and halt the parent process while the children ran. Queue runtime limits are based on accumulated CPU time of the initial job, which in this case was the parent process. However, since application PE's are not timeshared, CPU time normally approximates wall clock time. In this case, though, no child process ran long enough to hit the runtime limit, and the difference between wall clock time and accumulated CPU time for the parent process was dramatic. The queue from which it was running had stopped hours before, but currently running jobs are not stopped (checkpointed) when a queue is stopped. (This job was killed once the systems administration staff determined what was happening.) The system was so confused by this version of parent/child process that it had a block of PE's allocated to this job the entire length of the run, even though PE's were regularly freed once a child process ended.

Another frequently observed user strategy that disrupts normal system operation is the auto-resubmission of batch jobs. Users expect the behavior of this strategy to be that the job goes to the bottom of the queue, and the job will not run again until the other queued jobs have completed. Unfortunately, this is not what happens. As soon as the initial job completes, NQS recognizes that the resubmitted job is the perfect fit for the resources just freed. NQS always attempts to maximize system utilization and therefore will not wait for other jobs to complete or migration to occur in order to fit a larger PE job in the system. Consequently, other user jobs are locked out if they do not fit in the remaining available PE's. Under special circumstances, NASA does allow autoresubmission, with the constraint that users put a significant time delay in their submission scripts between job queue submissions. This delay allows other queued jobs to execute and is more likely to produce the user's intended behavior on the system.

These types of codes negatively impact the T3E by monopolizing resources and subverting queuing and resource management systems. It is critical to the efficient use of T3E resources that users understand the programming model of the codes that they wish to run and the allocation methods used by the T3E. Such was not the case in these instances.

7. Summary

The T3E at the Goddard Space Flight Center is a unique computational resource within NASA. As such, it must be managed to effectively support the diverse research efforts across the NASA research community yet still enable the ESS Grand Challenge Investigator teams to achieve their performance milestones, for which the system was intended. To date, all Grand Challenge Investigator teams have achieved the 10 GFLOPS milestone, eight of nine have achieved the 50 GFLOPS milestone, and three have achieved the 100 GFLOPS milestone. In addition, many technical papers have been published highlighting results achieved on the NASA T3E, including some at this Workshop. The successes enabled by the NASA T3E computing environment are best illustrated by the 512 PE upgrade funded by the NASA Earth Science Enterprise earlier this year. Never before has an HPCC computing testbed been so well received by the general NASA science community that it was deemed critical to the success of a core NASA science effort[11]. NASA looks forward to many more success stories before the conclusion of the NASA–SGI/Cray cooperative agreement in June 1999.

References and Notes

1. Anderson, A., Brooks, J., Grassl, C. and Scott, S., "Performance of the Cray T3E Multiprocessor," *Proceedings of the 1997 ACM/IEEE SC97 Conference*, available on CDROM, November 1997.

2. Cray Research, Inc., *T3E Programming with Coherent Memory Streams*, 1996. Available online at http://www.psc.edu/machines/cray/t3e/streams/streams.html. Related description at http://www.cray.com/products/systems/t3e/streams.html.

3. Broner, G., "UNICOS/mk: A Scalable Distributed Operating System," *Proceedings of the Thirty-Seventh Semi-Annual Cray User Group Meeting*, pp. 237-240, March, 1996.

4. Cray Research Inc., *UNICOS/mk Resource Administration Manual*, SG-2602, 1996.

5. Ibid.

6. "Top500 Supercomputer Sites," http://www.netlib.org/benchmark/top500.html.

7. Carey, G. F., McLay, R., Bicken, G., and Barth, W., "Parallel Finite Element Solution of 3D Rayleigh-Benard-Marangoni Flows," to be published in *International Journal for Numerical Methods in Fluids.*

8. For a full description of this research activity, including project description, codes and performance measurements, see http://astro.uchicago.edu/Computing/HPCC.

9. For a full description of this research activity, including project description, codes, and performance measurements, see http://hpcc.engin.umich.edu/HPCC.

10. Ding, C.H.Q., "Evaluations of HPF for Practical Scientific Algorithms on T3E," *Lecture Notes in Computer Science*, Vol. 1401, P.M.A. Sloot, ed. Springer-Verlag, 1998, pp. 223-232.

11. "NASA Strategic Plan," http://www.hq.nasa.gov/office/nsp/.

THE METACENTER ROADMAP

Mary Hultquist, James Patton Jones
MRJ Technology Solutions
NASA Ames Research Center, M/S 258-6
Moffett Field, CA 94035-1000
650-604-0814

maryh@nas.nasa.gov

Abstract

As scientists increasingly take advantage of high performance computing in their research, the computational resources at any given site are not always adequate for the emerging needs of the scientific community. The focus of the NASA Metacenter Project is to provide these resources by linking together multiple supercomputing sites while giving the scientific user a single interface to access the increased computing capabilities.

This paper discusses the successes of the first NASA Metacenter project, the current development effort, the coordination necessary to create a metacenter, and a brief look at future directions.

1.0 Phase 1 NASA Metacenter

The Metacenter Project at NASA began in 1995 with the goal of balancing the workloads on IBM SP2 systems located at NASA-Langley and NASA-Ames. The difference in size and user base on each system caused an imbalance in the utilization of each system. By effectively merging these systems, better use could be made of the existing cycles. Talks began between the two sites, and a common user interface was devised. It was decided that all users on each system would be given access to both systems in the Metacenter. The batch queuing system, Portable Batch System (PBS), and the newly developed Peer Scheduler would handle moving the jobs between the systems.

This first attempt was quite successful, as it achieved more effective use of NASA Supercomputers by making the systems more easily available to researchers. It provided quicker turn-around for batch jobs, a larger range of available resources for computation, and a better distribution of the computational workload across multiple supercomputers.
However, it did require the user to take more responsibility for specifying the resources she would need for a given job. Since it was possible that the job would execute on a site other than where it was launched, the required files would have to move with the job. This was quite cumbersome at the beginning, where a single typographical error could cause a job to fail once it had reached the top of the queue. A graphical user interface, called *xpbs*, allowed the user to click on the requested files, minimizing the typographical error problem.

This first phase continued until the SP2s were decommissioned in February 1998. Since that time, NASA-Lewis and ICASE have joined the team, and the architecture has moved from the homogeneous distributed memory systems to SGI Origin2000s and a cluster of Sun Sparcstations.

2.0 Phase 2 NASA Metacenter

NASA-Lewis joined the Metacenter as the third aeronautics site in the Computational Aerosciences (CAS) Program in mid-1997. Plans were made to place small SGI Origin2000 systems at the three centers to continue working on expanding the homogeneous NASA Metacenter to three sites. With the inclusion of ICASE, the opportunity emerged to examine heterogeneous computing in the

near-term. These developments, as well as plans to support NASA's Information Power Grid (IPG) effort, have shaped the developmental focus of the second phase.

The current development work being explored by these sites is twofold: (1) being able to share jobs while maintaining different batch queuing systems, and (2) sharing jobs across heterogeneous architectures.

The first area of development is being explored using another layer of software such as Globus or Legion. This additional layer will translate the syntax of one batch queuing system to another, provide authentication between systems, and manage the job migration and tracking. This is especially important in the realm of IPG, since most sites involved in such an effort might prefer to keep their current interface with the grid portions being handled behind the scenes, rather than retraining their users in some other interface.

Site	Systems	Job Management Systems
NASA-Ames	8 processor SGI Origin 2000	Portable Batch System
NASA-Langley	16 processor SGI Origin 2000	Portable Batch System
NASA-Lewis	24 processor SGI Origin 2000	Load Sharing Facility
ICASE	Sun Sparcstation Cluster	Portable Batch System

Table 1. Initial Sites and Systems in the Phase 2 Metacenter.

The Globus software is currently being installed and integrated with two job management systems. NASA-Ames, NASA-Langley, and ICASE are using PBS as the default job management system. NASA-Lewis is using the Load Sharing Facility (LSF) as its job management system. Each of these products needs to interact with the Globus software on the Origin2000s and the Sun Sparcstations.

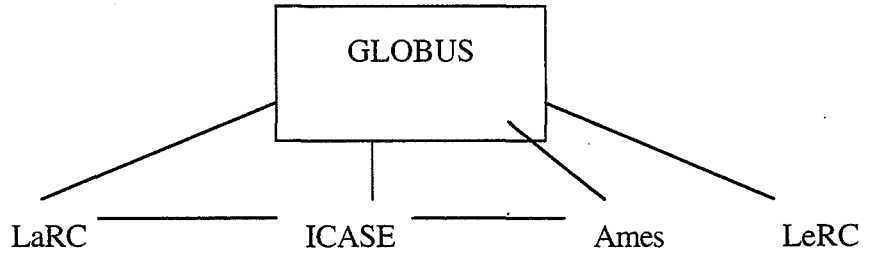


Figure 1. Globus is used as an intermediary between systems running differing job management systems. PBS jobs can either use the Peer Scheduler to share jobs with other PBS sites or Globus to share jobs with systems running LSF or another job management system.

Another middleware product under consideration is the Legion software. When choosing between these two packages, the Metacenter Team decided to integrate Globus first because of its "tools" approach. The Legion software was not as easy to implement in the current environments, while

Globus allowed for a more piecemeal approach. Both these packages will be studied, as well as others which may be appropriate, as the IPG effort continues.

3.0 Technology Transfer

The first phase of the NASA Metacenter was deemed a success, not so much for what was accomplished, but more so that it is considered useful to groups beyond the testbeds. There have been two off-shoots of the NASA Metacenter. The first, the NAS Cray Metacenter, used some of the technology of the NASA Metacenter to create a metacenter of the production Cray C90s and J90s at NAS. This environment is currently in production.

Beyond NASA, and closer to the original Metacenter, NASA is in the process of transferring the metacenter technology to two Department of Defense (DoD) Major Shared Resource Center (MSRC) sites: ASC at Wright-Patterson Air Force Base, Ohio, and U.S. Army Corps of Engineers Waterways Experimental Station (CEWES) in Vicksburg, Mississippi. These sites are creating an IBM SP metaqueuing environment based on the NASA Metacenter and PeerScheduler (the PBS scheduler that supports the dynamic movement of workload as needed across the Metacenter). In an agreement between these three sites, NASA is providing technical assistance in setting up this metaqueuing environment. This collaboration has been beneficial to all sites; the MSRCs receive technical assistance for their metaqueuing environment and NASA understands the requirements of computing environments beyond its gates. Many of the MSRC requirements have been included in the design of the next Peer Scheduler, as the groups continue the collaboration beyond the MSRC sites.

4.0 How to Create a Metacenter

The NASA/ASC/CEWES collaboration began with a white paper detailing the necessary steps to create a metacenter. A number of general issues have been identified as necessary components in creating any metacentered environment, not specifically NASA or MSRC. These can be broken down into technical issues and infrastructure. While it may seem that the technical issues would be more of a challenge, it is the issues of infrastructure that take the most time and energy. This was true of the first phase of the NASA Metacenter, even though there was considerable software to be written.

Some of the technical issues for creating a homogeneous metacenter include:

- Installing and maintaining the same level of software on each system. This includes everything from operating system and patch levels to compilers, third party software, and "home-grown" software.
- Expected environments. This includes naming conventions (or global variables) to find home directories and scratch filesystems, as well as the amount of memory and filesystem space dedicated to the user's job.
- How does a user track her job, once it has entered the metacenter?
- The systems must trust each other enough to share jobs. Additional layers of security may be added, but files must be able to be transferred in either direction.
- The networks between the systems should be the fastest possible, otherwise bottlenecks occur, and file transfers become an even greater issue.

Additional negotiations are required for some of the technically simpler, but more involved, infrastructure issues. Some of these include:

- Does a user have access to all the systems in the Metacenter?

- Is one approved account request form enough or does a user have to apply for each system in the metacenter? What if there are different security requirements?
- Are allocations (or time sold) merged for all systems in the metacenter or does a user receive a separate allocation for each system (or architecture for heterogeneous metacenters)?
- How is time tracked across the metacenter? Which site is the "master" for maintaining this information?
- Where does a user go to get help when there is a problem? How does each site's User Services group handle problems? Are they opened and tracked locally? What happens if the problem has to do with another site's systems? How does each site ensure that problems do not get dropped between sites?
- Where does a user go to get information on how to use the metacenter? Is there a central web page? If so, on which site's system does it reside? Is there training? Is it local or is it standardized to include the environments a user may encounter?

With each additional site that is added to a metacenter, these questions need to be answered anew. It is especially difficult with sites and systems which are already in production, since the users are already expecting a given environment, and the infrastructure for handling these types of user issues are already in place. This requires a change to how each site does business, which can lead to delays in implementation.

5.0 The Next Peer Scheduler

The NASA/MSRC team has begun to design the next Metacenter, which would considerably extend the NASA Metacenter. One major decision was to pull forward the Peer Scheduler design, redesigning and reimplementing as necessary. Where possible, the new Peer Scheduler will reuse code from existing schedulers.

The new Peer Scheduler will include support for heterogeneous computing, increased scalability, and increased reliability. It will support redundancy in order to ensure no single point of failure. The new design will be highly configurable allowing tuning based on many different aspects. Discussions have included the possibility of proxy users in order to minimize administration issues. Availability of resources that can satisfy the jobs requirements will be performed shortly after job submission.

This new scheduler will build upon the experiences and lessons learned from the NASA Metacenter, the NAS/HSP Cray Cluster, the DoD Origin2000 cluster, and the DoD MetaQueue/Distributor model. Each of these models have requirements and designs unique to their user community. Addressing these requirements for a combined model will bring multi-site, heterogeneous computing one step closer to implementation.

6.0 Conclusion

The NASA Metacenter Project began by pulling together unique systems and sites with considerable coordination and some new software to hold it together. What emerged was a combined resource that provided scientists greater availability of computing cycles. This environment, although somewhat unfriendly, allowed them to do more science. The overriding goal of the next phase of the NASA Metacenter Project is to make that environment more friendly, so that scientists can concentrate more on science and less on how to get their jobs to run. Increasing the available types of resources, maintaining their expected environments at their local sites, and making the back-end as transparent as possible are all steps in creating this truly usable environment for scientists.

7.0 Additional Information

For more information on some of the work listed above, please refer to the following web pages.

The NASA Metacenter (under construction)
http://parallel.nas.nasa.gov/Parallel/Metacenter

The Globus Project
http://www.globus.org/

The Legion Project
http://legion.virginia.edu

The NAS/HSP Cray Cluster
http://science.nas.nasa.gov/ACSF/Metacenter

PBS Distribution Site
http://pbs.mrj.com/

Appendix

The NASA Metacenter: Phase 2 Team

Cristy Brickell
Tom Crockett
Archemedes de Guzman
Steve Heistand
Ed Hook
Mary Hultquist
Kim Johnson
James Jones
Isaac Lopez
Piyush Mehrotra
Yvonne Malloy
Chuck Niggley
Jens Petersohn
Bill Petray
Karen Pischel
Karl Schilke
Ian Stockdale
Geoff Tennille
Judith Utley
Rita Williams
Sarita Wood
Lou Zechtzer

THE PROGRAMMING ENVIRONMENT ON A BEOWULF CLUSTER

Phil Merkey, Donald Becker, Erik Hendriks
CESDIS/USRA

The Beowulf Project at Goddard Space Flight Center is a software development project which focuses on enhancements to the Linux operating system and a software support for high speed networks to support cluster computing. Making this software freely available on the Web contributes to the NASA Beowulf project's goals of providing cost effective, high-performance computational resources in the form of cluster computers built entirely from PC-marketplace COTS components. Several large clusters have been constructed that have obtained sustained performance of 10 Gflops. However, most systems being installed at NASA sites, government labs and universities are modest systems of 16 to 64 processors and are being constructed to address the computational requirements of an increasing diverse community. This talk will focus on the programming environment provided by a Beowulf cluster and will argue that programming a Beowulf cluster requires an equivalent level of effort as programming a vendor supplied MPP.

A cluster computer is more tightly coupled than a network of workstations. This enables configurations that are easier to maintain and easier to program. The software for Beowulf clusters is based on the integration of freely available system software and programming tools. As such the programmer is presented with a generic, portable, vendor independent multiprocessor; Beowulf clusters support the two most popular message passing packages PVM and MPI along with BSP and a distributed shared memory package developed at Goddard. Even though MPI and PVM span the space between NOWs and MPPs, key parameters such as processor speed, task granularity, communication latency and bandwidth affect performance tuning and scaling. The system software and programming tools determine the user's perception of the multiprocessor as a single machine. Together these makeup the "programming environment". This talk addresses these different programming models and compares them to their counterparts on a tightly coupled MPP and to a loosely coupled NOW. In addition, the pragmatic issues of program development, portability, debugging, scaling and performance tuning will be discussed. These issues will be considered from an application programmer's perspective with illustrations drawn from experiences obtained by working with the Earth and spaces sciences applications at Goddard.

5/3-61
018195

MULTITHREADED PROGRAMMING IN EARTH - MEETING THE CHALLENGES OF HIGH PERFORMANCE COMPUTING

366890

6P.

Gerd Heber, CAPSL, University of Delaware, 140 Evans Hall, Newark, DE 19716, 302-831-3276

Rupak Biswas, MRJ/NASA Ames Research Center, MS T27A-1, Moffet Field, CA 94035, 650-604-4411

Guang R. Gao, CAPSL, University of Delaware, 140 Evans Hall, Newark, DE 19716, 302-831-8183

1. The Challenges

A key problem in the design and use of modern high performance computer architectures is *latency*: how many CPU-cycles does it take to fetch a datum from a local or remote (involving a network device) memory, or to synchronize two activities in different nodes? This is a serious issue in getting performance out of large-scale machines with deep memory hierarchies and a large number of processing nodes. Fine-grain multithreaded architectures [8,9,10,11] strive to hide this latency through rapid switching between different threads of computation. Another challenge is to increase the number of instructions which can be issued per cycle in a modern superscalar processor. Superscalar techniques (branch prediction, speculative execution, data/dependence speculation, out-of-order execution, register renaming, etc.) require a tremendous amount of complex hardware and it is questionable whether a sustained issue rate of two instructions per cycle justifies the hardware investments. On the other hand, as pointed out for example in [1,2], it is also not clear how a compiler can make the best use of these hardware capabilities. In addition to its *latency hiding* capabilities, multithreading can serve as a basis for a multiple-issue uniprocessor as it has been proposed recently in [1,2].

2. The EARTH Programming Model

In the EARTH (*Efficient Architecture for Running THreads*) [6,7] programming model, threads are sequences of instructions belonging to an enclosing function. Threads always run to completion – they are non-preemptive. Synchronization mechanisms are used to determine when threads become executable (or ready). Although it is possible to spawn a thread explicitly, in most cases a thread starts executing when a specified *synchronization slot* reaches zero. A synchronization slot counter is decremented each time a synchronization signal is received. In a typical program, such a signal is received when some data become available. Besides the counter, a synchronization slot holds the identification number, or *thread id*, of the thread that is to be started when the counter reaches zero. This mechanism permits the implementation of dataflow-like firing rules for threads (a thread is enabled as soon as all data it will use are available). Key features of the EARTH model are:

- Split-phase communication/synchronization operations designed for variable as well as unpredictable latencies
- Support for different levels of parallelism (fine, medium, coarse)

〈 Efficient thread-level dynamic load balancing.

2.1 The EARTH Architecture

An EARTH computer consists of a set of EARTH nodes connected by a communication network. Each EARTH node has an *Execution Unit* (EU) and a *Synchronization Unit* (SU) linked to each other by queues (see Figure 1). The EU executes active threads while the SU handles the synchronization and scheduling of threads as well as communication with remote processors.

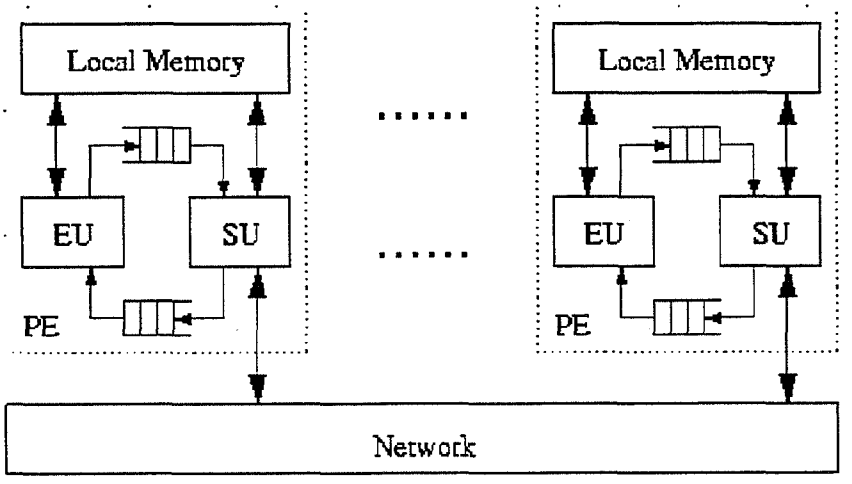


Figure 1: The EARTH architecture model.

The function of the queues shown in Figure 1 is to buffer the communication between the EU and SU. The *ready queue,* written by the SU and read by the EU, contains a set of threads ready to be executed. The EU fetches a thread from the ready queue whenever the EU is ready to begin executing a new thread. The *event queue*, written by the EU and read by the SU, contains requests for synchronization events and remote memory accesses, generated by the EU. The SU reads and processes these requests. Requests from the EU for remote data can go directly to the network or go through the local SU; implementation constraints will determine the best mechanism, so this is not defined in the model.

To assure flexibility, the EARTH model does not specify a particular instruction set. Instead, ordinary arithmetic and memory operations use whatever instructions are native to the processor(s) serving as the EU. The EARTH model specifies a set of EARTH operations for synchronization and communication. These operations are mapped to native EU instructions according to the constraints of the specific architecture. For instance, on a machine with ASIC SU chips, the EU EARTH instructions would most likely be converted to loads and stores from/to memory-mapped addresses that would be recognized and intercepted by the SU hardware.

To maximize portability, the EARTH model makes minimal assumptions about memory addressing and sharing. An EARTH multiprocessor is assumed to be a distributed memory machine in which the local memories combine to form a global address space. Any node can specify any address in this global address space. However, a node cannot read or write a non-local address directly. Remote addresses are accessed with special EARTH operations for remote access. A remote load is a *split-phase transaction* with two phases: *issuing the operation* and *using the value returned.* The second phase is performed in another thread, after the load has been completed.

2.2 Threads

A thread is an atomically scheduled sequence of instructions. It can be:
〈 A parallel function invocation (*threaded function invocation*)
〈 A code sequence defined (by a *user* or a *compiler*) to be a thread.
Usually, the body of a threaded function is partitioned into several threads. A thread shares its "enclosing frame" with other threads within the same threaded function invocation. The *state* of a thread includes its instruction pointer and a "temporary register set".

When an EU executes a thread, it executes the instructions according to their sequential semantics. In other words, instructions within a thread are scheduled using an ordinary program counter. Notice that this does not preclude the use of semantically-correct out-of-order and parallel execution to increase the instruction issue rate within a thread. Both conditional and unconditional branches are only allowed to destinations within the same thread.

EARTH threads are *non-preemptive*. Once a thread begins execution, it remains active in the EU until it executes an EARTH operation to terminate the thread. If the CPU should stall (e.g., due to a cache miss), the thread will not be swapped out of the EU. There are no mechanisms to check that data accessed by an executing thread is actually valid or to suspend the thread if it is not, except for normal register checks such as register scoreboarding. Therefore, data and control dependences must be checked and verified *before* a thread begins execution. This is done explicitly using *synchronization slots* and *synchronization signals*. A sync signal is used by the producer of a datum to tell the consumer that the datum is ready. A sync slot is used to coordinate the incoming sync signals, so that a consumer knows when all required data are ready. Each sync signal is directed to a specific sync slot. Sync signals and slots are handled with explicit EARTH operations, and are made visible in the Threaded-C language.

3. EARTH Implementations and the Threaded-C language

Some multithreaded systems, like the TERA [11] machine, have direct hardware support and offer a basic "thread instruction set". The current implementations of the EARTH architecture, on the other hand, use off-the-shelf RISC processors which represents a trade-off between the availability of systems, the flexibility of software simulation, and the size of a typical research budget. The platforms with EARTH implementations available include the MANNA machine, IBM SP2, and Beowulf workstation clusters. A port for a SMP cluster of UltraSPARC is in preparation. Generally, there is a trade-off between portability and efficiency. An EARTH implementation with its communication layer based on TCP/IP is certainly portable. However,

due to the overhead of TCP/IP and an operating system in general, the efficiency of such a system will be rather poor.

The programming language used for EARTH systems, Threaded-C [3], is ANSI-C with extensions for thread generation, execution, synchronization, and invocation of parallel threaded functions. Threaded-C also can serve as the target language for higher level languages like EARTH-C. The core of the EARTH operations supported by Threaded-C consists of the following:

- Thread synchronization: SPAWN, SYNC, INCR_SYNC
- Data transfer & synchronization: DATA_SYNC
- Split-phase data requests: GET_SYNC, BLKMOV_SYNC
- Function invocation: INVOKE, TOKEN

Reference [3] is a good introduction to programming in Threaded-C. From a programmers point of view the Threaded-C language resembles more to a processor instruction set, the EARTH instruction set, than a library. This gives the user the necessary flexibility to write efficient code and qualifies the language as a target language for compilers.

4. The Programmers' View

Language shapes the way we think, and determines what we can think about. (B.L.Whorf)

Many applications in high-performance computing demonstrate a lot of parallelism at the algorithmic level. However, the restricted expressiveness of the programming language and architectural constraints (e.g., locality, bandwidth, distributed memory) force the user to abandon a considerable amount of that parallelism and introduce a lot of unnecessary synchronization. Standard APIs like MPI or OpenMP are oriented towards rather coarse-grained and regular parallelism, and are of limited use in addressing issues like the latency problem. On the other hand, languages with support for fine-grained multithreading allow for a considerable relaxation of synchronization, and support different granularity levels. Threaded-C offers the programmer a (logically) global address space, but does not support the *illusion* of uniform and predictable access times. Communication (= remote memory access) is one-sided and asynchronous. Therefore, the programmer has not to worry about matching sends and receives in his code and there is no need for polling.

5. Applications

Reference [5] gives a good overview and a detailed analysis of costs for fine-grained multithreading with off-the-shelf hardware. A so-called EARTH benchmark suite (EBS) has been assembled and is used for evaluation purposes. Table 1 lists some of the benchmarks contained in the EBS and the speedups achieved on EARTH-MANNA. The code size of the benchmarks ranges between 70 and about 20,000 lines. The first nine programs are implemented in Threaded-C and the remaining are written in EARTH-C, which uses Threaded-C as a target language.

Benchmark	Problem Size	Problem Domain	2 nodes	4 nodes	8 nodes	16 nodes
Ray Tracing	512x512	Image Processing	1.99	3.96	7.90	15.46
Protein Folding	3x3x3 Cube	Biochemistry	1.89	3.61	7.29	14.51
TSP	10	Graph Searching	1.95	3.90	7.79	15.57
Tomcatv	257	SPEC Benchmark	2.14	4.19	8.08	15.38
Wave2D	150x150	CFD	1.98	3.83	6.24	10.02
Matrix Multiply	512x512	BLAS	1.86	3.70	7.39	14.90
N-Queens	12	Graph Searching	2.09	4.18	8.36	16.63
Paraffins	20	Chemistry	1.82	3.52	7.10	13.50
Cvfem	5381 nodes	Heat Flow	1.92	3.65	6.91	11.24
Tree-Add	1M	Graph Searching	1.88	3.77	7.51	15.01
Power	10000	Nonlinear Opt.	1.93	3.85	7.37	13.64
Voronoi	64k	Comput. Geometry	1.59	2.77	5.04	9.40
Heuristic-TSP	32k	Search Problem	2.00	3.93	7.59	13.10
PSRS	2M Int.	Parallel Sort	1.00	1.52	2.39	3.85
Barnes-Hut	1k-points	N-Body	1.00	2.06	3.68	6.63
Health	6:4 tree	Simulation	1.95	3.84	7.15	13.06
Mst	512 nodes	Graph Searching	0.41	0.83	1.92	4.01

Table 1: Sample Applications from the EBS.

One of the attractive features of the EARTH system is its capability of dynamic load balancing by thread migration or dynamic function invocation. This is a feature of the EARTH runtime system and transparent to the user. For example, on EARTH-SP2 there are eight different load balancers available, and the programmer makes his choice via a compilation flag. Among other things, we exploited this feature in the parallelization of a raytracing algorithm, and more recently, in the parallel adaption of unstructured meshes [4,5]. An unevenly distributed mesh leads to high imbalances in the workload of the numerical computation as well as in the workload for the adaption itself is imbalanced too. In an MPI style implementation the programmer has the burden of load balancing, i.e., the partitioning and remapping of the mesh. Systems with support for object and/or workload migration like EARTH offer a lot of new opportunities to handle this problem and rely on the runtime system instead of a "handcrafted" solution, which has to be redone with every new problem. Table 2 lists some sample results (execution times in ms) from a mesh adaption simulation on EARTH-MANNA and EARTH-SP2. The simulation is explained in detail in [6,7].

Machine	Size	1 node	2 nodes	4 nodes	8 nodes	16 nodes
MANNA	512x16x16	2936	1503	888	506	317
SP2	512x16x16	609	324	204	156	-
MANNA	1024x16x32	6338	3118	1726	980	619
SP2	1024x16x32	1234	651	401	289	-
MANNA	2048x32x32	15332	6729	3639	1951	1138
SP2	2048x32x32	2459	1295	793	556	-
MANNA	4096x32x64	36847	14459	7346	4096	2438
SP2	4096x32x64	4834	2585	1576	1118	-
MANNA	8192x64x64	106615	35291	18499	9416	5816
SP2	8192x64x64	9890	5288	3170	2334	-

Table 2: Sample results from a mesh adaption simulation [6,7].

Our previous and ongoing experiments show that the EARTH system is a promising approach in order to make a large class of irregular and dynamic applications accessible to high performance computing in an efficient manner.

5. Current Research

In our current research at CAPSL (Computer Architecture and Parallel Systems Laboratory) we try to address a whole spectrum of topics related to hard- and software support for fine-grained multithreading. Among these are in the field of fine-grained architecture/compiler support:

- Simulation and design of a hardware SU
- Superstrand Architecture [1,2]
- Studies in Instruction Level Parallelism
- Compilation for fine-grained multithreaded systems.

The existing ports of EARTH are for distributed memory machines. The design of an EARTH system for shared memory and DSM machines is one of our ongoing projects. Ports of EARTH for the following platforms are in progress or preparation:

1. A cluster of 20 UltraSPARC SMP interconnected by FastEthernet and Myrinet at UDel
2. The PowerMANNA [13], a PowerPC620 based research machine provided by GMD FIRST, Berlin, Germany.

Our multithreaded application studies deal with adaptive unstructured grid methods, signal processing transforms (e.g, wavelets), and radiocity.

6. References

1. A. Marquez, K.B. Theobald, X. Tang, G.R. Gao: *A Superstrand Architecture*, CAPSL Technical Memo 14, University of Delaware, 1997.
2. A. Marquez, K.B. Theobald, X. Tang, T. Sterling, G.R. Gao: *A Superstrand Architecture and its Compilation*, CAPSL Technical Memo 18, University of Delaware, 1998.
3. K.B. Theobald, J.N. Amaral, G. Heber, O. Maquelin, X. Tang, G.R. Gao: *Overview of the Threaded-C Language*, CAPSL Technical Memo 19, University of Delaware, 1998.
4. G. Heber, R. Biswas, P. Thulasiraman, G.R. Gao: *Using Multithreading for the Automatic Load Balancing of Finite Element Meshes*, In Proceedings of Irregular'98, Berkley, Lecture Notes in Computer Science 1457, pp.132-143, Springer, 1998.
5. G. Heber, R. Biswas, P. Thulasiraman, G.R. Gao: *Using Multithreading for the Automatic Load Balancing of 2-D Adaptive Finite Element Meshes*, CAPSL Technical Memo 20, University of Delaware, 1998.
6. H.H.J. Hum, O. Maquelin, K.B. Theobald, X. Tang, G.R. Gao, L. Hendren: *A Study of the EARTH-MANNA Multithreaded System*, International Journal of Parallel Programming Vol. 24, No. 4, August 1996.
7. H.H.J. Hum et al.: *A Design Study of the EARTH Multiprocessor*, In Proceedings of the PACT95, pp.59-68.
8. H.H.J. Hum, K.B. Theobald, G.R. Gao: *Building Multithreaded Architectures with Off-the Shelf Microprocessors*, In Proceedings of IPPS'94, pp. 288-294.

AN EVALUATION OF AUTOMATIC PARALLELIZATION TOOLS

Michael Frumkin, Michelle Hribar, Haoqiang Jin, Abdul Waheed, Jerry Yan
MRJ Technology Solutions, Inc., NASA Ames Research Center
MS T27A-2, Moffett Field, CA 94035-1000
{frumkin, hribar, hjin, waheed, yan}@nas.nasa.gov
(650) 604-2782

Abstract. Porting applications to new high performance parallel and distributed computing platforms is a challenging task. Since writing parallel code by hand is time consuming and costly, ideally, porting codes would be automated by using some parallelization tools and compilers. In this paper, we evaluate three parallelization tools and compilers: 1) CAPTools: an interactive computer aided parallelization tool that generates message passing code, 2) the Portland Group's HPF compiler and 3) using compiler directives with the native FORTAN77 compiler on the SGI Origin2000. We use the NAS Parallel Benchmarks and a computational fluid dynamics application, ARC3D.

1 Introduction

High performance computers have evolved rapidly over the past decade. At NASA Ames Research Center, our scientists expend a very large effort porting codes in an attempt to fully utilize the performance potential of each new high performance architecture we acquire. Furthermore, NASA is currently working on an Information Power Grid Initiative to produce a computational grid that will work in concert with computational grids being assembled at PACI [15, 16]. In anticipation of a widely distributed and heterogeneous computing environment, together with increasing complexity and size of future applications, we will not be able to afford to continue our porting efforts every three years. Instead, in order to protect investments in code maintenance and development, the parallelization process needs to 1) require less time and effort, 2) generate codes with good performance, 3) be able to handle all type of aerospace applications of interest to NASA and 4) be portable to new machines and the Information Power Grid.

Currently, there are three major approaches besides hand-coding for porting applications to parallel architectures:

1. Using semi-custom building blocks (e.g., *PETSc* [2], *NHSE* software [14]);

2. Data parallel languages and parallelizing compilers (e.g., *HPF* [7], *FORTRAN-D* [1], *Vienna FORTRAN* [3], *ZPL* [19], *pC++/Sage++* [10], *HPC++* [9]); and

3. Computer aided parallelization tools and translators (e.g., *KAP/Pro Tool-Set* [13], *SUIF* [20], *FORGExplorer* [12] and *CAPTools* [6]).

Given the aforementioned alternatives to writing parallel programs by hand, the effectiveness of the different approaches is a worthwhile study, especially for the aerospace applications that are of interest to us. We initiated a careful comparison of the effectiveness of the alternative parallelization approaches using the NAS Parallel Benchmarks and a computational fluid dynamics (CFD) application, ARC3D, as the initial test suite. We use the SGI Origin 2000, a distributed shared memory computer which is the most recent acquisition at NASA Ames. We decided to first evaluate three parallelization tool/compilers: 1) CAPTools[6], a computer aided translator tool for message passing code, 2) Portland Group's HPF compiler [11], a data parallel language, and 3) the usage of compiler directives available with the native Fortran77 compiler on the Origin 2000 [18]. We did not use the semi-custom building blocks since that approach would require rewriting the applications in terms of pre-defined library functions.

In this study, we evaluate the different approaches in four areas: 1) user interaction, 2) limitations, 3) portability and 4) performance. Berthou and Colombet [4] performed a similar comparison of an automated parallelization tool, a data parallel language, and hand-written message-passing code for two applications on the Cray T3D. They found that the message passing code had by far the best performance. In our study, we use a more powerful parallelization tool and a wider variety of test programs. We find that some of the automated approaches can achieve very good performance and show promise for our parallelization needs at NASA. Furthermore, we consider other factors than just performance in our evaluation.

2 Comparison of Parallelization Approaches

In this study, we evaluate the three parallelization approaches for our test suite on the SGI Origin 2000. We tested each approach for four of the NAS Parallel Benchmarks and for a CFD application, ARC3D. The NAS Parallel Benchmarks [5] are derived from CFD codes and are used to compare the performance of highly parallel computers. Version 2 of the benchmarks are hand-written codes using FORTRAN 77 and MPI. Four serial versions (LU, SP, BT and FT) of the benchmarks from the latest release, NPB2.3 are the starting point for our study. ARC3D [17] is a well-known, moderate size CFD application. It is similar in structure to SP, but is a more realistic application.

We evaluate each of the approaches in four areas: 1) user interaction, 2) limitations, 3) portability and 4) performance. Because of space constraints, we can only briefly address each area; for the full results, please see [8].

2.1 User Interaction

The type of user interaction required by each automated approach is different. Here, we briefly describe each approach and the user interaction required.

First, the automated translating tool, CAPTools, requires that the user guide the parallelization process through graphical interfaces. The software performs dependency analysis automatically and requires only that the user decide which arrays to partition and in which dimensions. The code is automatically parallelized based on the dependency analysis and array partitioning. The tool also provides additional capabilities to help tune the code. Generating an initial code with CAPTools does not require much effort; choosing arrays to partition is the only required decision. Tuning the code requires more effort. The user must remove unnecessary dependences and experiment with different partitioning techniques and communication schemes in order to achieve parallel code with good performance. With the benchmarks and ARC3D, the tuning process took a few weeks for each code.

HPF, the data parallel language, also relies on data distribution to achieve parallelism; the user adds directives to the serial code which distribute arrays. Once the arrays are partitioned, the loops are automatically parallelized by the compiler based on these data distributions. This is similar to the CAPTools tool; however, the user must enter these directives by hand in HPF instead of with mouse clicks. The user can also add independent directives to force the parallelization of loops that the compiler cannot do automatically. Finally, HPF provides an array syntax for array assignments that is more efficient than using loops for array assignments. Inserting these directives and array assignments into serial code by hand can be quite time consuming. For this reason, generating an initial parallel code using HPF requires the most time of all the approaches. Tuning the code can be more difficult as well; the HPF model hides details from the user. Nevertheless, we found that the entire time to program and tune a code in HPF took a few weeks.

Using the compiler directives on the Origin 2000 is similar to HPF; however, it provides directives to parallelize loops instead of distributing data. In our tests, we used native tools available on the Origin 2000 to help with the insertion of directives. First, we used the Power FORTRAN Accelerator (PFA) which automatically inserts parallelization directives into serial code. We then used

the Parallel Analyzer View (PAV), a graphical tool, to present the results of the automated parallelization performed by PFA. We then performed further tuning by inserting directives by hand to parallelize loops and distribute data. Generating an initial parallel code requires just minutes; however, tuning the test codes took several weeks. We found that we could achieve reasonable code with little effort; however, we will show that with much more effort, we could achieve code with excellent performance.

In summary, while the approaches differ in the amount of time required to generate an initial code, they all required a few weeks to tune each code. In comparison, it took several months to generate parallel code by hand.

2.2 Limitations

Each approach has some limitations. First, the version of CAPTools that we tested cannot handle serial code with unstructured grids and indirect array accesses. These characteristics are usually present in large CFD applications. Furthermore, CAPTools requires that the data distribution remain static during the course of the application. For this reason, CAPTools cannot effectively parallelize FT. The most efficient way to parallelize the code is to transpose the data so that the FFT calculation in FT is never performed over a partitioned dimension. CAPTools, however, cannot change the data distribution during the course of the application.

HPF requires that loops that are performed in parallel have no dependences between processors' sections of data. This means that the pipelined computation in SP, BT and LU can not be expressed as parallel loops in HPF. Instead, the data is redistributed so that the pipelined computation's data dependences are contained within the processor. For SP, BT and LU, the communication required for the pipelined computation is much less than for the data redistribution. For this reason, HPF is not the most effective way to parallelize these codes.

The Origin compiler directives can be used to parallelize any code; however, it is not always effective for all codes. For example, adding compiler directives did not result in reasonable code for LU. The main loop in LU has dependences in multiple dimensions which prevented it from being parallelized.

2.3 Portability

Since the goal of our research is to avoid hand-porting code to new machines, portability of the resulting codes is an important issue. CAPTools' communication is performed by calls to a library. This library is machine-dependent and can be based on MPI, PVM or machine-specific communication calls or directives. Porting this library to different machines is relatively easy. One of the purported strengths of HPF is that it is based on a standard and consequently portable. Currently, there is an HPF compiler for most major high performance systems, but whether this will remain true in the future depends on the success and acceptance of this language. Finally, the compiler directives on the Origin 2000 are not portable. However, there are directives based on the OpenMP standard which could be used instead of the native directives. This would potentially allow the code to be portable to other shared memory systems.

2.4 Performance

Figure 1 compares the execution time of the three automated approaches (CAPTools, SGI-pfa and PGHPF) to the performance of the hand-coded benchmark (NPB-2.3). Figure 2 provides the comparison for ARC3D. Additionally, in Figure 2, there is a comparison for SP that includes a an additional comparison labeled "direct". This corresponds to hand-optimizing the Origin 2000 compiler directives code for better cache performance. We provide this version as a means of comparison, but because it requires much more work, it is more like hand-coding than automated parallelization.

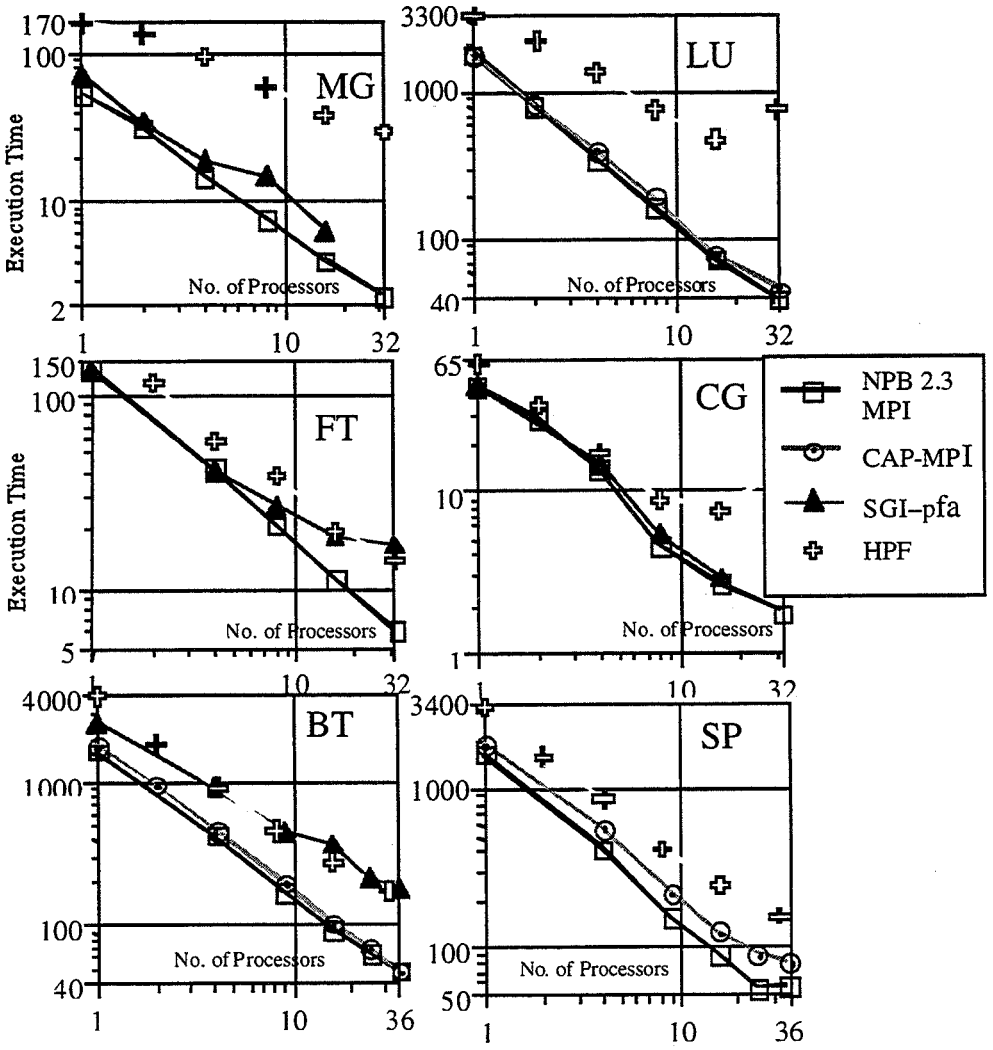


Figure 1: Performance Comparison for 4 Benchmarks

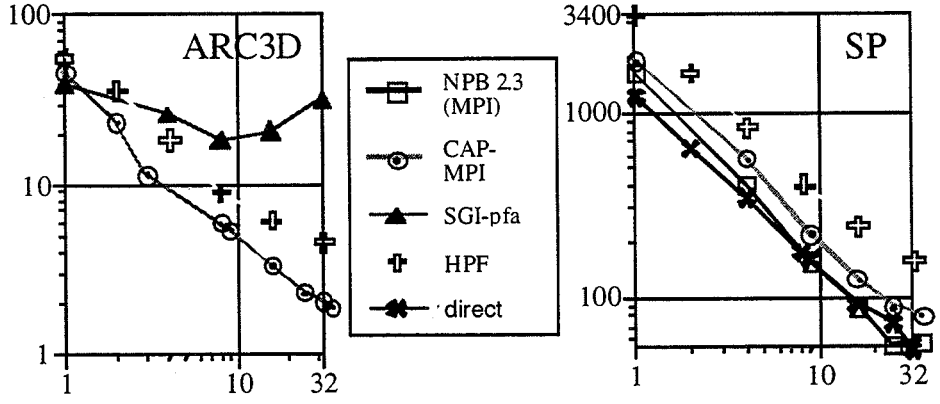


Figure 2: Performance Comparison for ARC3D and SP

In general, we note the following trends:

- CAPTools consistently produces code with the best performance of the three automated approaches. Nevertheless, it is unable to parallelize FT, MG and CG.

- The HPF code has the worst performance for almost all cases. Its performance on a single processor is still over twice as slow as the other approaches, signifying that part of the reason for its relatively poor performance is the compiler. Furthermore, because it must use data transposition for LU, BT, SP and ARC3D instead of parallel pipelined computation and communication, the performance is worse than the other versions.

- The Origin compiled code has good performance in some cases. For BT, we used the automated tool and did some tuning to force parallelization of the loops. The performance of the resulting code is reasonable. However, for SP, when we used just the automated compiler, the performance was poor. When we did many more optimizations by hand (direct version), the performance was better than the hand-coded NPB2.3 as shown in Figure 2. These optimizations optimized the cache usage and improved the serial performance of the code as well.

- The good performance of the cache optimized code (direct) emphasizes the importance of serial optimizations. The original serial test codes were not written with cache performance in mind; therefore, their performance, both serial and parallel on the Origin 2000 is not as good as it could be. More work is being done on incorporating these cache optimizations into the generated parallel codes.

In summary, CAPTools had the best performance of the automated approaches for the applications it could parallelize. The compiler directives approach has even better performance, however, if the user performs cache optimizations by hand.

3 Conclusions and Future Research

Comparing CAPTools, HPF and the Origin compiler directives has provided some insight for automating parallelization of aerospace applications. While all of the approaches are considered "automated" they still require user input/tuning to generate reasonable code. In summary, HPF requires the most work for the least payoff in terms of performance. CAPTools requires reasonable user input and generates code with good performance, but it cannot currently handle some features that are integral to most CFD applications. Finally, the compiler directives can achieve great performance, but with a lot of effort and without a guarantee of portability.

More positively, while the study provides an interesting comparison of the different approaches, the preliminary results show that automated parallelization approaches do exhibit promise for our needs at NASA. In a relatively short amount of time we were able to generate parallel codes with reasonably good performance. These initial results are part of a more extensive study of evaluating parallelization tools and compilers for aerospace applications at NASA. We are extending this study to incorporate the evaluation of other parallelization tools/compilers (e.g. *SUIF* [20], the D System [1], other architectures (CRAY T3E and network of workstations), and other more complex applications of interest to NASA (e.g. OVERFLOW). The results of some of these experiments will be reported in an upcoming paper.

4 References

[1] Vikram S. Adve, John Mellor-Crummey, Mark Anderson, Ken Kennedy, Jhy-Chun Wang, and Daniel Reed, "An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs," presented at Supercomputing '95, San Diego, CA, 1995.

[2] Satish Balay, Bill Gropp, Lois Curfman McInnes, and Barry Smith, "PETSc Library" . Argonne National Laboratory: http://www.mcs.anl.gov/petsc/petsc.html.

[3] S. Benkner, "Vienna FORTRAN 90- An Advanced Data Parallel Language," presented at International Conference on Parallel Computing Technologies (PACT-95), St. Petersburg, Russia, 1995.

[4] Jean-Ives Berthou and Laurent Colombet, "Which Approach to Parallelizing Scientific Codes-That is the Question," *Parallel Computing*, vol. 23, pp. 165-179, 1997.

[5] NAS Division NASA Ames Research Center, "NAS-Parallel Benchmarks" , 2.3 ed. Moffett Field, CA: http://science.nas.nasa.gov/Software/NPB, 1997.

[6] M. Cross, C.S. Ierotheou, S.P. Johnson, P. Leggett, and E. Evans, "Software Tools for Automating the Parallelisation of FORTRAN Computational Mechanics Codes," *Parallel and Distributed Processing for Computational Mechanics*, 1997.

[7] High Performance FORTRAN Forum, "High Performance FORTRAN Language Specification Version 1.0," *Scientific Programming*, vol. 2, 1993.

[8] Michael Frumkin, Michelle Hribar, Haoqiang Jin, Abdul Waheed, and Jerry Yan, "A Comparison of Automatic Parallelization Tools/Compilers on the SGI Origin2000," presented at to appear in SC98, Orlando, FL, 1998.

[9] Dennis Gannon, Peter Beckman, Elizabeth Johnson, Todd Green, and Mike Levine, "HPC++ and HPC++ Lib Toolkit," Indiana University, Bloomington, IN, 1997.

[10] Dennis Gannon, Shelby X. Yang, and Peter Beckman, "User Guide for a Portable Parallel C++ Programming System pC++," Department of Computer Science and CICA, 1994.

[11] Portland Group, "PGHPF" : http://www.pgroup.com.

[12] Applied Parallel Research Inc., "FORGE Explorer" : http://www.apri.com.

[13] Kuck and Associates Inc., "Parallel Performance of Standard Codes on the Compaq Professional Workstation 8000: Experiences with Visual KAP and the KAP/Pro Toolset under Windows NT," , Champaign, IL.

[14] NHSE, "HPC-Netlib Software Catalog" : http://nhse.cs.utk.edu/rib/repositories/hpc-netlib/catalog.

[15] NPACI, "http://www.npaci.edu/Research" .

[16] PACI, "http://www.cise.nsf.gov/acir/paci.html" .

[17] T.H. Pulliam, "Solution Methods in Computational Fluid Dynamics," in *Notes for the von Karman Institute for Fluid Dynamics Lecture Series*. Belgium: Rhode-St-Genese, 1986.

[18] Inc. Silicon Graphics, "MIPSpro FORTRAN 77 Programmer's Guide" . http://techpubs.sgi.com/library/dynaweb_bin/0640/bin/nph-dynaweb.cgi/dynaweb/SGI_Developer/MproF77_Pb/@Generic_BookView.

[19] Lawrence Snyder, "A ZPL Programming Guide," University of Washington, 1998.

[20] Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W.K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, Mary W. Hall, Monica Lam, and John Hennessy, "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers," Computer Systems Laboratory, Stanford University, Stanford, CA.

AN EXPERIMENT IN SCIENTIFIC CODE SEMANTIC ANALYSIS

Mark E. M. Stewart
Dynacs Engineering, Inc.
2001 Aerospace Parkway
Brook Park, OH 44142
(216) 977-1163
Mark.E.Stewart@lerc.nasa.gov

Abstract

This paper concerns a procedure that analyzes aspects of the meaning or semantics of scientific and engineering code. This procedure involves taking a user's existing code, adding semantic declarations for some primitive variables, and parsing this annotated code using multiple, distributed expert parsers. These semantic parsers are designed to recognize formulae in different disciplines including physical and mathematical formulae and geometrical position in a numerical scheme. The parsers will automatically recognize and document some static, semantic concepts and locate some program semantic errors. Results are shown for a subroutine test case and a collection of combustion code routines. This ability to locate some semantic errors and document semantic concepts in scientific and engineering code should reduce the time, risk, and effort of developing and using these codes.

Introduction

From a syntactic or programming language perspective, scientific programs are uses of a programming language that specify how numbers are to be manipulated. However, from the perspective of semantics or meaning, scientific programs involve an organization of physical and mathematical equations and concepts. The programs from a wide range of scientific and engineering fields use and reuse these fundamental concepts in different combinations. This paper explains an experiment in representing, recognizing, and checking these fundamental scientific semantics.

What motivates this experiment is that scientific code semantics is a central issue in the checking and documentation of scientific code. Reducing the errors in a scientific or engineering program until its results are trusted involves ensuring the program's semantics are correct. Further, this debugging process is expensive and time consuming because it is primarily a manual task. The existing software development tools (lint, ftnchek, make, dbx, sccs, call tree graphs, memory leak testing) do not fully solve this problem and deal only superficially with semantics. Further, verification techniques (comparison with available analytic and experimental results, verification of convergence and order of accuracy) can only detect the presence of an error, and finding this error often leads to a time-consuming manual search.

Similarly, the traditional tools for program documentation (suggestive variable names, program comments, and program manuals) are often not adequate. Understanding another programmer's code is usually frustrating and time consuming even with good documentation. Further, having confidence in a code requires a large time investment.

Structured programming addresses both of these problems. Currently software reuse through subroutine libraries and object-oriented programming[1] also targets these problems, but cannot help when modifications and custom software are required. Recently there has been work in high-level specification languages[2] where a symbolic manipulation program (Maple, Mathematica) is used to write subroutines or even programs. However, high-level specification languages are not applicable to legacy code and require substantial changes in software development practices.

Ontologies have been developed in other fields. In natural language understanding, ontologies have been developed to represent written text.[3] Using an ontology for engineering knowledge representation and tool integration has also been studied.[4]

The current experiment was conducted because of the limitations of these existing tools and approaches. As a complementary tool, automated semantic analysis[5] could reduce the time, risk, and effort during original code development, subsequent maintenance, second party modification, and reverse engineering of undocumented code.

Thesis

The principal thesis of this semantic analysis experiment is that fundamental physical and mathematical formulae and concepts are reused and reorganized in scientific and engineering codes. Further, a parser[6,7,8] can recognize each reuse.

Method

In outline, the current method for testing this thesis consists of four key stages. First, the user adds semantic declarations to their existing program (A.1).

```
C?  M == MASS
C?  ACC == ACCELERATION
    FF = M*ACC                                  (A.1)
```

These declarations provide the mathematical or physical identity of primitive variables in the user's program. Second, the procedure syntactically parses the user's program into a data structure representation. Third, a translation scheme converts parts of the user's program into phrases in different context languages, for example, (A.1) is converted to its physical dimensions expression (A.2), and its physical quantity expression (A.3).

```
(M) * ( LT**-2 )                                (A.2)
MASS * ACCELERATION                             (A.3)
```

These different context languages reflect the ways scientists and engineers consider program expressions, including physical formula, dimensions, units, geometrical location, geometrical axis and orientation in a grid. Fourth, distributed expert parsers examine the translated phrases and attempt to recognize formulae from their area of expertise. For example, a kinematics expert parser

```
FORCE :  MASS * ACCELERATION                    (A.4)
```

would include the rule (A.4), and be able to recognize the phrase (A.3) as "FORCE" due to Newton's law. Further, the units expert parser is able to reduce (A.2) as well as performing various dimensional tests. The other expert parsers act similarly.

When an expert parser recognizes an expression, it annotates the data structure representation of the user's program with the observation. Other expert parsers can use this observation to recognize more of the expression. Further, the annotated data structure representation contains all the results of the semantic analysis, and a graphical user interface (GUI) displays these results as shown in Figure 1. The user may point to variables and expression in his/her code and any semantic interpretation and its derivation are displayed. The GUI will highlight recognized errors, undefined quantities, and unrecognizable expressions. Further, the GUI provides detailed scientific and technical definitions and explanations.

The expert parsers can recognize more than expressions involving declared and derived variables. Unique constants such as 3.1415, 459.67, 6.626196E-34 or 1716.5 are recognized without

this subroutine. Other recognized formulae include temperature formulae, viscosity and thermal conductivity calculated from the power law, Reynolds number and Prandtl number.

Twenty ALLSPD subroutines (5500 FORTRAN statements) form an additional test case where an understanding fraction of 0.51 has been achieved. This level of understanding was achieved after adding rules and infrastructure to the procedure, and debugging it with the ALLSPD routines as test data. Additional work will yield higher levels of understanding.

Conclusions

The current results for this semantic analysis procedure demonstrate potential and provide partial proof that the method can be a general tool for semantic analysis and code documentation. The ALLSPD results do not fully prove the generality of the method because different programs use different subsets of the fundamental physical and mathematical rules. Consequently, the level of understanding will vary from code to code at least until a comprehensive set of fundamental rules can be incorporated into the semantic analysis procedure.

To fully develop this potential will require additional work. In particular, additional mathematical, physical, and geometric knowledge must be added to the expert parsers. It is also important to develop the infrastructure of the method. However, both of these development directions must be pursued in tandem while proving the procedure on more scientific and engineering codes. More theoretically, there are other applications of this procedure that have not been explored, including tracking physical assumptions and analyzing more geometrical information.

Acknowledgments

This work was supported by the Propulsion Systems Base Program at NASA Lewis Research Center through the Computing and Interdisciplinary Systems Office (contract NAS3-27816). Greg Follen and Joe Veres were the monitors. The author thanks Ambady Suresh and Scott Townsend for helpful discussions about this work.

Bibliography

[1] W. Y. Crutchfield, M. L. Welcome, "Object Oriented Implementation of Adaptive Mesh Refinement Algorithms," Scientific Programming 2 (1993) 2, 145-156.

[2] E. Kant, "Synthesis of Mathematical Modeling Software," IEEE Software, May 1993.

[3] J. Allen, Natural Language Understanding (Benjamin/Cummings, Menlo Park, 1987)

[4] M. R. Cutkosky, et. al., "PACT: An Experiment in Integrating Concurrent Engineering Systems," IEEE Computer, Jan. 1993.

[5] M. E. M. Stewart, "A Semantic Analysis Method for Scientific and Engineering Code," NASA CR 207402, April 1998.

[6] A.V. Aho, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques, and Tools (Addison-Wesley, Reading, 1986).

[7] S. C. Johnson, "Yacc----Yet Another Compiler-Compiler," Comp. Sci. Tech. Rep. No. 32. (AT&T Bell Laboratories, Murray Hill, 1977).

[8] J. R. Levin, T Mason, D. Brown, Lex and Yacc (O'Reilly, Sebastopol, 1992).

[9] Shuen, J.-S., Chen K.-H. and Choi Y., "A Coupled Implicit Method for Chemical Non-equilibrium Flows at All Speeds," J. of Comp. Phys., 106, No. 2, 306, 1993.

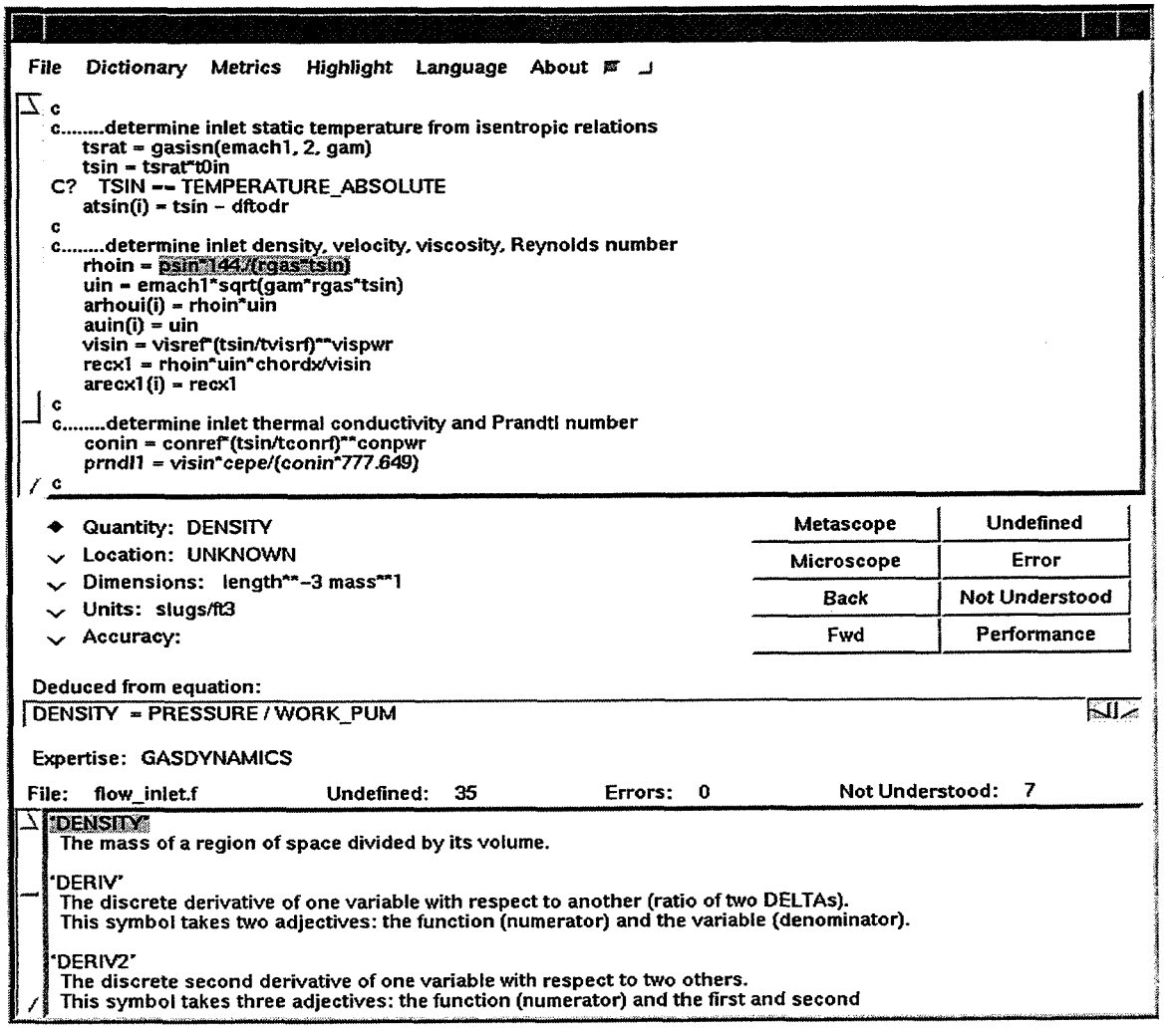


Figure 1: GUI display for the semantic analysis program. The top window displays the user's code, and variables and expressions may be selected for explanation. The middle region explains this selected text. In this case, the physical quantity is density, it does not have a grid location, and it has the displayed dimensions, units and derivation. The bottom region displays the semantic dictionary/lexicon.

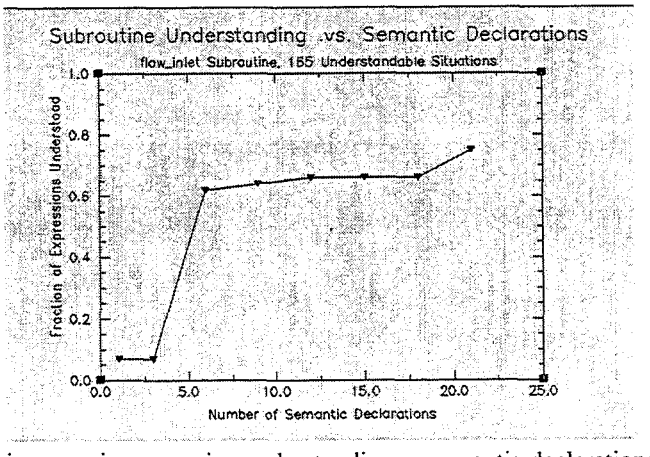
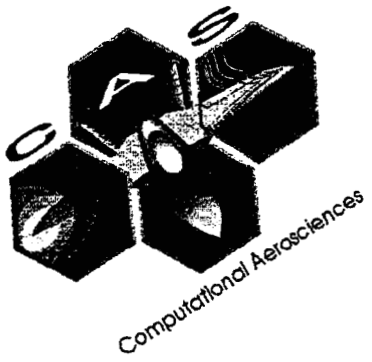


Figure 2: Graph showing the increase in expression understanding as semantic declarations are added. The subroutine contains 165 expressions to recognize and approximately 100 lines of code.

OMIT THIS PAGE



Session 4:

Applications for Parallel/Distributed Computers

516-07
018-210

ABS. ONLY

3D MULTISTAGE SIMULATION OF EACH COMPONENT OF THE GE90 TURBOFAN ENGINE

Mark Turner & Dave Topp(513)243-1182, mark.turner@ae.ge.com
General Electric Aircraft Engines

Joe Veres(216)433-2436, joe.veres@lerc.nasa.gov
NASA Lewis research Center

366873

1 P.

A 3D multistage simulation of each component of the GE90 Turbofan engine has been made. This includes 49 blade rows. A coupled simulation of all blade rows will be made very soon. The simulation is running using two levels of parallelism. The first level is on a blade row basis with information shared using files. The second level is using a grid domain decomposition with information shared using MPI.

Timings will be shown for running on the SP2, an SGI Origin and a distributed system of HP workstations. On the HP workstations, the CHIMP version of MPI is used, with queuing supplied by LSF (Load Sharing Facility). A script-based control system is used to ensure reliability.

An MPEG movie illustrating the flow simulation of the engine has been created using pV3, a parallel visualization library created by Bob Haimes of MIT. PVM is used to create a virtual machine from 10 HP workstations and display on an SGI workstation.

A representative component simulation will be compared to rig data to demonstrate its usefulness in turbomachinery design and analysis.

5/7-07
018218

366814

APPLICATION OF MULTI-STAGE VISCOUS FLOW CFD METHODS FOR ADVANCED GAS TURBINE ENGINE DESIGN AND DEVELOPMENT

S . 'Mani' Subramanian and Paul Vitt
ASE Technologies, Inc.
4015 Park Drive, Suite 203
Cincinnati, OH 45241
(513)- 563 - 8855

David Cherry and Mark Turner
GE Aircraft Engines
1 Neumann Way
Cincinnati, OH 45215
(513)- 243 - 1182

The primary objective of the research is to develop, apply and demonstrate the capability and accuracy of 3-D multi-stage CFD methods as efficient design tools on high-performance computer platforms, for the development of advanced gas turbine engines. Propulsion systems that are planned and that are currently in development for next generation civilian and military aircraft applications under NASA's Advanced Subsonic Technology (AST), DoD's Integrated High Performance Turbine Engine Technology (IHPTET) programs will be required to operate under complex flow conditions, imposed by strict performance expectations and goals. Some of the expectations and goals include higher thrust, lower emission levels, higher pressure ratios, smaller size, lower weight, fewer stages, lower fuel consumption and higher efficiency. These goals necessitate blades with high turning angles, stages with small axial gaps between blade rows, and non-axisymmetric flowpath. It becomes important to use design methods that treat the stator and rotor airfoils as a complete system for providing information regarding the influence of one blade row on the other for overall engine performance. The particular aspect of this very complex problem that is presently of interest to NASA and to US Aircraft Engine companies, and the focal point of this research is the prediction and understanding of the 3-D multi-stage interaction effects in advanced gas turbine engines. More importantly, to use the information for design optimization and performance improvements in next generation engines to power US commercial and military aircraft.

Using the HPCCP computational resources, several 3-D multi-stage aerodynamic analyses were performed for high pressure and low pressure turbine designs under flight and rig conditions. Results are presented here for a Boeing 777 class, high and low pressure turbine engine stage configuration at take-off conditions. The analysis included cooling flow addition details and effects of seal leaks through both turbine stages to realistically represent the actual engine operation. The turbine geometry consisted of 18 blade rows, that were solved simultaneously due to fully subsonic flow conditions. Using the parallel version of the average passage code and with a total of 9.4 million grid points, results were obtained using typically 16 to 60 processors. Load balancing the processors between blade rows provided good parallel efficiency. The overall agreement of the rig analysis with experimental data was very good, providing confidence in the average passage solution approach. The HPCCP computational resource was an excellent testbed for these real world simulations, and very good parallel performance efficiencies were achieved for these complex flow analyses.

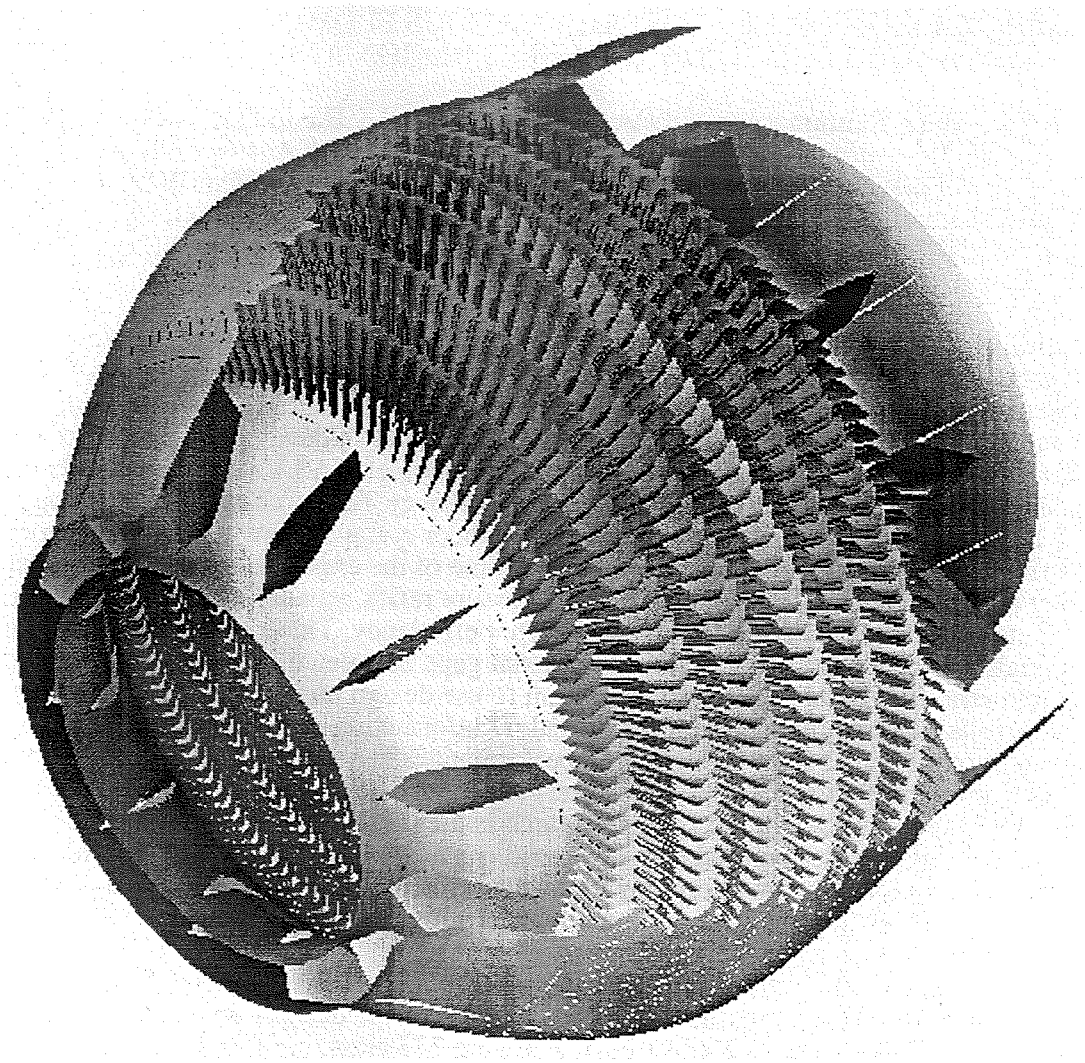


Figure 1. Turbine Geometry for Multi-Stage Analysis

The Development of a Multi-Purpose 3-D Relativistic Hydrodynamics Code

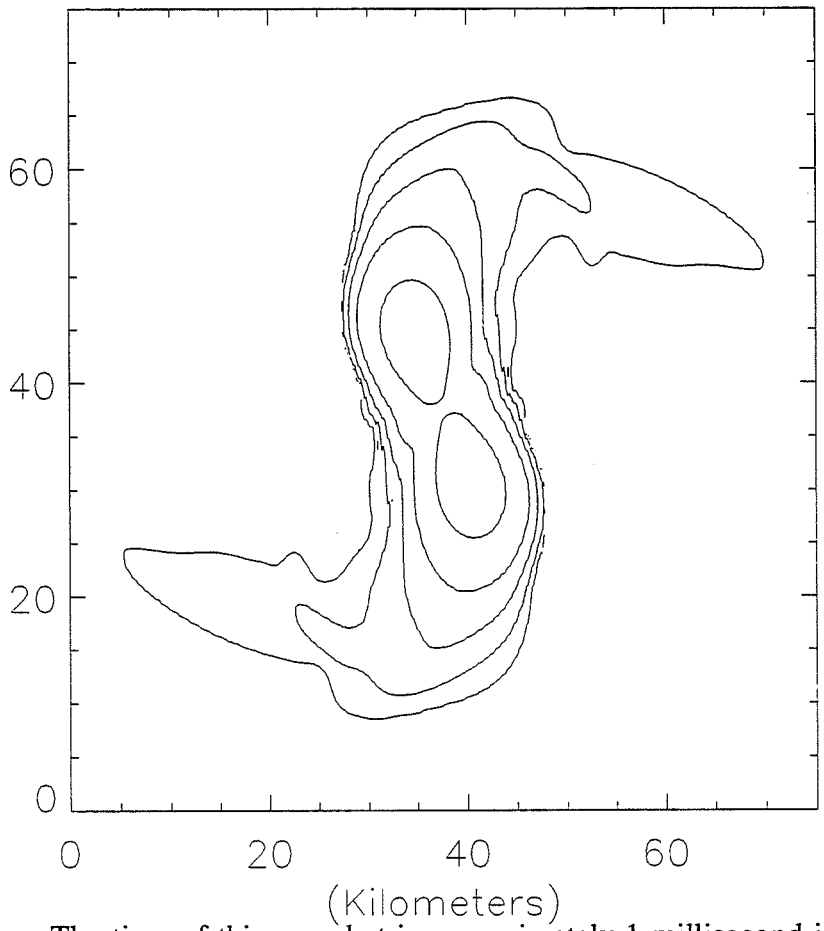
F. Douglas Swesty
National Center for Supercomputing Applications & Department of Astronomy
University of Illinois, Urbana IL, 61801
email: dswesty@ncsa.uiuc.edu phone: (217)-244-1976

Many of the most interesting phenomena in astrophysics involve gas flows around compact objects such as white dwarfs, neutron stars, and black holes. Furthermore, a new generation of NASA high energy astronomy missions is providing an unprecedented quantity and quality of observational data from such objects that presents a tremendous modeling challenge for astrophysical theorists. Compact objects posses strong gravitational fields and a correct quantitative description of fluid flows in and around these objects must take into account Einstein's theory of relativity. Furthermore, the gravitational fields are created by the fluid itself and thus the Einstein gravitational field equations must be solved simultaneously and self--consistently with the fluid dynamics equations.

As one of the NASA ESS HPCC Grand Challenge science teams we have endeavored to meet these challenges by developing several codes that solve both the relativistic Euler equations of fluid dynamics and the Einstein field equations describing gravity. This complex set of equations poses unique design problems. The relativistic analogs of the Euler equations consist of five hyperbolic equations describing energy, momentum, and baryon conservation which are similar in structure to the non-relativistic Euler equations. Thus we can apply a wide variety of traditional explicit fluid dynamics techniques to solve this set of equations. The Einstein field equations are far more complex and present a more difficult challenge to numerical solution. The Einstein equations are normally written as a set of sixteen second order differential algebraic equations (DAEs) that are not amenable to numerical solution. However, with an appropriate reformulation the equations can be cast into a the form of a set of 37 coupled first order hyperbolic PDE's. By employing this latter form we can make use of much of our intuition about CFD to implement a numerical scheme for the solution of these equations.

Because of the complexity of the aforementioned equations the numerical time--evolution during a simulation requires a large number of floating point operations per zone per timestep. Additionally, the simulation requires nearly 100 3-D arrays providing a large memory footprint for the code. These requirement together with the need to evolve for many timesteps, for most problems of interest, demands the use of cutting edge parallel computing platforms. Accordingly, we have chosen to implement the codes within a message passing context. This choice has allowed us to obtain excellent scalability and the widest possible range of portability. I will discuss our design strategies and choices for the implementation of these codes. Finally, I will discuss the testbed problem that we are using to drive the development of these codes: modeling the merger of two neutron stars in a binary system. Ultimately, our astrophysical goal in this problem is to provide a systematic comparison of Newtonian, post-Newtonian, and general relativistic models for this phenomenon. I will briefly present some results for this testbed problem.

The graphic displays a set of isodensity contours (in the orbital plane) from a Newtonian merger



of two neutron stars. The time of this snapshot is approximately 1 millisecond into the merger, when the neutron stars have started to coalesce. The direction of motion of the stars is counter-clockwise. The data is from a simulation by Doug Swesty, Ed Wang, and Alan Calder of NCSA.

PARALLELIZATION OF THE PHYSICAL-SPACE STATISTICAL ANALYSIS SYSTEM (PSAS)

J.W. Larson, J. Guo, and P.M. Lyster

Atmospheric data assimilation is a method of combining observations with model forecasts to produce a more accurate description of the atmosphere than the observations or forecast alone can provide. Data assimilation plays an increasingly important role in the study of climate and atmospheric chemistry. The NASA Data Assimilation Office (DAO) has developed the Goddard Earth Observing System Data Assimilation System (GEOS DAS) to create assimilated datasets. The core computational components of the GEOS DAS include the GEOS General Circulation Model (GCM) and the Physical-space Statistical Analysis System (PSAS). The need for timely validation of scientific enhancements to the data assimilation system poses computational demands that are best met by distributed parallel software.

PSAS is implemented in Fortran 90 using object-based design principles. The analysis portions of the code solve two equations. The first of these is the "innovation" equation, which is solved on the unstructured observation grid using a preconditioned conjugate gradient (CG) method. The "analysis" equation is a transformation from the observation grid back to a structured grid, and is solved by a direct matrix-vector multiplication. Use of a factored-operator formulation reduces the computational complexity of both the CG solver and the matrix-vector multiplication, rendering the matrix-vector multiplications as a successive product of operators on a vector. Sparsity is introduced to these operators by partitioning the observations using an icosahedral decomposition scheme. PSAS builds a large (~128MB) run-time database of parameters used in the calculation of these operators.

Implementing a message passing parallel computing paradigm into an existing yet developing computational system as complex as PSAS is nontrivial. One of the technical challenges is balancing the requirements for computational reproducibility with the need for high performance. The problem of computational reproducibility is well known in the parallel computing community. It is a requirement that the parallel code perform calculations in a fashion that will yield identical results on different configurations of processing elements on the same platform. In some cases this problem can be solved by sacrificing performance. Meeting this requirement and still achieving high performance is very difficult. Topics to be discussed include: current PSAS design and parallelization strategy; reproducibility issues; load balance vs. database memory demands; possible solutions to these problems.

PARALLELIZING OVERFLOW: EXPERIENCES, LESSONS, RESULTS

Dennis C. Jespersen
MS T27A-1
NASA/Ames Research Center
Moffett Field, CA 94035-1000
(650) 604-6742
{jesperse@nas.nasa.gov}

1. Introduction

The computer code OVERFLOW is widely used in the aerodynamic community for the numerical solution of the Navier-Stokes equations. Current trends in computer systems and architectures are toward multiple processors and parallelism, including distributed memory. This report describes work that has been carried out by the author and others at Ames Research Center with the goal of parallelizing OVERFLOW using a variety of parallel architectures and parallelization strategies.

This paper begins with a brief description of the OVERFLOW code. This description includes the basic numerical algorithm and some software engineering considerations. Next comes a description of a parallel version of OVERFLOW, OVERFLOW/PVM, using PVM (Parallel Virtual Machine). This parallel version of OVERFLOW uses the manager/worker style and is part of the standard OVERFLOW distribution. Then comes a description of a parallel version of OVERFLOW, OVERFLOW/MPI, using MPI (Message Passing Interface). This parallel version of OVERFLOW uses the SPMD (Single Program Multiple Data) style. Finally comes a discussion of alternatives to explicit message-passing in the context of parallelizing OVERFLOW.

2. Basics of the code

OVERFLOW is a computer code for the numerical solution of the Navier-Stokes equations, oriented toward applications in aerodynamics [1], [2]. [3]. OVERFLOW uses finite differences in space on structured meshes, implicit time-stepping, and general overlapping grids for dealing with complex geometries [4]. A variety of different time-stepping methods and spatial differencing schemes are available. The OVERFLOW code is authored by P. Buning. OVERFLOW has a demonstrated capability to solve flow problems with a large number of unknowns in complex geometries [5], [6].

OVERFLOW handles general geometries by allowing overset or "chimera" meshes. In this approach individual meshes are generated about individual geometrical components such as wings, nacelles, or pylons, and allowed to overlap or cut holes in one another in an arbitrary fashion. Boundary data needed at outer or hole boundaries of a given component mesh is obtained by interpolation from another mesh. This overset mesh capability is essential for OVERFLOW in practical situations.

OVERFLOW allows multitasking on multiprocessor machines such as the Cray-C90. The multitasking is via explicit directives; different computational planes L = constant or K = constant are given to different processors. This multitasking can be thought of as a medium-grain multitasking, as it is above the simple do-loop level and below the zonal level.

Considerations of software engineering play an important role in working with OVERFLOW. Maintainability, readability, portability, and ease of revision are all important criteria. OVERFLOW is a fairly large code. A recent snapshot of the code counted approximately 88000 lines of Fortran code (Fortran77) and 1500 lines of C code. Of the lines of Fortran code, about one-third are comment lines, and of these comment lines about one-half are empty comment lines, used to enhance readability. The snapshot revealed 963 subroutines spread over 952 source files.

OVERFLOW is a user-driven code, that is, most of the features in OVERFLOW are there because of user request. This implies that parallelization efforts, to be widely useful, have to address most or all of the components of the code (turbulence models, boundary conditions, implicit solvers, etc.). Parallelization of a small subset of the code would be of limited use.

Finally, OVERFLOW is a dynamic package, it is not a fixed target. It is continually evolving, so additions and modifications to the package must be carefully integrated into the package so that they are intelligible to others who make modifications, and they should be able to be carried forward into the future.

3. Explicit message-passing with no zonal splitting

A parallel version of OVERFLOW called OVERFLOW/PVM exists. It is integrated in the main OVERFLOW package and is automatically distributed as part of OVERFLOW. The code is written in the manager/worker style exploiting parallelism at the zonal level. The code is fault-tolerant, in the sense that the code can recover from the failure of a worker. There is no splitting of zones across processors, and each worker handles a single zone. The various output files produced by the serial version of OVERFLOW (residuals, forces and moments, minimum density and pressure, turbulence residuals) are also produced by the OVERFLOW/PVM code.

The goal of integrating the parallel code into the main OVERFLOW package was reached by modifying selected OVERFLOW subroutines with conditional compilation blocks ("`#ifdef PVM`") that would be activated only if compiling for the PVM target. A total of just 15 OVERFLOW subroutines were modified in this way. The rest of the new code was separated into 60 subroutines with about 6500 total lines of code. The OVERFLOW subroutines that needed modification fell into three classes: startup, shutdown, and output. The low number of modified OVERFLOW subroutines and the absence of modification of the lower-level internal numerical subroutines greatly eases the burden of maintaining the OVERFLOW/PVM code as the OVERFLOW package is continually modified and upgraded.

The most serious problem with OVERFLOW/PVM stems from the prohibition on splitting zones across processors. If a zone has too many grid points for the memory of any of the processors, then either the code performance will suffer greatly due to swapping on the node that has too many grid points, or else the code will completely fail to run (if the operating system on the node lacks virtual memory). There are many practical cases in which one or more zones has more grid points than can be accommodated on a single processor. In order for the OVERFLOW/PVM package to be used in such cases, manual regridding (a tedious and time-consuming procedure) is usually necessary.

Even if there is sufficient memory on all the nodes there is a fundamental problem having to do with zone imbalance and maximum possible speedup. Suppose, for example, that a given problem has one zone with half of the grid points, while the other half of the grid points are spread among several zones. Then no matter how many processors are used, the speedup for OVERFLOW/PVM cannot be more than 2. This is simply because the largest zone will always be a bottleneck if zonal splitting is not allowed. This indicates that the OVERFLOW/PVM approach is best suited for cases where the grid is such that (or can be generated such that) there is no one zone with a large plurality of the grid points.

4. Explicit message-passing with zonal splitting

The problem with the largest zone being a bottleneck for OVERFLOW/PVM motivated the work on OVERFLOW/MPI. The goals of this effort included the following: ability to partition a zone across multiple processors, to alleviate the largest zone bottleneck; ability to cluster several zones onto a

single processor, for good load balancing; ability to produce the various output files emitted by the serial version of OVERFLOW (residuals, forces and moments, minimum density and pressure, turbulence residuals).

This work was begun in conjunction with Ferhat Hatay (formerly Research Scientist with MCAT, Inc., now System Engineer with HAL Computer Systems, Inc.) We decided to adopt the SPMD (single program, multiple data) paradigm for this project. We also decided to adopt \MPI, hoping for better bandwidth and latency on tightly-coupled multiprocessors.

It should be emphasized at the outset that this parallel version of OVERFLOW could have been coded with PVM. The term "OVERFLOW/MPI" is simply a convenient way to say "parallel implementation of OVERFLOW using the SPMD style and allowing zones to be split across processors".

It turns out that we needed to modify 58 subroutines of OVERFLOW (compared to the 15 that needed modification in the OVERFLOW/PVM project). The rest of the new code was separated into 144 subroutines with almost 20000 lines of code. Allowing for partitioning of a given zone across multiple processors accounts for the bulk of the increased complexity.

Why is it that allowing partitioning increases the complexity so much? The reasons are several, and some of them are nonobvious. Allowing an overlap of "halo" points and sending data from one processor to update a neighbor's halo points is an well-understood idea. OVERFLOW has the additional complexity of using implicit time stepping (explicit time stepping is not an option!), so code which solves linear systems (scalar pentadiagonal, scalar tridiagonal, block tridiagonal) spread across processors must be developed and tested. There are several algorithms for solving sparse banded linear systems spread across processors. We chose to implement pipelined Gaussian elimination, both one-way and two-way, as our solvers. Periodic solvers would add an additional level of complexity (these are not yet implemented into OVERFLOW/MPI).

One nonobvious reason for the increased complexity has to do with boundary conditions. A very common boundary condition in aeronautics applications is the wake cut boundary condition in a C-mesh. In this setup, a single physical point corresponds to two distinct computational points and the flow variable values at these two computational points are determined by averaging the values from the adjacent points in physical space above and below the cut. In essence, the boundary condition is nonlocal in computational space. If the grid is partitioned so that the grid points in the lower part of the wake go to a different processor from the grid points in the upper part of the wake, then the wake cut boundary condition will require interprocessor communication. In fact, any boundary condition that is nonlocal in computational space may require interprocessor communication. There are other boundary conditions in OVERFLOW that are nonlocal in computational space.

The question of chimera boundary conditions arises for OVERFLOW/MPI. What if a zone is partitioned across multiple processors and each processor needs to read or write chimera boundary data? Our solution is to designate one of the processors as the ``local master", and to have all chimera boundary data communication occur via local masters.

The code has been tested on SGI workstations (with the MPICH implementation), on the IBM SP2, Cray-T3E, Cray-J90, and on several SGI multiprocessor platforms (Power Challenge, Onyx2, Origin2000).

Now we consider performance for OVERFLOW/MPI. First we study speedup. The test case is a 69x61x50 mesh (single zone, 210450 points). The case was run on 1,2,4, and 8 nodes on 3 parallel machines: IBM SP2, SGI Origin2000, and Cray T3E. With 2 nodes the mesh was partitioned 2x1x1, with 4 nodes the mesh was partitioned 2x1x2, and with 8 nodes the mesh was partitioned 4x1x2 (the middle dimension has a periodic boundary condition and partitioning in a

periodic direction is currently not allowed). The data are given in Table 1. They show respectable speedups for the code, especially in light of the fact that 210450 grid points are not too many to begin with, and that spreading them over 8 processors gives each processor about 26300 grid points, quite a small number.

Machine	No. of nodes	Sec/step	Speedup
SP2	1	14.74	
	2	8.55	1.72
	4	4.75	3.11
	8	2.81	5.24
Origin 2000	1	7.99	
	2	4.73	1.69
	4	2.91	2.74
	8	1.83	4.38
T3E	1	20.15	
	2	10.90	1.85
	4	5.43	3.71
	8	3.42	5.90

Table 1: Speedup for OVERFLOW/MPI

In the second case for OVERFLOW/MPI performance, we show scaled speedup. Here we start out with the same original 69x61x50 mesh, and for the same geometry generate new meshes with 2,4, and 8 times the number of mesh points. Thus the four meshes have approximately 210K, 420K, 840K, and 1680K points. These meshes are run with 1,2,4, and 8 processors, respectively, so the number of grid points per processor remains constant. In Table 2 efficiency is defined as $T(1)/T(N)$, i.e., time for 1 processor divided by time for N processors.

Machine	No. of nodes	Sec/step	Efficiency
SP2	1	14.74	
	2	17.64	0.84
	4	17.80	0.83
	8	20.32	0.73
Origin 2000	1	7.99	
	2	9.38	0.85
	4	14.01	0.57
	8	26.74	0.48
T3E	1	20.15	
	2	22.33	0.90
	4	24.81	0.81
	8	28.72	0.70

Table 2: Scaled Speedup for OVERFLOW/MPI

Table 2 indicates that the SP2 scaled efficiency degrades gracefully as the number of nodes is increased. Also, the T3E efficiency degrades slowly (but the single-node performance of the T3E

is significantly slower than the single-node performance of the SP2 or the Origin2000). Finally, the Origin2000 efficiency suffers a sharp drop in going from 2 to 4 processors, possibly because of the architecture of that machine which features 2 processors attached to a shared memory as a single "node".

5. Beyond explicit message-passing

In this section I will try to take a look at explicit message-passing as a general parallelization tool in the context of OVERFLOW, consider its strengths and weaknesses, and consider possible alternatives.

It is this author's opinion that manually writing explicit message-passing code is a low-level, tedious, error-prone task which is not an appropriate or cost-effective use of a human programmer's time. If explicit message-passing is to be used it should be produced automatically, either as part of some higher-level programming language or else via some automatic tool with user input. At least one such tool is available (CAPTools[8]).

Now consider current trends in high-performance computer systems. The trend seems to be away from pure distributed memory systems and toward some form of shared memory, perhaps distributed shared memory where the memory is physically distributed across processors but logically shared. There are even software systems that can run on a set of workstations and make the separate workstations appear to have a shared memory ([9], [10]).

It appears that the partitioning and clustering of OVERFLOW/MPI is necessary for good performance and efficiency, but that it creates many headaches in terms of code reliability and maintenance. So it may be worthwhile to look for another way to exploit parallelism.

James Taft has proposed such a method ([11], also this conference). In this method, which he calls multilevel parallelism, there is no explicit message-passing, yet parallelism can be exploited at the zonal level and at the intrazonal level. The zones are clustered into groups (a group consists of one or more zones) and a "master" process is forked for each group. Each forked process also has one or more threads associated with it. The masters proceed in parallel, and each master utilizes its given number of threads along with the medium-grain multitasking directives already in OVERFLOW. Thus there is parallelism at the level of the masters and parallelism beneath each master. Some sort of distributed shared memory is assumed for the underlying architecture.

This idea has some attractive features. It avoids all explicit message-passing and it uses the existing multitasking features of OVERFLOW. There is less low-level code modification necessary, even less than in OVERFLOW/PVM; only the low-level portions of the code that deal with writing files of residuals, etc., need to be modified. This idea should be portable to a variety of machines, especially since the medium-grain multitasking is compatible with OpenMP [12] and since some sort of distributed shared memory seems to be more and more common.

In summary, the OVERFLOW code has been parallelized with explicit message-passing in two distinct fashions: manager/worker style with no splitting of zones across nodes (OVERFLOW/PVM), and in SPMD style with zonal splitting and coalescing (OVERFLOW/MPI). The former version is part of the standard OVERFLOW distribution, and the latter version is ready to be part of the standard OVERFLOW distribution.

OVERFLOW/MPI, because it allows zonal splitting, is more efficient than OVERFLOW/PVM, but this efficiency comes at a high price in terms of software development cost and code maintainability. This seems to be inherent in explicit message-passing.

Newer approaches using distributed shared memory and multitasking directives have great promise and should be vigorously pursued.

A longer version of this paper may be found by following the links from the author's web page (`http://science.nas.nasa.gov/~jesperse`).

References

[1] P.G. Buning, I.T. Chiu, S. Obayashi, Y.M. Rizk, and J.L. Steger, "Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent", AIAA-88-4359-CP, AIAA Atmospheric Flight Mechanics Conference, August 1988, Minneapolis, MN.

[2] K.J. Renze, P.G. Buning, and R.G. Rajagopalan, "A Comparative Study of Turbulence Models for Overset Grids", AIAA-92-0437, AIAA 30th Aerospace Sciences Meeting, Reno, NV, Jan. 6--9, 1992.

[3] M. Kandula and P.G. Buning, "Implementation of LU-SGS Algorithm and Roe Upwinding Scheme in OVERFLOW Thin-Layer Navier-Stokes Code", AIAA-94-2357, AIAA 25th Fluid Dynamics Conference, Colorado Springs, CO, June 1994.

[4] J.A. Benek, P.G. Buning, and J.L. Steger, "A 3-D CHIMERA Grid Embedding Technique", AIAA-85-1523-CP, July 1985.

[5] J.P. Slotnick, M. Kandula, and P.G. Buning, "Navier-Stokes Simulation of the Space Shuttle Launch Vehicle Flight Transonic Flowfield Using a Large Scale Chimera Grid System", AIAA-94-1860, AIAA 12th Applied Aerodynamics Conference, Colorado Springs, CO, June 1994.

[6] L.M. Gea, N.D. Halsey, G.A. Intemann, and P.G. Buning, "Applications of the 3D Navier-Stokes Code OVERFLOW for Analyzing Propulsion-Airframe Integration Related Issues on Subsonic Transports", ICAS-94-3.7.4, Proceedings of the 19th Congress of the International Council of the Aeronautical Sciences (ICAS 94), Anaheim, CA, Sept. 1994, pp. 2420--2435.

[7] M.J. Djomehri and K. Gee, personal communication.

[8] `http://www.gre.ac.uk/~captool/`

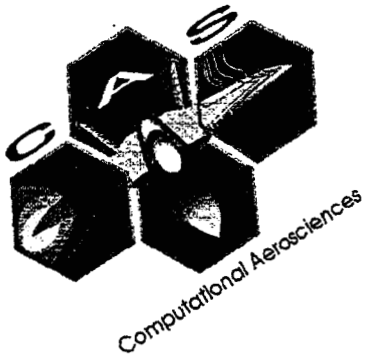
[9] `http://suif.stanford.edu/~scales/sam.html.`

[10] `http://www.cs.rice.edu/~willy/TreadMarks/overview.html.`

[11] "OVERFLOW Gets Excellent Results on SGI Origin2000", NAS Newsletter, `http://science.nas.nasa.gov/Pubs/NASnews/98/01/.`

[12] `http://www.openmp.org/.`

OMIT THIS PAGE



Session 5:

Applications for Parallel/Distributed Computers

PERFORMANCE AND APPLICATION OF PARALLEL OVERFLOW CODES ON DISTRIBUTED AND SHARED MEMORY PLATFORMS

M. Jahed Djomehri
Calspan Co., NASA Ames Research Center, M/S 258-1
Moffett Field, CA 94035
(650)604-6216, djomehri@nas.nasa.gov

Yehia M. Rizk
NASA Ames Research Center, M/S 258-1
Moffett Field, CA 94035
(650)604-4466, yrizk@mail.arc.nasa.gov

The presentation discusses recent studies on the performance of the two parallel versions of the aerodynamics CFD code, OVERFLOW_MPI and _MLP. Developed at NASA Ames, the serial version, OVERFLOW, is a multidimensional Navier-Stokes flow solver based on overset (Chimera) grid technology. The code has recently been parallelized in two ways. One is based on the explicit message-passing interface (MPI) across processors and uses the _MPI communication package. This approach is primarily suited for distributed memory systems and workstation clusters. The second, termed the multi-level parallel (MLP) method, is simple and uses shared memory for all communications. The _MLP code is suitable on distributed-shared memory systems. For both methods, the message passing takes place across the processors or processes at the advancement of each time step. This procedure is, in effect, the Chimera boundary conditions update, which is done in an explicit "Jacobi" style. In contrast, the update in the serial code is done in more of the "Gauss-Sidel" fashion. The programming efforts for the _MPI code is more complicated than for the _MLP code; the former requires modification of the outer and some inner shells of the serial code, whereas the latter focuses only on the outer shell of the code.

The _MPI version offers a great deal of flexibility in distributing grid zones across a specified number of processors in order to achieve load balancing. The approach is capable of partitioning zones across multiple processors or sending each zone and/or cluster of several zones into a single processor. The message passing across the processors consists of Chimera boundary and/or an overlap of " halo" boundary points for each partitioned zone. The MLP version is a new coarse-grain parallel concept at the zonal and intra-zonal levels. A grouping strategy is used to distribute zones into several groups forming sub-processes which will run in parallel. The total volume of grid points in each group are approximately balanced. A proper number of threads are initially allocated to each group, and in subsequent iterations during the run-time, the number of threads are adjusted to achieve load balancing across the processes. Each process exploits the multitasking directives already established in Overflow.

Performance of the _MPI and _MLP codes on the Origin 2000 128 CPU system is shown on the next page. The numerical test case used here consist of 41 overlap grid zones about a test article mounted in the NASA Ames pressure wind tunnel. The total number of grid points are 16 million. Table 1 and 2 show performance data, elapsed execution wall time, total floating point counts per second, and the average counts per CPU. Figures 1 through 3 display plots of scaled performances; speedup, floating-point, and execution-time per time step for each of the above parallel codes. A reasonable speedup is observed with each of the codes, with a maximum close to 6000 Mflops for the _MLP code with the 128 processors. The performance on 64 processors is close to the serial OVERFLOW code performance on the Cray C-90 using 16 processors.

S22-62
018251

MULTI-LEVEL PARALLELISM (MLP)

A SIMPLE HIGHLY SCALABLE APPROACH TO PARALLELISM FOR CFD

James R. Taft
Sierra Software, Inc.
NASA AMES Research Center
M/S 258-5
Moffett Field, CA 94035
jtaft@nas.nasa.gov
(650) 604-0704

366819
61

Abstract

High Performance Computing (HPC) platforms are continually evolving toward systems with larger and larger CPU counts. For the past several years these systems almost universally utilize standard off-the shelf microprocessors for the heart of their design. Virtually all hardware vendors have adopted this design approach as it dramatically reduces costs for building large systems.

Unfortunately, systems built from commodity parts usually force researchers to embark upon large code conversion efforts in order to take advantage of any potentially high levels of performance. Historically, this has been a daunting, and often unsuccessful task. For those who have attempted it, the effort often consumed many man-years of effort. Codes used in heavy production environments were often deemed to be impossible to convert before the effort was even begun.

Recent developments at the NASA AMES Research Center's NAS Division have demonstrated that the new generation of NUMA based Symmetric Multi-Processing systems (SMPs), such as the Silicon Graphics Origin 2000, can successfully execute legacy vector oriented CFD production codes at sustained rates far exceeding processing rates possible on dedicated 16 CPU Cray C90 systems.

This high level of performance is achieved via shared memory based Multi-Level Parallelism (MLP). This programming approach, outlined below, is distinct from the message passing paradigm of MPI. It offers parallelism at both the fine and coarse grained level, with communication latencies that are approximately 50-100 times lower than typical MPI implementations on the same platform. Such latency reductions offer the promise of performance scaling to very large CPU counts. The method draws on, but is also distinct from, the newly defined OpenMP specification, which uses compiler directives to support a limited subset of multi-level parallel operations.

The NAS MLP method is general, and applicable to a large class of NASA CFD codes. The MLP methodology and techniques are described below. The method is discussed in general terms, followed by discussion of the technique as applied to the OVERFLOW CFD code. Finally, performance results are presented for a 35 million point problem executed on Origin 2000 systems varying in size from 8 to 256 CPUs. Over 20 GFLOPS was ultimately obtained.

1.0 What is Multi-Level Parallelism (MLP)

Simple fine grained automatic parallel decomposition of application codes is not new. It has been utilized extensively for about two decades. It began with the introduction of the Cray Research XMP line of supercomputers. Much of the parallelism achieved on this machine was transparently provided by the compiler at the loop level, in which different iterations of the computational loops were executed in parallel on different CPUs in the system. In some instances, these parallel loops

contained subroutine calls. Often these routines did a significant amount of work, and the efficiency of the parallel executions were greatly enhanced when this occurred. This "coarsening" of the fine grained loop level parallelism supported by the compilers was the major focus of optimizing efforts for many years, and was the accepted way of achieving high levels of performance on the Parallel/Vector machines from Cray, and others.

In the 80's computer budgets began to shrink and many researchers turned to an alternative model of parallel computation based on simultaneously executing many communicating independent parallel processes. This true "coarse grained" approach was ideally suited to executions on networks of inexpensive workstations. Performance however, was often elusive.

Coarse grained parallelism seriously began to be accepted in the community with the introduction of the platform independent Parallel Virtual Machine (PVM) message passing library from the Oak Ridge National Laboratory [1]. This was the standard for many years. Today, the most popular method of implementing this level of parallelism is via the PVM successor, the Message Passing Interface (MPI) library, from the Argonne National Laboratory [2].

Historically, codes decomposed with these message passing libraries were most often destined for execution on networks of single CPU workstations, or their topological equivalents, such as the Intel Paragon, Thinking Machines CM5, or IBM SP2. Applications developers spent substantial amounts of time attempting to decompose the problems so that communication between each CPU on the interconnect was at an absolute minimum. No thought was given to multiple levels of parallelism as the architectures simply did not support it.

With the advent of inexpensive moderately parallel RISC based SMPs from HP, DEC, Sun, and SGI, expanding to a second level of parallelism was possible. Users could decompose the problem at the coarsest level with MPI across SMPs, and use the compiler to provide fine grain parallelism at the loop level within an SMP via directives such as those within OpenMP [3]. In general however, clusters of SMPs were still treated as a series of discrete single processor entities, and MPI messages were still exchanged between CPUs even within a single SMP.

While the MPI/OpenMP hybrid approach is potentially better at scaling than the pure MPI solution, the approach has the major drawback that it is still subject to the relatively high MPI latencies whenever messages are used. More importantly, it requires a major rewrite of the code to fully decompose the problem for coarse grained parallel execution.

2.0 What is Shared Memory MLP

Very recently, manufacturers have adopted a new architectural design philosophy resulting in a hierarchical SMP that supports very large CPU counts (>100), albeit with non-uniform memory access (NUMA). The Origin 2000 system from SGI is one such system. For many applications 100 CPUs is more than enough computational power to solve the problem in a reasonable timeframe, and the need to traverse multiple SMPs to achieve the desired level of sustained performance is not necessary. This opens the door to some interesting possibilities. In particular, the high latency HiPPI connections between SMPs can be removed from the problem as there is only one SMP involved. MPI can also be dropped as there is no need to spawn processes on other SMPs, and there are much simpler ways of spawning them on a single SMP.

Given a true SMP architecture and a problem that fits within it, one can define a new way of performing multi-level parallel executions. To distinguish it from past approaches, we define it as Shared Memory MLP. It differs from the MPI/OpenMP approach in a fundamental way in that it does not use messaging at all. All data communication at the coarsest and finest levels is accomplished via direct memory referencing instructions, which are fully supported by the SMP instruction set and underlying hardware. Furthermore, shared memory MLP is different from just

OpenMP (when used in its limited multi-level mode) in that it makes extensive use of independent UNIX processes and shared memory arenas to accomplish its goals. These features are not supported by OpenMP. Both of these features allow shared memory MLP to provide superior performance to the alternatives. More importantly, they provide a simpler mechanism for converting legacy code than either OpenMP or MPI.

For shared memory MLP, the coarsest level of parallelism is not supplied by spawning MPI processes, but rather by the spawning of independent processes via the standard UNIX fork, a system call available on all UNIX systems. This is a much simpler method in that the user simply makes fork calls at any time in the execution of his program to create another process. The user may spawn as many such processes as desired, and each of the processes can execute on one or more CPUs via compiler generated parallelism. The advantage of the fork over the MPI procedure is that the forks can be inserted well after all of the initialization phase of a typical CFD code. Thus, the user does not need to dramatically alter and decompose the initialization sections of major production codes, a daunting task at best.

Once the forks take place, all communication of data between the forked processes is accomplished by allocating all globally shared data to a UNIX shared memory arena, another system call available on all UNIX RISC systems. Again this is a simple process and results in a dramatic reduction in communication latencies over MPI. By using the arena approach, all global communication takes place via memory load and store operations requiring just hundreds of nanoseconds, not the tens of microseconds typical of MPI messaging latencies.This dramatic 50-100 fold reduction in data access times provides the support needed for greatly enhanced parallel scaling needed in typical applications.

3.0 OVERFLOW-MLP

The shared memory MLP recipe described above is very apropos for the field of CFD. In particular, it is ideally suited for CFD computations that utilize multi-zonal approaches in which the total computational domain is broken into many smaller sub-domains. Several production CFD codes at NASA utilize this solution approach. OVERFLOW is one of them.

OVERFLOW was chosen as the test bed to examine the performance, ease of use, and robustness of the MLP technique. It is one of the largest consumers of machine resources at NASA sites. OVERFLOW is a 3D RANS code solving steady and unsteady flow problems of interest.The code consists of approximately 100,000 lines of FORTRAN. It is heavily vectorized, and has historically executed well on the C90 systems at NAS. Typical sustained performance levels are around 450 MFLOPS per processor, with sustained parallel processing rates of around 4.5 GFLOPS on dedicated 16 CPU C90 systems. As such it is considered a good vector/parallel code.

Shared memory MLP was inserted into OVERFLOW by constructing a very small library of routines to initiate forks, establish shared memory arenas, and provide synchronization primitives. Calls to these routines were inserted as needed into the C90 version of the code. This library will be available to users within the next few weeks.The initial effort to convert OVERFLOW to MLP required only a few man weeks and a few hundred lines of code changes. The effort involved slightly modifying the main program, and six other routines out of the nearly 1000 routines in the code.

Figure 1 shows the minor restructuring of the OVERFLOW logic that was needed to insert the MLP strategy. As can be seen in the figure, the main calculational sequence is a series of loops over time and grids. In addition, an outer loop may be executed if the multigrid integration option is selected. Also, a sub-cycling iteration may be enabled for any particular grid. The major change for MLP is to sub-divide the serial grid loop into two loops, a loop over groups, and a loop over the grids within each group. The outer loop is done in parallel, with one group assigned to each

MLP process. Each process performs the inner loop over just the grids assigned to it. All initialization and wrapup tasks remain unchanged from the C90 code, as do all of the solvers, etc.

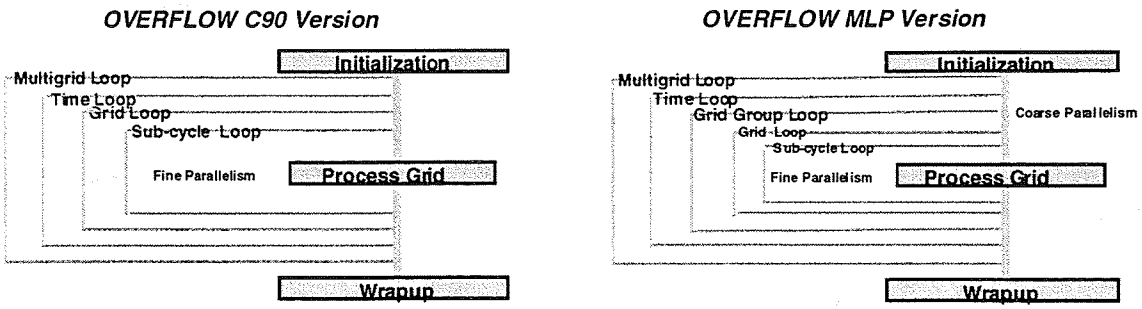


Figure 1 - Logic flow for C90 and MLP Versions of OVERFLOW

The MLP processes performing the work only need to communicate boundary data at a few key points in time during the course of the calculation. The remainder of the time is spent doing computations totally independent of each other.

Figure 2 depicts the MLP layout of the data and communication occurring within the Origin 2000 architecture. Each MLP process is assigned a given number of CPUs. The CPU count for each process is determined from a load balance analysis at run time that attempts to keep the number of points solved by each process about the same. Each process solves only those grids assigned to it. The grids for each process are allocated to memories close to the CPUs executing the MLP process assigned to the grids. The boundary data is archived in the shared memory arena by each process as it completes its processing of a grid. Other processes read this data directly from the arena as needed. At the end of a time step all processes are synchronized at a barrier, and the procedure repeats for each time step taken.

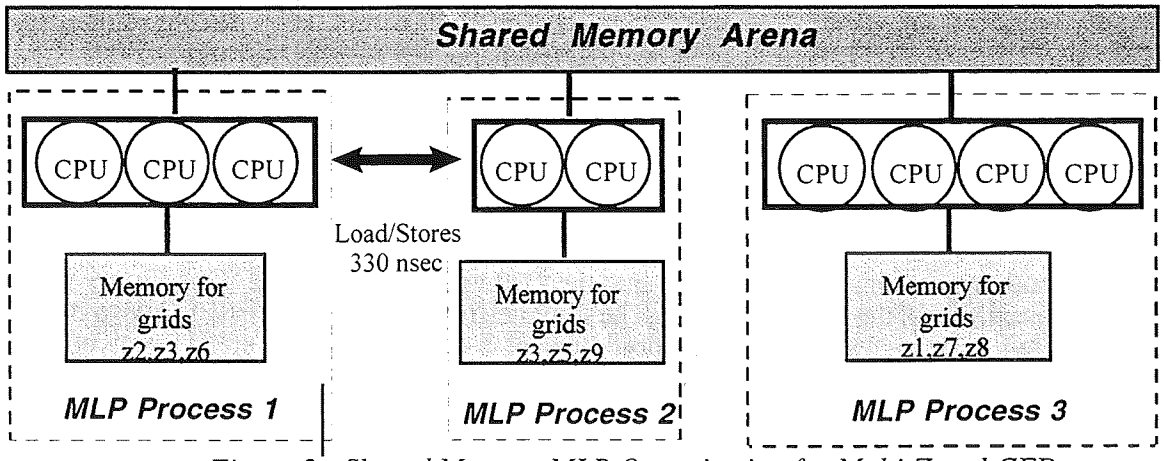


Figure 2 - Shared Memory MLP Organization for Multi-Zonal CFD

Doing the computation of zones in parallel is not new. In fact the MPI version of OVERFLOW already does this. The unique feature of the shared memory MLP approach is that it does so with no message passing and only a few hundred lines of code changes. The MPI implementation requires approximately 10,000 additional lines of code. The end result is that the MLP code is

simpler to maintain, continues to execute well on C90 systems, and now executes well on parallel systems at very high sustained levels of performance as seen below.

4.0 OVERFLOW-MLP Performance Results

The major focus during the development of the MLP technique was on obtaining efficient parallel scaling. It is a fact that all of the best RISC based microprocessors rarely achieve in excess of 100 MLFOPS per processor on typical production CFD codes. Memory access is almost always the inhibitor to higher levels of single CPU performance. Thus, unless a large CFD problem can scale to more than a hundred processors, sustained computation in excess of 10 GFLOPs is not likely. At least 10 GFLOPS is needed on the important large problems of today in order to solve them in an acceptable time frame.

It was clear that the MLP technique offered the promise of a tremendous reduction in communication latencies over an MPI implementation. In order to stress test the technique to the fullest, a large real production problem was selected that fully exercised OVERFLOW's typical options for solvers, smoothers, and turbulence models. The problem selected consisted of 35 million points divided among 160 3D zones. The zones varied in size from ~1.5 million points to ~15 thousand points. A total of 10 time steps were executed on various numbers of CPUs. Figure 3 shows the results of this test.

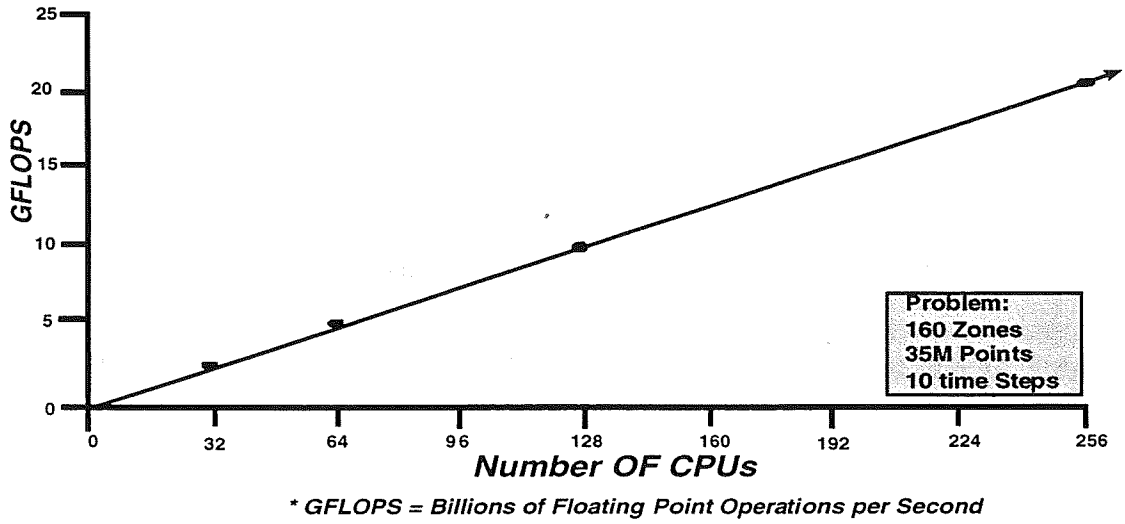


Figure 3 - OVERFLOW-MLP Performance versus CPU Count for Origin 2000

As can be seen the performance scales almost perfectly linearly with increasing processor count. Performance on 64 CPUs is about 5 GFLOPS. Performance on 128 CPUs is about 10 GFLOPS, and performance on 256 CPUs was 20.1 GFLOPS. Performance per CPU remained steady at about 80 MFLOPS.

The code under test is the standard C90 version of OVERFLOW 1.8. It contains no single CPU optimizations whatsoever. This was intentional as major modifications to the hundreds of routines to optimize them for RISC systems would make it extremely difficult to maintain synchronization with the OVERFLOW releases from NASA Langley. As it is, the MLP functionality can be added in less than one day once a new C90 release is received.

The fact that OVERFLOW-MLP is a pure vector code and yet executes at sustained performance levels in excess of 20 GFLOPS on RISC systems is remarkable. Essentially this indicates that the

new RISC systems will be able to significantly extend the performance envelop for large vector oriented production CFD codes for the first time. At the same time the codes can maintain compatibility (and performance) with the existing vector architectures. This is a very important feature as we enter this transition period from vector to RISC over the next few years.

5.0 Summary

The development of the MLP technique, implementation in the OVERFLOW code, and achievement of 20 GFLOPs of sustained performance required about a one man-year level of effort. The vast majority of that time was spent in learning the OVERFLOW code itself.

The test case presented above clearly demonstrates that the MLP technique is robust. It has been used by several groups at this point, and has been shown to perform well on problems ranging from a few million points to over 35 million points. Tentative work indicates 40+ GLFOPS are achievable in the near term on the 35 million point problem and 512 Origin processors.

The new techniques of Multi-Level Parallelism developed under the O2000 Optimization Effort have demonstrated dramatic cost and performance benefits for production CFD codes at NASA AMES. The popular CFD code, OVERFLOW, has sustained 20 GFLOPs in performance when solving the largest CFD problem ever attempted at NAS. If success continues, the newly developed MLP techniques will allow fast code conversions, a dramatic reduction in run times for the largest CFD problems, and allow this on platforms that are an order of magnitude lower in cost than typical traditional vector supercomputer resources.

References

1. A. Geist, etal. PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Cambridge, MA, 1994.

2. W. Gropp, etal. Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press, Cambridge, MA, 1994.

3. OpenMP Architecture Review Board. OpenMP. A proposed Standard API for Shared Memory Programming. October 1997.

MASSIVELY PARALLEL COMPUTATIONAL FLUID DYNAMICS CALCULATIONS FOR AERODYNAMICS AND AEROTHERMODYNAMICS APPLICATIONS

Jeffrey L. Payne and Basil Hassan
Sandia National Laboratories
Aerosciences and Compressible Fluid Mechanics Department
Mail Stop 0825
P. O. Box 5800
Albuquerque, NM 87185-0825

E-mail: jlpayne@sandia.gov; Phone: (505) 844-4524
E-mail: bhassan@sandia.gov; Phone: (505) 844-4682

Abstract

Massively parallel computers have enabled the solution of complicated flow fields (turbulent, chemically reacting) that were previously intractable. Calculations are presented using a massively parallel CFD code called SACCARA (Sandia Advanced Code for Compressible Aerothermodynamics Research and Analysis) currently under development at Sandia National Laboratories as part of the Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI). Computations were made on a generic reentry vehicle in a hypersonic flowfield utilizing three different distributed parallel computers to assess the parallel efficiency of the code with increasing numbers of processors. The parallel efficiencies for the SACCARA code will be presented for cases using 1, 10, 50, 100 and 500 processors. Computations were also made on a subsonic/transonic vehicle using both 236 and 521 processors on a grid containing approximately 14.7 million grid points. Ongoing and future plans to implement a parallel overset grid capability and couple SACCARA with other mechanics codes in a massively parallel environment are discussed.

Introduction

The massively distributed parallel CFD code currently under development at Sandia National Laboratories as part of the Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI) Program is called SACCARA (Sandia Advanced Code for Compressible Aerothermodynamics Research and Analysis). SACCARA is currently being developed from PINCA [1,2], a distributed parallel version of the commercial, finite volume, Navier-Stokes code, INCA [3], from Amtec, Inc. SACCARA solves the 2-D/Axisymmetric and 3-D Full Navier-Stokes equations for laminar and turbulent flows in thermo-chemical nonequilibrium. SACCARA is applicable from subsonic through hypersonic flows and can allow for perfect gas, equilibrium, and finite-rate chemistry. Standard zero-, one-, and two-equation turbulence models are also available. The code employs multiblock structured grids with point-to-point matchup at the block interfaces. The solution is driven to a steady state using the lower-upper symmetric Gauss Seidel (LU-SGS) or diagonal implicit solution advancement scheme based on a

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

combination of the work of Yoon et al. [4,5] and Peery and Imlay [6]. The inviscid fluxes are evaluated using the flux vector splitting of Steger and Warming [7] or the symmetric TVD flux function of Yee [8]. The viscous terms employ a standard central differencing scheme.

Parallel Implementation of SACCARA

SACCARA can be run on a variety of shared memory or distributed memory parallel platforms as well as in a serial mode. The code can be compiled using the standard message-passing interface (MPI) [9] as well as Intel's native NX [10] parallel calls. Currently SACCARA can only allow for a single grid block per processor. Communication between the grids is handled through a layer of ghost cells at each block interface where pertinent information is updated explicitly after each global iteration step. All the input information, including the block interface information, is contained in a single input data file using a namelist format. The multiblock, structured grid is input in standard PLOT3D format [11] or in INCA format [3]. Both the input data file (IDATA) and input mesh file (IMESH) are the same for either serial or parallel operation.

The grids used in SACCARA are typically made using GRIDGEN [12] or any other multiblock structured grid generator. GRIDGEN has the ability to generate not only the IMESH file in PLOT3D format but also much of the IDATA file, including the block interface information. The blocking structure of the grid is generally created based on the topology of the geometry of interest, without regard to the number of processors that will be used to solve the problem. The DECOMP [1] code is used to subdivide the grid into the number of blocks that equals the total number of processors desired for parallel operation. DECOMP, which uses many of the SACCARA subroutines, was recently modified to include part of the BREAKUP [13] code. The inputs to DECOMP are the IDATA and IMESH files and the total number of desired processors. Given the original blocking topology, DECOMP will determine the optimal load-balanced system. DECOMP first determines the total number of grid points and divides it by the number of total processors to obtain an average number of grid points per processor. DECOMP then begins subdividing those blocks whose total number of grid points is greater than the average along the I, J, and K coordinate directions. Blocks with fewer than the average number of grid points are not subdivided. DECOMP then chooses the divisions in the coordinate direction which best minimizes the surface area to volume ratio of each new grid block to ultimately reduce communication time during the solution. In addition, DECOMP may slightly reduce or increase the total number of processors if it determines it can create a more load-balanced decomposition with a different number of total processors. Finally, DECOMP creates the new IDATA and IMESH files to be used as input to SACCARA for parallel execution.

Results

SACCARA computations have been made on a hypersonic flowfield surrounding a generic reentry vehicle utilizing three different computer platforms. The calculations were made on three machines: the ASCI Red (Intel) at Sandia National Laboratories, the ASCI Blue ID (IBM SP2), and ASCI Blue TR at Lawrence Livermore National Laboratories, using 1, 10, 50, 100, and 500 processors to assess the parallel efficiency of the code with increasing numbers of processors. The size of the computational grid was varied to maintain 13,000 grid points on each processor.

The ASCI Red machine has 4500+ nodes. Each node has two 200-MHZ Pentium Pro processors with 128 Megabytes of memory/node. The ASCI Blue ID machine had 128 nodes. Each node has one IBM 66-MHz Power2 processor with 256 Megabytes of memory. The ASCI Blue TR machine, which recently replaced the ASCI Blue ID machine, has 158 nodes. Each node has four IBM 332-MHz 604e processors with 512 Megabytes of memory/node.

The parallel efficiency is defined as the ratio of the main loop serial CPU time to the main loop parallel CPU time with the number of grid points per processor remaining fixed. The efficiency of the code (Figure 1) is seen to drop off as expected due to increasing communication overhead as more processors are used. In Figure 1 a direct comparison of parallel efficiency can be made between the ASCI Blue ID and the ASCI Blue TR machine. There are a number of factors that increase the parallel overhead of the ASCI Blue TR machine. A major factor is the ASCI Blue TR machine requirement that distributed parallel codes use the slow IP method to communicate between nodes. The IP communication method is approximately three times slower than IBM's high-speed switch in dedicated mode. There is a planned update to the ASCI TR Machine to modify the current method for running a distributive parallel code on the TR machines to allow the use of the high-speed switch. This modification should provide a substantial increase in the parallel efficiency.

Computations were also made on the subsonic/transonic vehicle shown in Fig. 2 on 521 processors (ASCI Red) and 236 processors (ASCI Blue). The entire grid contained approximately 14.7 million grid points. The computational grid size and point distribution was based on previous axisymmetric solutions of a nonfinned vehicle at 0° angle of attack. The axisymmetric solution was run on a sequence of grids to determine the solutions senitivity to the grid resolution. The decomposed block structure on the surface and pressure contours are also shown in Fig. 2. The simulation of the subsonic/transonic vehicle in Figure 2 was performed at a Mach number of 0.8 and an angle of attack of 5°. The flowfield around the vehicle was assumed to be fully turbulent, and the one-equation, Baldwin Barth model was used to model the turbulent flowfield. The simulation was run on 236 processors and took 12.7 days of CPU time to reach a converged solution on the ASCI Blue TR machine. The simulation required 147,000 iterations to reach convergence and ran at aproximately 23 Gflops. The solution was considered converged when the global L2 norm of the momentum residuals had dropped by five orders of magnitude and the surface quanities no longer changed. The SACCARA code utilizies a point implicit method in its solution algorithm. This method is very efficient and highly stable but is inherently slow to converge. The resulting 147,000 iterations to reach convergence for this type of problem is not unexpected. The aerodynamic coefficients resulting from the simulation are shown in Table 1. The aerodynamic coefficients are presented in the form of percent difference compared to wind tunnel data and parameterized coefficients based on flight test data. The wind tunnel data is from a test performed in Sandia's Trisonic Wind Tunnel (TWT). The parameterized coefficients were determined by fitting the aerodynamic coefficients until the accelerometer and rate gyro data obtained from trajectory simulations matched those obtained in flight testing.

The parameterized coefficients can be viewed as a set of aerodynamic coefficients that, when used in a trajectory simulation code, accurately reproduce the vehicle's dynamic motion in flight. The accuracy of the single set of parameterized coefficients is not known. However it is

instructive to see how these coefficients compare with wind tunnel measurements and the coefficients computed with SACCARA. Table 1 shows good agreement between the pitching moment (C_m) and normal force coefficient (C_N) measured in the wind tunnel and the coefficients predicted by the CFD simulation. There is a larger discrepancy between the parameterized normal force coefficient and the normal force coefficient predicted in the CFD simulation. The almost zero percent difference with the center of pressure (X_{cp}/L) indicates that ratio of pitching moment-to-normal force is consistent between the different data sets. The rolling moment (C_l) coefficient comparison in Table 1 shows good agreement with the parameterized flight test data but a larger percent difference with the wind tunnel data. The largest disagreement is with the total axial force coefficent (C_A). It is believed that most of the difference in the prediction of axial force coeffcient is due to error in predicting the base pressure properly. The Balwin-Barth turbulent model is known not to accurately model the free shear layer in the base region. A 1 percent change in the average pressure acting on the base of the vehicle can result in a 20 percent change in the axial force coefficient. Additional work is planned to examine these results in more detail and run additional simulations using the Spalart-Allmaras one-equation model and the k-ω two-equation turbulence. Comparing these solutions should provide insight into the base pressure predictions and sensitivity to different turbulence models.

Table 1: Force and Moments

Percent Difference from Computation					
	C_A	C_N	C_m	C_l	X_{cp}/L
Flight Test	**56.0%**	**-9.54%**	**5.5%**	**-7.67%**	**.75%**
Wind Tunnel	**59.2%**	**2.4%**	**5.5%**	**-13.1%**	**.01%**

Ongoing and Future Work

SACCARA is continually being modified to handle new problems of interest. A multilevel overset, or Chimera, grid capability is being added to SACCARA to be used in a parallel environment. This capability will reduce the dependence on having point-to-point match up at the grid block interfaces, which can complicate the grid generation process on complex vehicle geometries. Overset grids will allow griding of individual geometric or flowfield features without regard for the overall grid topology. This will simplify the multiblock structured grid generation as well as reduce the total number of grid points necessary to solve a particular problem. In addition, overset grids will allow the solution of moving or multibody problems, such as store separation, without having to regrid as one body is moved relative to the other.

Work is also being performed to couple SACCARA with other mechanics codes in a distributed parallel environment. This work has included coupling SACCARA's compressible fluid mechanics capability to a parallel material thermal response code, COYOTE [14,15], for hypersonic reentry vehicle ablation prediction [0]. Ablation of an axisymmetric sphere-cone nosetip along a given trajectory using the coupled technique is shown in Figure 3. Current coupling between the codes has been achieved by exchanging surface interface information via file transfer. Ongoing work will allow these codes to be coupled using parallel calls. Finally,

plans are underway to couple both SACCARA and COYOTE to a six degree-or-freedom flight dynamics code for a full trajectory simulation of an ablating hypersonic vehicle.

References

1. Wong, C. C., Soetrisno, M., Blottner, F. G., Imlay, S. T., and Payne, J. L., "PINCA: A Scalable Parallel Program for Compressible Gas Dynamics with Nonequilibrium Chemistry," SAND 94-2436, Sandia National Laboratories, Albuquerque, NM, 1995.
2. Wong, C. C., Blottner, F. G., Payne, J. L., and Soetrisno, M., "Implementation of a Parallel Algorithm for Thermo-Chemical Nonequilibrium Flow Solutions," AIAA Paper No. 95-0152, Jan. 1995.
3. INCA User's Manual, Version 2.0, Amtec Engineering, Inc., Bellevue, WA, 1995.
4. Yoon, S., and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations," AIAA Paper No. 87-0600, Jan. 1988.
5. Yoon, S., and Kwak, D., "Artificial Dissipation Models for Hypersonic External Flow," AIAA Paper No. 88-3708, 1988.
6. Peery, K. M., and Imlay, S. T., "An Efficient Implicit Method for Solving Viscous Multi-Stream Nozzle/Afterbody Flow Fields," AIAA Paper No. 86-1380, June 1986.
7. Steger, J. L., and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Applications to Finite Difference Methods," *Journal of Computational Physics*, Vol. 40, 1981, pp. 263-293.
8. Yee, H. C., "Implicit and Symmetric Shock Capturing Schemes," NASA TM-89464, May 1987.
9. Message Passing Interface Forum. Document for a standard message-passing interface. Technical Report No. CS-93-214, University of Tennessee, Knoxville, TN, April 1994
10. McCurley K. S., "Intel NX compatibility under SUNMOS," SAND 93-2618, Sandia National Laboratories, Albuquerque, New Mexico, June 1995
11. Walatka, P. P., and Buning, P. G., "PLOT3D User's Manual," NASA TM 101067, 1989.
12. GRIDGEN User's Manual, Version 12, Pointwise, Inc., Bedford, TX, Nov. 1997.
13. Barnette, D. W., "A User's Guide for BREAKUP: A Computer Code for Parallelizing the Overset Grid Approach," SAND 98-0701, Sandia National Laboratories, Albuquerque, New Mexico, April 1998.
14. Gartling, D. K., and Hogan, R. E., "COYOTE II - A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part I - Theoretical Background," SAND 94-1173, Sandia National Laboratories, Albuquerque, New Mexico, October 1994.
15. Gartling, D. K., and Hogan, R. E., "COYOTE II - A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part II - User's Manual," SAND 94-1179, Sandia National Laboratories, Albuquerque, New Mexico, October 1994.
16. Hassan, B., Kuntz, D. W., and Potter, D. L., "Coupled Fluid/Thermal Predictions of Ablating Hypersonic Vehicles," AIAA Paper No. 98-0168, January 1998.

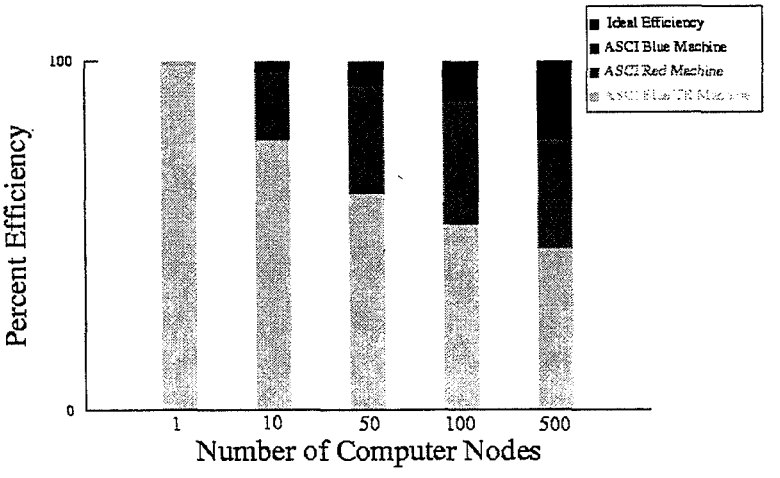


Figure 1. Parallel efficiency of SACCARA on DOE ASCI machines.

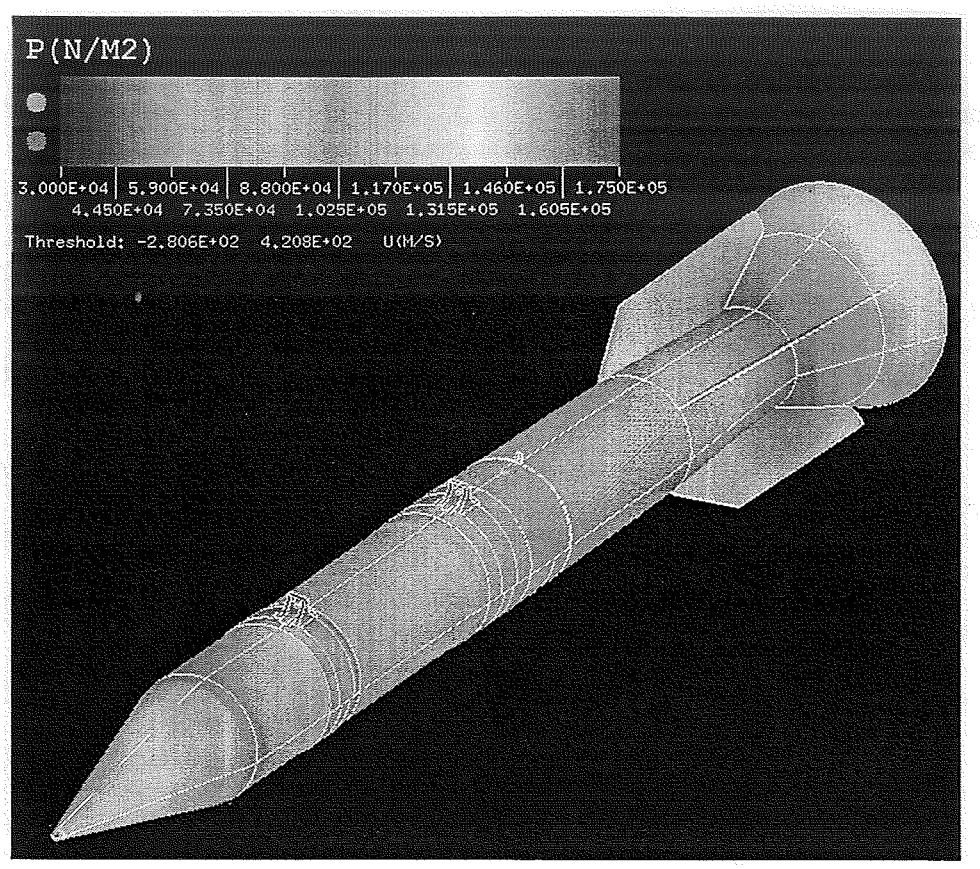


Figure 2. Decomposed block structure and pressure contours on subsonic/transonic vehicle.

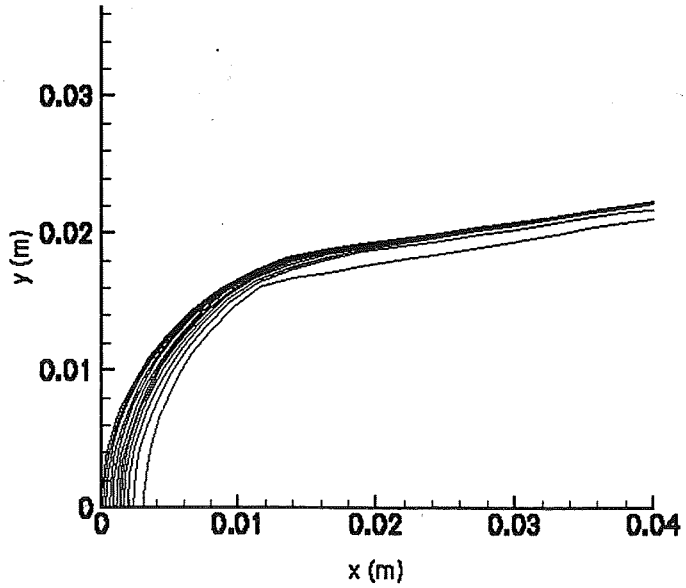
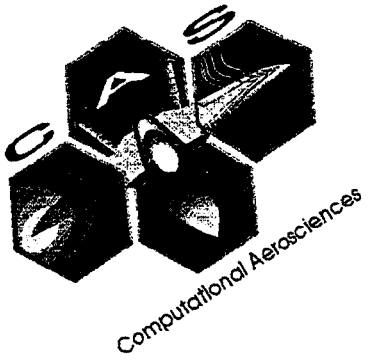


Figure 3. Ablated surface shapes at selected trajectory points for a sphere-cone re-entry vehicle (every other trajectory point shown).

omit this page



Session 6:

Multidisciplinary Design and Applications

DEVELOPMENT OF AN EARTH SYSTEM MODEL IN HIGH PERFORMANCE COMPUTING ENVIRONMENTS

C.R. Mechoso, L.A. Drummond, J.D. Farrara, J.A. Spahr
Department of Atmospheric Sciences
University of California, Los Angeles
405 Hilgard Ave
Los Angeles, CA 90034
drummond@atmos.ucla.edu
(310) 825-9205

Abstract

Under the framework of NASA High Performance Computing and Communications for the Earth and Space Sciences (HPCC-ESS) program, we are developing an Earth System Model (ESM). The ESM produced by this effort will be used for ensembles of climate simulations in high performance computing environments to study the coupled atmosphere/ocean system dynamics with chemical tracers. Here, we outline the design, implementation, optimization and performance of two modules of the ESM: 1) Models (dynamical and chemical), and 2) Data Broker. As illustrated in Figure 1, The Data Broker module is used for handling the communication between the Models module and the Earth System Model Information System (ESMIS) module.

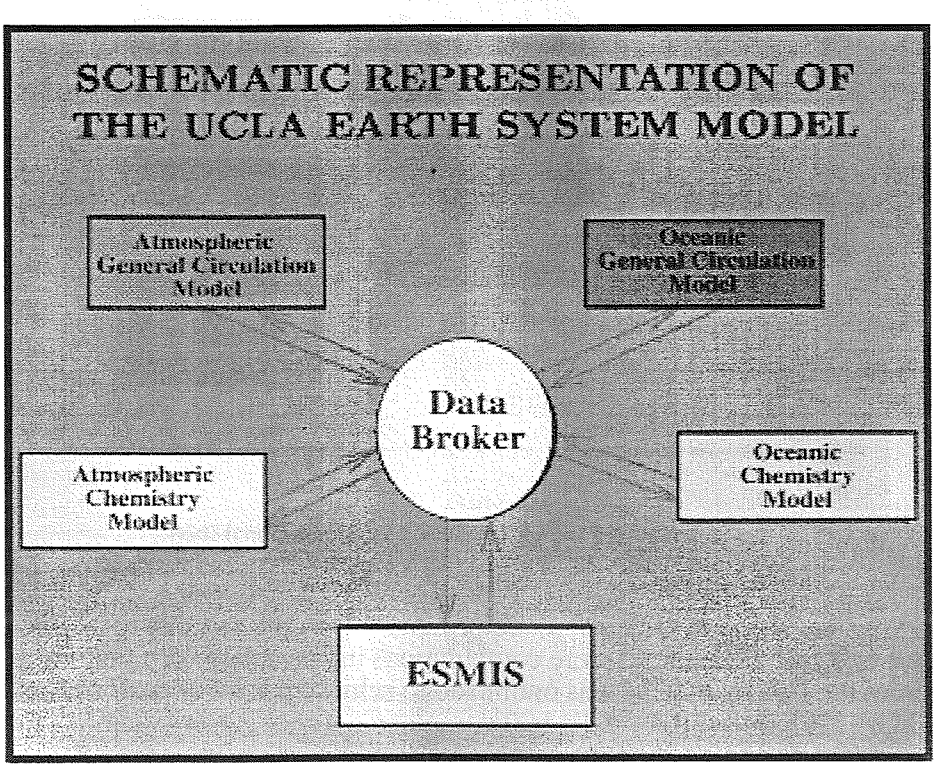


Figure 1
Schematic of the Earth System Model

1. The Models Module

Currently the "Models" module includes an atmospheric general circulation model (UCLA AGCM), an oceanic general circulation model (Parallel Ocean Program, POP), and an atmospheric

chemistry model (UCLA ACM). The UCLA AGCM is a state-of-the-art finite-difference model of the global atmosphere that is constantly undergoing code and algorithmic revisions to improve its physical parameterizations, dynamics and computing (Mechoso et al. [1998]). The UCLA AGCM has been implemented in parallel using a two-dimensional domain decomposition in the horizontal (Wehner et al. 1995). In this decomposition each subdomain is a rectangular region which contains all grid points in the vertical.

Figure 2 presents an schematic of the major components of the UCLA AGCM. The AGCM/Dynamics computes the evolution of the fluid flow governed by the primitive equations and the AGCM/Physics computes the parameterizations. The results obtained by the AGCM/Physics are supplied to the AGCM/Dynamics as forcing for the flow calculations. The parallel performance of the UCLA AGCM in MPP environments is impacted by load imbalances and poor cache reusability in both AGCM/Dynamics and AGCM/Physics. Load imbalances in the AGCM/Dynamics are primarily static and due to the use of polar filters that are required to avoid use of the extremely small timestep necessary to satisfy the Courant Friedrich-Levy (CFL) condition near the pole. Thus, the filters are not applied to all latitudes.

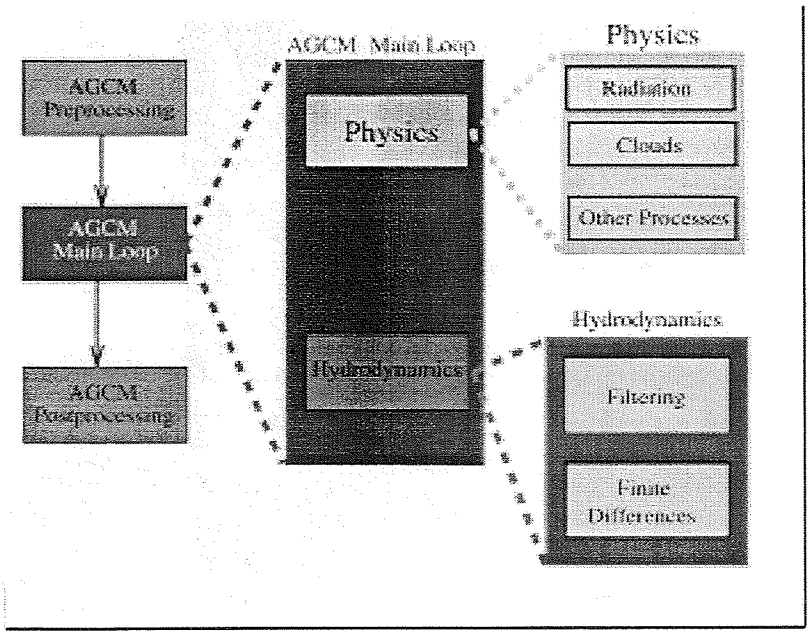


Figure 2
An schematic of the UCLA AGCM computational loop.

In the AGCM/Physics, the load imbalances are primarily dynamic and due to physical processes that vary in time and space leading to more computations in one subdomain than in others. The parallel version of the UCLA AGCM was optimized to achieve a level of performance that approaches 40 GFLOPS on a CRAY T3E-600 computer.

The POP code is also based on a two-dimensional (longitude-latitude) domain decomposition (Smith et al., 1992), and uses message passing to handle data exchanges between distributed processes. The UCLA AGCM has been coupled to POP in a task parallel paradigm (see Figure 3) in which both models periodically exchange information at the air-sea interface during a simulation. The AGCM sends wind stress, heat and water fluxes to POP at the end of each simulated day. POP sends back sea surface temperatures (SSTs) to the AGCM half a simulated day later. The coupled system completes a day of simulation in 54 seconds on 787 T3E-600

nodes using an AGCM resolution of 2.5° lon. x 2° lat and 29 layers and a version of the POP that covers the North Atlantic ocean with a resolution of 1/6° lon. x 1/6° lat. and 37 vertical levels and realistic bottom topography.

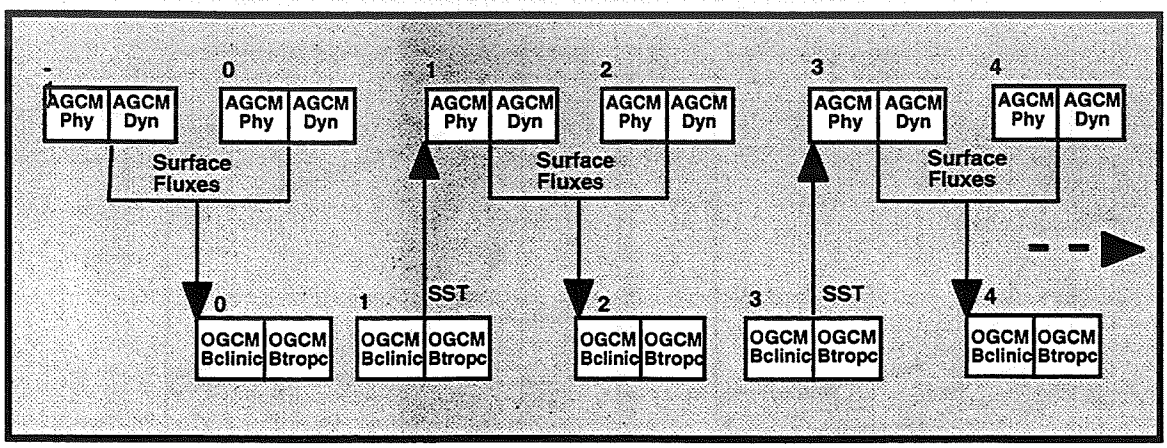


Figure 3

AGCM/POP coupling interval. Both codes run simultaneously and exchange fields at given coupling intervals.

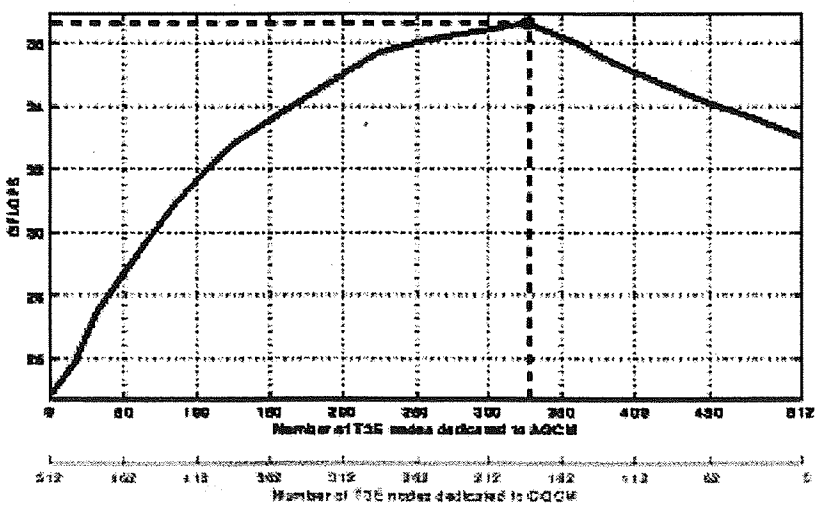


Figure 4

In the X-direction, the upper pannel shows the number of T3E-nodes dedicated to the AGCM and the lower pannel the number of T3E-nodes dedicated to POP.

The UCLA/AGCM and POP are independent codes and ideally can be run in distributed computing environments as long as it can be guaranteed that both models will be running at the same time and are available to be synchronized for coupling. In the CRAY T3D and T3E series, this is only possible when both codes are combined in a single executable. Combining the codes in a single executable sets limits to the resolution of the models in the coupled system because of the individual memory requirements.

Based on the individual performances of the parallel UCLA/AGCM and the POP code, Figure 4 presents a curve of the expected performance (in GFLOPs) of the coupled system after distributing 512-nodes of a CRAY T3E computer between the two models. In this case a maximum of 36.6 GFLOPs was found when dedicating 327 nodes to the AGCM and 185 to the POP code.

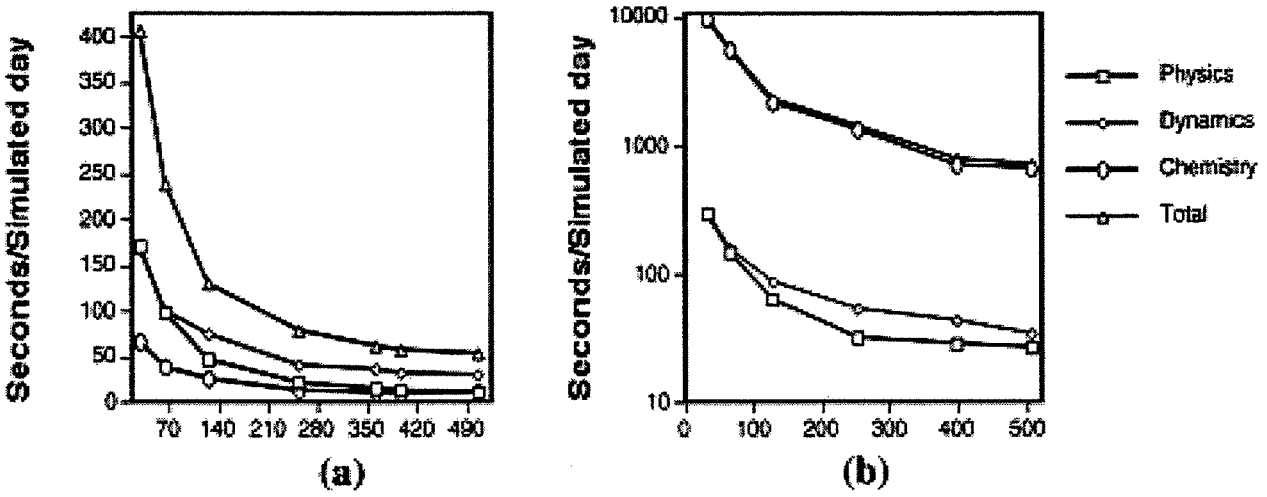


Figure 5

Performance of the AGCM/ACM on CRAY T3E-600. The AGCM resolution is 2.5° lon. x 2° with 29 vertical layers . In Figure 5a, the ACM includes 2 active species (CFC11 and CFC12) and one inactive specie. In Figure 5b, the ACM includes 19 active species and 6 inactive species .

The AGCM has also been coupled to an atmospheric chemistry model (ACM). The ACM describes transformations of chemical active gas and aerosol tracers in the atmosphere and includes algorithms to solve photochemical and thermochemical coupled systems, a detailed treatment of the microphysics of small particles including growth/evaporation, coagulation, sedimentation and deposition, and a fully integrated radiation package (Elliot et all, 1995).

The AGCM/ACM coupling has been implemented in such a way that AGCM and ACM codes are combined in a single executable. We chose this configuration because the ACM uses computational modules (like the advection scheme) that are already available and optimized in the AGCM code. Figure 5 presents the performance of the AGCM/ACM. The timing curves refer to the wall-clock time in seconds per simulated day of the AGCM/Physics, AGCM/Dynamics, Chemistry model and Total AGCM/ACM. In Figure 5-a, the chemistry includes two CFC active species and Ozone as an inactive specie, the AGCM has a timestep of 1 hour while the chemistry model is called every 6 simulated hours. In Figure 5-b, the chemistry includes 19 active species, 6 inactive ones, and the chemistry is called every 2 simulated hours. In the latter case, we observed that most of the CPU time is consumed by the ACM.

2. The Data Broker Module

The "Data Broker" module has been designed and implemented to perform the data exchanges and synchronizations involved in coupling different models with different global resolutions and scales running in distributed environments. The data broker has been implemented in a distributed manner to avoid potential bottlenecks from a centralized processor performing all the Data Broker tasks. The Distributed Data Broker (DDB) consists of three library components: Data Transformation Library (DTL), Model Communication Library (MCL) and the Communication Library (CL).

The DTL contains a set of callable routines for performing data transformations from one grid scale to the other. The MCL contains two sets of callable routines to support communication between applications. A set of routines in the MCL allow a model to register into a coupled system environment using the DDB. During registration, a model specifies all of the data that it will be consuming and/or producing, the geographical domain of data, the size of the computational grid and the frequencies for data production and/or consumption. In addition, there has to be a dedicated process to fill the role of Registration Broker (RB) at registration time. The RB is only active at the beginning of a coupled run and can be used as one of the parallel processes working in one of the models. The RB gathers domain and grid information from models to be coupled, and creates tables that are later used by other MCL routines.

The other set of MCL routines supports high-level data exchanges, in which a process producing data has to make a single call to transfer the produced data to one or more consumers. Likewise, a single MCL call is necessary to request data from one or more producers and the consumer process waits until the requested data has completely arrived. The CL is a set of routines for implementing the actual message passing interface between the models. A single MCL calls gets translated into one or more CL calls depending on the information supplied by consumers and producers at registration time. Currently, the CL is implemented using PVM 3.

Conclusions.

The current performance of the Models module of the UCLA ESM approaches 40 GFLOPS in 768 nodes of a CRAY T3E-600, with chemistry that includes only CFC emission, dispersion, and loss through stratospheric photodissociation. A version of AGCM/OGCM/ACM that runs on the CRAY T3E series and can be ported to other parallel machines is available. An Oceanic Chemistry Model (OCM) is being added to the Models module. We are currently performing high-resolution AGCM/OGCM simulations.

The DDB was successfully tested for global model domains. The DDB is being used in the implementation of a coupled system that involves the UCLA AGCM and a mesoscale model running in a metacomputing environment. In this scenario, the DDB will handle the communication between the models and interface with the Metacomputing system named LEGION.

Acknowledgments.

This work has been supported by CAN-21425/041

References.

Arakawa, A., and W. H. Schubert, 1974: Interaction of a cumulus cloud ensemble with the large-scale environment. Part I., *J. Atmos. Sci.*, **31**, 674-701.

Elliot, S., X. Zhao, R. Turco, C.-Y. Kao and M. Shem., 1995: Photochemical numerics for global scale modeling: Fidelity and GCM: testing. *J. Applied Metereology*, **34**, 694-718.

Harshvardhan, R. Davies, D. A. Randall, and T.G. Corsetti, 1987: A fast radiation parameterization for atmospheric circulation models. *J. Geophys. Res.*, **92**, 1009-1016.

Mechoso, C.R., J.-Y. Yu, and A. Arakawa, 1998: A coupled GCM pilgrimage. From climate catastrophe to ENSO simulations. *General Circulation Model Development: Past, Present, and Future: Proceedings of a Symposium in Honor of Professor Akio Arakawa.*, D. A. Randall, Ed., Academic Press, submitted.

Mechoso, C.R., L.A. Drummond, J.D. Farrara, and J.A. Spahr, 1998b: The UCLA AGCM in High Performance Computing Environments. To appear in Proceedings from SC98.

Smith, R.D., J.K. Dukowicz, and R.C. Malone, 1992: Parallel ocean general circulation modeling. Physica D, **60**, 38-61.

Wehner, M.F., A. A. Mirin, P.G. Eltgroth, W.P. Dannevik, C. R. Mechoso, J. D. Farrara, and J. A. Spahr, 1995: Performance of a distributed memory finite-difference atmospheric general circulation model. Parallel Computing, **21**, 1655-1675.

Parallel Finite Element Solution of 3D Rayleigh - Bénard - Marangoni Flows.

G. F. Carey, R. McLay, G. Bicken, B. Barth, A. Pehlivanov
The University of Texas At Austin

August 25, 1998

Abstract

A domain decomposition strategy and parallel gradient-type iterative solution scheme have been developed and implemented for computation of complex 3D viscous flow problems involving heat transfer and surface tension effects. Details of the implementation issues are described together with associated performance and scalability studies. Representative Rayleigh-Benard and microgravity Marangoni flow calculations and performance results on the Cray T3D and T3E are presented. The work is currently being extended to tightly-coupled parallel "Beowulf-type" PC clusters and we present some preliminary performance results on this platform. We also describe progress on related work on hierarchic data extraction for visualization.

1 Introduction

Our main objective in the present work is to develop effective parallel algorithms and a distributed parallel implementation capable of high resolution 3D coupled flow and heat transfer computations including surface tension effects. This will permit us to make fundamental phenomenological flow studies at the grid resolution necessary to represent the fine scale surface-driven phenomena and to study the associated nonlinear free surface behavior. The discretization involves 3-D isoparametric "quadrilateral brick" finite elements with triquadratic velocity, trilinear pressure and triquadratic temperature approximation on the elements. A non-overlapping domain decomposition of the grid is generated with processor interfaces coincident with a subset of element surfaces. This implies that nodes on the subdomain interfaces are shared by adjacent processors. A secondary objective is to explore hierarchic approaches for extracting and visualizing the solution at a desired fidelity.

2 Discussion

We consider the transient flow of a viscous incompressible fluid as described by the Navier-Stokes equations coupled to the transport of heat by conduction and convection in the fluid. Buoyancy is included by means of the Boussinesq approximation as a temperature dependent body force term in the momentum equations and the velocity field enters the convective term in the heat transfer (energy) equation. The effect of thermocapillary surface tension enters as an applied shear stress which is also dependent on the surface temperature gradient. The Navier Stokes equations for viscous flow of an incompressible fluid may be written

$$\rho(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) + \nabla \cdot \boldsymbol{\tau} = \boldsymbol{f} + \beta(T - T^*)\boldsymbol{g} \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2}$$

in flow domain Ω where $\boldsymbol{u}$ is the velocity field, $\boldsymbol{\tau}$ is the stress tensor and is specified by Stokes hypothesis for a Newtonian fluid, $\boldsymbol{f}$ is an applied body force, $\boldsymbol{g}$ is the gravity vector, T^* is the reference external temperature, T is the fluid temperature and β is the thermal coefficient. At the solid wall boundaries the no slip condition applies so $\boldsymbol{u} = \boldsymbol{u}_w$ where $\boldsymbol{u}_w$ is the specified wall boundary velocity.

At the free surface, a shear stress due to thermocapillary surface tension acts. For example, on a horizontal free surface the tangential shear stress component τ_{zx} is given by

$$\tau_{zx} = \frac{\partial \gamma}{\partial x} = \frac{\partial \gamma}{\partial T}\frac{\partial T}{\partial x} \tag{3}$$

with a similar expression for τ_{zy}, where $\gamma(T)$ is the surface tension and T is temperature. The heat equation is

$$\rho c_p \left(\frac{\partial T}{\partial t} + \boldsymbol{u} \cdot \nabla T \right) - \nabla \cdot (k \nabla T) = Q \tag{4}$$

where k is the thermal conductivity of the fluid, ρ is the density, c_p is the heat capacity, and Q is a heat source term. Temperature, flux or mixed thermal boundary conditions may be applied. For example, in the R–B–M test problem later, we specify temperature $T = T_b$ on the base, zero normal flux $k\partial T/\partial n = 0$ on the side walls, and mixed conditions $k\partial T/\partial n = \alpha(T - T^*)$ (Robin) on the free surface, where T^* is the exterior temperature, and α is the heat transfer coefficient for the medium. Then (1), (2), (4) constitute a coupled system to be solved for velocity, pressure and temperature.

Introducing test functions $\boldsymbol{v}$, q in a weighted-residual statement for (1)-(2), integrating by parts using (3) and introducing the Stokes hypothesis, we obtain the weak variational statement: find the pair $(\boldsymbol{u}, p) \in V \times Q$ with $\boldsymbol{u} = \boldsymbol{u}_w$ at the wall boundaries and such that

$$\begin{aligned} \int_\Omega \frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{v}\, dx + \int_\Omega \boldsymbol{u} \cdot \nabla \boldsymbol{u} \cdot \boldsymbol{v}\, dx + \int_\Omega (\nu \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - p \nabla \cdot \boldsymbol{v})\, dx \\ + \int_\Gamma \frac{\partial \gamma}{\partial T} \nabla T \cdot \boldsymbol{v}\, ds \;=\; \int_\Omega (\boldsymbol{f} \cdot \boldsymbol{v} + \beta(T - T^*)\boldsymbol{g}) \cdot \boldsymbol{v}\, dx \end{aligned} \tag{5}$$

and

$$\int_\Omega \nabla \cdot \boldsymbol{u}\, q\, dx = 0 \tag{6}$$

hold for all admissible $(\boldsymbol{v}, q)$. Similarly, the weighted integral for (4) yields: Find $T \in W$ satisfying any specified (essential) temperature boundary conditions and such that

$$\begin{aligned} \int_\Omega \rho c_p \left(\frac{\partial T}{\partial t} w + \boldsymbol{u} \cdot \nabla T w \right) + k \nabla T \cdot \nabla w\, dx \\ = \int_\Omega Q\, w\, dx \end{aligned} \tag{7}$$

for all admissible test functions w with $w = 0$ on those parts of the boundary where T is specified.

The finite element formulation for (5), (6), (7) follows immediately on introducing the approximation subspaces V_h, Q_h, W_h for V, Q and W respectively.

The fully coupled scheme requires that the linearized coupled system be solved each timestep. Introducing the discretization of elements with finite element basis functions and substituting in the approximate weak statement we have a semidiscrete finite element system of the form

$$M\frac{dU}{dt}s(U) + \nu AU + Bp = F + b(T) \tag{8}$$

$$B^T U = 0 \tag{9}$$

$$M\frac{dT}{dt} + KT + CT = Q \tag{10}$$

A variety of integration schemes are applicable to advance the solution from a specified initial state $U(0)$, $T(0)$. In the results shown later we employ the midstep $\theta = 1/2$ scheme.

Both the coupled and decoupled algorithms require repeated system solves within each timestep. These systems are sparse and nonsymmetric but the asymmetry in the R-B-M microgravity problem is not strong. This implies that iterative methods with Jacobi preconditioning should be quite effective in solving each linear subsystem. In the present work we solve the respective systems in parallel over subdomains using biconjugate gradient iteration (BCG) with diagonal preconditioning.

The main computational steps in this solution algorithm are matrix-vector products, transpose matrix-vector products and vector dot products. Since the matrices are sparse (because of the local support of the element bases) we require fast parallel sparse matrix-vector product (MATVEC) routines in particular and also a fast parallel dot product routine. Frequently, in the parallel iterative literature the MATVEC is left to the user to provide but this is the 'heart' of the calculation and must be handled carefully if a fast scalable parallel solver is to be designed. We elaborate on the parallel MATVEC and dot product routines in the next section.

In the present work we use a *non–overlapping* decomposition by elements. This implies that the processor interfaces coincide with a subset of element faces and the nodes on those faces are shared by adjacent processors. This is natural in a finite element framework since it implies that the element calculations can be parallelized easily over the processor partition and also lends itself to element-by-element solution strategies by either iterative schemes or frontal elimination within each subdomain.

A subdomain element-by-element approach (in which each subdomain contains a sufficient number of elements) provides an efficient parallel strategy. Here the dense matrix operations at the element level can be exploited in the intensive computational kernels and the communication at the processor interface nodes can be overlapped with computation in the interior of each subdomain.

Let us consider a typical interior subdomain Ω_s containing elements w_e^s, $e = 1, 2, ..., E_s$ and let B_j^s, $j = 1, 2, ..., J$ denote those border elements of Ω_s that are adjacent to the subdomain boundary $\partial\Omega_s$. Let I_k^s, $k = 1, 2, ..., K$ denote the remaining elements interior to the subdomain. Computations involving this interior subset are local to the processor. This implies that communication requests can be initiated for the subdomain border element calculations while computation begins on the interior subset. Then the border element computations can be completed. Provided there are sufficient elements in the interior subset, this strategy will permit complete communication overlap.

For simplicity, consider an interior cube subdomain. There are 6 surface subdomain neighbors, 12 edge subdomain neighbors (that are not also surface neighbors) and 8 further vertex neighbors. Hence there are 26 subdomain neighbors involved in message passing to or from any given interior subdomain cube. For the border elements the interior face nodes are duplicated twice, once on each neighboring processor, interior edge nodes are duplicated four times and corner nodes are duplicated eight times. This has some bearing on the way we compute the global dot products in the algorithm. More specifically, a "masked" dot product is computed with masking weight, ω_i for node i. Here ω_i is the reciprocal of the number of subdomains sharing node i. e.g. $\omega_i = 1$ for subdomain interior node i, $\omega_i = \frac{1}{2}$ for face interior node etc.

The element matrix calculations are trivially parallelized over the processor subdomains and the subdomain matrix vector product can be conceptually separated as follows:

For elements e in the border:
    Compute EBE matrix-vector product.
    Sum element result into nodal result vector.
Extract interface nodal MVP results to send buffers and transfer to neighbor processors.
For elements e in the interior
    Compute EBE matrix-vector product.
    Sum element result into nodal result vector.
Sum receive buffer data for neighboring processors into nodal MVP result vector.

We use a combination of MPI and SHMEM to handle the communications. (SHMEM is used where speed is important, but SHMEM is less portable.) For example, global operations with SHMEM require a buffer at the same address on all processors. Hence, for operations like dot products, only one global location is needed and SHMEM is used. Operations that require broadcasting buffers of unknown size, are more difficult and we use MPI here.

MGFLO is designed for unstructured mesh finite element simulations on irregular partitions such as those generated by Chaco [?] or Metis [?]. This flexibility is facilitated by means of lists in C. Each processor has a set of elements which is further broken down to (1) "frame" elements which have a face, edge or corner shared with elements residing on another processor and (2) the remaining "interior" elements on the processor. Accordingly, we use three lists of pointers–one list is to all elements of the processor and the other two to the respective frame and interior elements. Note that elements are only stored once but there are two pointers to each element. To process elements we simply loop over the appropriate list (all elements, frame elements or interior elements). We remark that essential boundary conditions are stored nodally whereas flux data is stored by elements. Communication between processors is implemented using a list of nodes that are shared by each processor.

In the numerical studies presented later we consider only structured subdomains and use a simple mesh generator that can be partitioned simply. That is, the `mesh` routine knows that it has a structured mesh to create and partition. It also creates the three element lists and the list of nodes that need to be exchanged between processors. (For more general partitionings and grids one must set up the communication node lists accordingly).

3 Results

Beowulf-type Clusters

We have recently assembled a Beowulf-type cluster comprised of 16 Intel Pentium II processors for the investigation of parallel computing on workstation clusters. Each node is equipped with a single Pentium II processor with 128 MB of RAM. The processors are interconnected with a crossbar hub running 100 MBit Fast Ethernet.

Concurrently with this network performance study, the flow analysis program with MPI has been implemented for this architecture and preliminary performance studies have been carried out. Figure 1 shows the results of a scaled speed-up study on our Beowulf-cluster. Four cases were examined in this study: Coupled and Decoupled flow and heat transfer with non-optimized FORTRAN-coded BLAS and Coupled and Decoupled flow and heat transfer with Pentium II optimized assembly-coded BLAS. In the non-optimized case, the results show almost linear scaling for both coupled and decoupled flow regimes, with the variation from linearity due to system software

running in the background. In the optimized case, linear scaling is seen up to 8 or 9 processors after which the performance begins to tail off. This degradation in performance may be showing the communications boundedness of these workstation clusters as computation begins to out run communication.

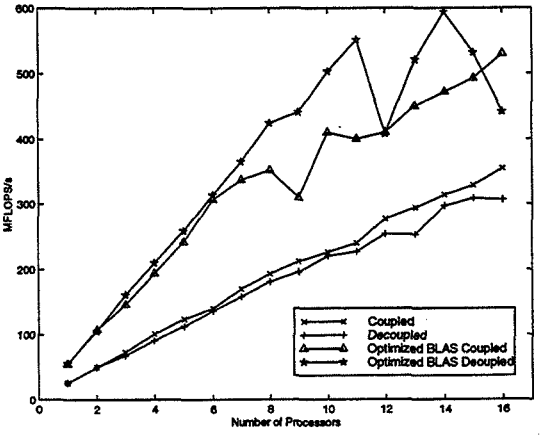


Figure 1: Parallel speedup scaling on the Beowulf cluster

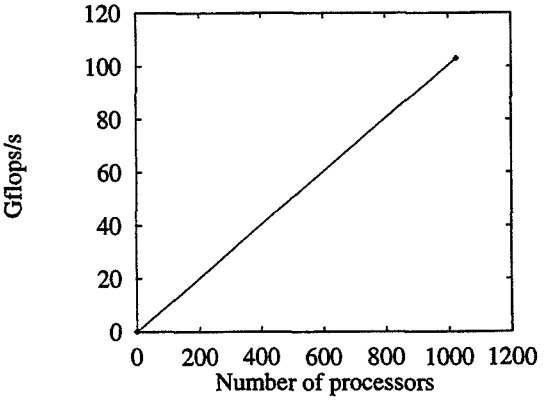


Figure 2: Parallel speedup scaling on the T3E-900

T3E Thermocapillary Studies

A study was first undertaken for scaling through 1024 processors on the T3E-900 as shown in Figure 2 and delivers a maximum of approximately 118 gigaflops. This result is for scaled problem size (the subgrid size per processor is fixed and sufficiently large.

As a phenomenological study, we then considered the fluid flow in a $L \times L \times L/4$ domain driven by an axisymmetric Gaussian heat flux distribution given by $q = \exp(-50((x-0.5)^2 + (y-0.5)^2))$ and $d\gamma/dT = 5$ N/m-K on the top surface of the domain. Here $L = 1$ m is taken as the reference length of the problem. On the other faces of the domain, Dirichlet boundary conditions are applied (i.e., zero velocity, reference temperature $T = 10°$ C). The Prandtl number is 0.3 and we simulate the flow under low gravity ($g = 0.01\ m/s^2$) on a $16 \times 16 \times 8$ mesh with an $8 \times 8 \times 4$ processor

partitioning. The Rayleigh and Marangoni numbers are 761 and 43.25, respectively, where the reference temperature difference $\Delta T = 10.38°C$ is taken to be the difference between the maximum local temperature in the flow and the reference temperature. The flow pattern and temperature distribution on the mid vertical ($y = 0.5$ m) plane are given in Figure 3. The maximum local velocity obtained is 27.7 m/s.

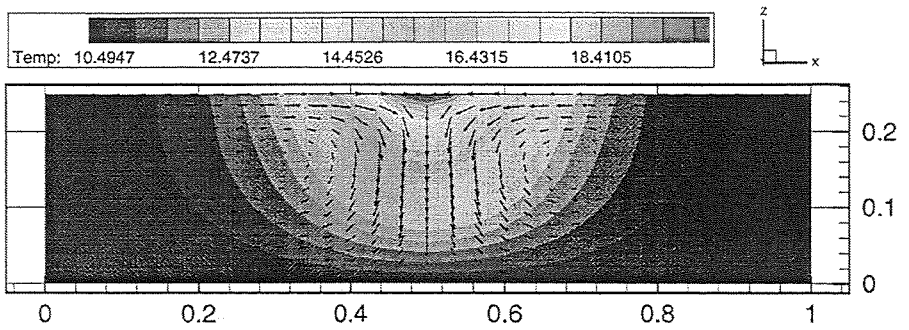


Figure 3: Temperature plot and vector velocity plot on the section $y = 0.5$.

4 Hierarchic Adaptive Extraction

Part of our work involves techniques for hierarchic visualization of very large scale data sets such as those obtained in the previous flow simulations. The basic approach is motivated by the ideas used in adaptive mesh refinement and coarsening strategies, multigrid solution schemes and wavelet multiresolution techniques. The work exploits tree-traversal algorithms and data structures for adaptive refinement/coarsening to enable selective hierarchic extraction and compression of solution data for visualization locally or at a remote site. Error or feature indicators provide a theoretical framework to guide adaptive mesh schemes and provide a mechanism for selectively screening the simulation results. A prototype "data handler" that incorporates some of these primitives is under development and testing. Representative examples involving meshes of "quadrilateral brick" elements in 3D are investigated. These illustrate, for instance, how selective hierarchic visualization using a feature indicator can be applied. Supporting timing studies for remote visualization of hierarchic data using nested meshes or multiresolution approaches are also presented. The extension of these schemes to parallel distributed data sets using mesh partitioning strategies is also considered.

In the most basic case the results are known at all levels of a uniformly refined structured grid and there is a supporting uniform tree data structure. Then this tree can be traversed to collect data from a given level or different levels to be sent for local or remote visualization. Similarly, if the grid is generated by uniform refinement of a coarse unstructured grid, then the same basic approach is applied. Note that the tree can be traversed either from the top level (leaf level) down to the lowest (root) level or vice versa. Since we are interested in a minimalist approach to large data sets, it will generally be preferable to begin at the root level and work up the tree selectively refining rather than at the leaf level working down the tree with selective coarsening. However, this is secondary since the main issue is the size of the data to be transferred and this is the same independent of the direction of traversal.

A preferable approach is to reconstruct an adaptively graded mesh using some quasi–uniform coarse level grid and the approximate solution or problem data as the grading function to guide refinement. For example, we can superpose a containing cube on the applications grid, then refine to an octet. Subcubes are then selectively refined based on the application grid density function or a feature/error indicator computed from the solution on the application grid. Several variations on this theme can be easily deduced. Note that this form of hierarchic approach also permits easy parallel implementation as described later.

Significant mesh irregularity is allowed; i.e. any adjacent elements do not necessarily lie on adjacent tree levels. In the present implementation, the numbering of the leaf elements corresponds to the Morton space-filling curve which will be used in the parallel implementation for load balancing.

In order to demonstrate the procedure, data was adaptively extracted for a function with larger gradients defined on the unit cube. Then a slice through the cube is taken. The image from the extracted data set in is very similar to that obtained from the full data set.

Composing a hierarchic visualization from multiple processors may be achieved by parallel generation of a compressed file and then transfer of the compressed data for viewing. Since this will consist essentially of nodal data for elements at a given level it can be handled in the proposed visualization framework. More specifically, We implemented the RemoteBroker on a Cray T3E. Processor 0 is responsible for the socket communications with the LocalBroker and VisBroker. Also, it synchronizes the other processors. As a test example, we approximated an analytical function using the relative error in L^2-norm as a feature indicator. In our case the threshold level was set to 0.001. Also, the maximum element level (i.e. the tree depth) was limited to 5. Adaptive octree without restriction on mesh "irregularity" was used. For this test the domain (the unit cube) was subdivided into 8 equal subcubes and each of them assigned to a different processor. Because of the fixed domain partition, the busiest processor was handling approximately 1/6th of the total work. The CPU times (in seconds) for octree creation, octree processing (assigning local node numbers, etc.) are determined. Also the wall-clock time for MPI communications and sending data through sockets is presented. Approximately 1.5 MB reduced data was sent to the local machine (SGI Onyx). These timing results have been compared to those in the case of a single processor. The speedup is approximately equal to 6, as expected.

Concluding Remarks

The present work is part of a grand challenge HPC study funded by NASA to investigate scalable parallel coupled viscous flow and heat transfer analysis to explore microgravity and thermocapillary free surface effects. The basic analysis code is, however, more general and provides a framework for scalable distributed HPC applications to this class of transport processes of interest to NASA, industry and elsewhere. We have developed a Galerkin finite element formulation and solution algorithm for both fully coupled and decoupled strategies with an implementation using `noweb` and C. Noweb enables the use of alternate compiler systems and enhances portability of the code (e.g. the "same" code runs MPI only on our Beowulf system and both MPI and SHMEM on the Cray T3E). The program is also designed for analysis with unstructured grids and irregular partitions, this being achieved through the use of lists.

Acknowledgments

This research has been supported by NASA ESS Grand Challenge Project NCCSS- 154 and by DARPA grant DABT63–96-C-0061. We express our appreciation to D. Cline, H. Swinney and S. van Hook for their suggestions, and to S. Swift, C. Harle for and V. Carey their assistance.

References

[1] Carey, G. F. (Ed). 1989. Parallel Supercomputing: Methods, Algorithms and Applications, John Wiley & Sons, U.K.

[2] Gillion, P., and G. M. Homsy. Combined Thermocapillary-Bouyancy Convection in a Cavity: An Experimental Study. *Physics of Fluids,* **8**(11):2953-2963, 1996.

[3] Koschmieder, E. L., and DS. A. Prahl. Surface-Tension-Driven Bénard Convection in Small Container. *Journal of Fluid Mechanics,* **215**:571-583, 1990.

[4] McLay, R., S. Swift, and G. F. Carey. 1996. "Maximizing Sparse Matrix-Vector Product Performance on RISC based MIMD Computers." J. of Parallel and Distributed Computing, 37, 146-158.

[5] Schatz, M. F., S.J. Vanhook, W.D. McCormick, J.B. Swift, H.L. Swinney: "Instability and transition to disorder in surface-tension driven Bénard convection" In preparation.

Engineering Overview of a Multidisciplinary HSCT Design Framework Using Medium-Fidelity Analysis Codes

R. P. Weston, L.L. Green, A.O. Salas, J.A.Samareh, J. C. Townsend, J.L. Walsh
MultiDisciplinary Optimization Branch
NASA Langley Research Center, Hampton, VA 23681-2199

Abstract

Submitted for the NASA Computational Aerosciences Workshop 98, NASA Ames Research Center, Aug. 25-27, 1998.

An objective of the HPCC Program at NASA Langley has been to promote the use of advanced computing techniques to more rapidly solve the problem of multidisciplinary optimization of a supersonic transport configuration. As a result, a software system has been designed and is being implemented to integrate a set of existing discipline analysis codes, some of them CPU-intensive, into a distributed computational framework for the design of a High Speed Civil Transport (HSCT) configuration. The proposed paper will describe the engineering aspects of integrating these analysis codes and additional interface codes into an automated design system. Information about the CORBA-based computational environment will appear in a separate paper (ref. 1).

The objective of the design problem is to optimize the aircraft weight for given mission conditions, range, and payload requirements, subject to aerodynamic, structural, and performance constraints. The design variables include both thicknesses of structural elements and geometric parameters that define the external aircraft shape. An optimization model has been adopted that uses the multidisciplinary analysis results and the derivatives of the solution with respect to the design variables to formulate a linearized model that provides input to the CONMIN optimization code, which outputs new values for the design variables.

The analysis process begins by deriving the updated geometries and grids from the baseline geometries and grids using the new values for the design variables. This free-form deformation approach (ref. 2) provides internal FEM grids that are consistent with aerodynamic surface grids. The next step involves using the derived FEM and section properties in a weights process to calculate detailed weights and the center of gravity location for specified flight conditions.

The weights process computes the as-built weight, weight distribution, and weight sensitivities for given aircraft configurations at various mass cases. Currently, two mass cases are considered: cruise and gross take-off weight (GTOW). Weights information is obtained from correlations of data from three sources: 1) as-built initial structural and non-structural weights from an existing database, 2) theoretical FEM structural weights and sensitivities from Genesis, and 3) empirical as-built weight increments, non-structural weights, and weight sensitivities from FLOPS.

For the aeroelastic analysis, a variable-fidelity aerodynamic analysis has been adopted. This approach uses infrequent CPU-intensive non-linear CFD (ref. 3) to calculate a non-linear correc-

tion relative to a linear aero calculation for the same aerodynamic surface at an angle of attack that results in the same configuration lift. For efficiency, this nonlinear correction is applied after each subsequent linear aero solution during the iterations between the aerodynamic and structural analyses. Convergence is achieved when the vehicle shape being used for the aerodynamic calculations is consistent with the structural deformations caused by the aerodynamic loads. To make the structural analyses more efficient, a linearized structural deformation model has been adopted, in which a single stiffness matrix can be used to solve for the deformations under all the load conditions.

Using the converged aerodynamic loads, a final set of structural analyses are performed to determine the stress distributions and the buckling conditions for constraint calculation. Performance constraints are obtained by running FLOPS using drag polars that are computed using results from non-linear corrections to the linear aero code plus several codes to provide drag increments due to skin friction, wave drag, and other miscellaneous drag contributions.

The status of the integration effort will be presented in the proposed paper, and results will be provided that illustrate the degree of accuracy in the linearizations that have been employed.

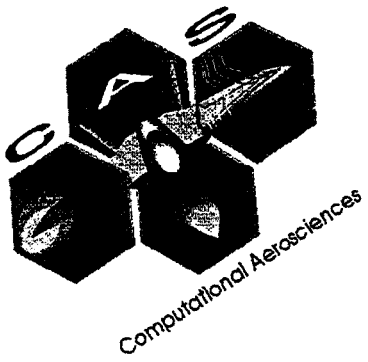
References:

1. Sistla, Raj, et al; "An Object Oriented Framework for HSCT Design," Submitted to the NASA Computational Aerosciences Workshop, August, 1998.

2. Samareh, J. A.; "Geometry Modeling and Grid Generation for Design and Optimization," ICASE/LaRC/NSF/ARO Workshop on Computational Aerosciences in the 21st Century, Hampton, VA, April 22-24, 1998.

3. Biedron, R.T., et al; "Parallel Computation of Sensitivity Derivatives With Application to Aerodynamic Optimization of a Wing," Submitted to the NASA Computational Aerosciences Workshop, August, 1998.

OMIT THIS PAGE



Session 7:

Multidisciplinary Design and Applications

Edward J. Hall
Rolls-Royce Allison
2001 S. Tibbs, T-14A
Indianapolis, IN 46206-0420
Edward.J.Hall@allison.com
(317) 230-3722

TURBINE ENGINE HP/LP SPOOL ANALYSIS

Abstract

The primary objective of this task is to develop and demonstrate the capability to analyze the aerodynamics in the complete high pressure (HP) and low pressure (LP) subsystems of both the NASA/General Electric (GE) Energy Efficient Engine (EEE) and the Allison AE3007 engine using three-dimensional Navier-Stokes numerical models. The analysis will evaluate the impact of steady aerodynamic interaction effects between the components of the HP and LP subsystems. The computational models shall be developed from the geometric components contained in the HP and LP subsystems including: engine nacelle, inlet, fan blades, bifurcated bypass and core inlet, bypass vanes, core inlet guide vanes, booster stage, core compressor blades, high pressure turbine blades, low pressure turbine blades, mixer, and exhaust nozzle. Predictions will be obtained using the ADPAC aerodynamic analysis code. The analysis shall be performed and optimized for workstation cluster computing platforms using parallel processing techniques. The concept of "zooming" in the analysis shall be demonstrated by substituting a lower order cycle model of the HP or LP subsystems using results from the National Cycle Program (NCP). Ultimately, the analysis will include the effects of operation at angle of attack by modeling a complete rotation of the fan wheel.

Introduction

The motivation for this program is based on three primary elements. First, the NASA vision for technological advances under the Computational Aerosciences (CAS) and High Performance Computing and Communications programs are focused under the Numerical Propulsion System Simulation (NPSS) project. The NPSS system is intended to provide a computational framework for design and analysis of complete gas turbine engine systems. Second, the growing maturity of computational fluid dynamics (CFD) tools, particularly with respect to turbomachinery flows. Finally, competitive trends in the gas turbine industry which seek to reduce engine development time and cost, while improving engine performance.

This project is dedicated to develop and demonstrate the capability to analyze the aerodynamic performance of the complete high pressure (HP) and low pressure (LP) subsystems of two modern gas turbine engines as part of the NPSS system capability. The engines considered are the NASA/GE Energy Efficient Engine (EEE) and the Rolls-Royce Allison AE3007 engine. The analysis employs combined 3-D Navier-Stokes and simplified

cycle performance analyses to represent the performance of individual subsystems in the overall engine simulation.

A primary objective of this research is to demonstrate the NPSS goal of*zooming* in the simulation of a complete propulsion system.*Zooming* is losely defined as the ability to substitute models of varying levels of fidelity (1-D, 2-D, 3-D, etc.) for subsystems in the overall engine simulation. To achieve this goal, this study focuses on employing combined analyses with both essentially 0-D cycle deck performance analyses based on the National Cycle Program (NCP) and complex 3-D analyses based on the ADPAC Navier-Stokes analysis tool. As a first step in this procedure, NCP performance models are developed for the complete EEE and AE3007 engines. The sections which follow briefly describe tasks related to the development of the complete NCP and CFD models for the EEE and AE3007 engines, respectively.

Description of the Energy Efficient Engine

The Energy Efficient Engine (EEE) program [1] was developed to create fuel saving technologies for transport aircraft engines whichwould be introduced into service in the late 1980's and 1990's. The EEE development cycle included candidate engines from two manufacturers: Pratt & Whitney and General Electric. Both manufacturers designed and tested components as part of the technology demonstrations necessary to validate the final engine designs. The General Electric design was selected for engine testing, and included separate tests of the core and integrated core/low spool (ICLS) configurations. The flight propulsion system was projected to have a thrust specific fuel consumption of 0.551 lbm/hr/lbf at the maximum cruise design point (35,000 ft. ISA). The ICLS achieved a static corrected take-off thrust of 37,415 lbf. Technology developed during the EEE program has since been applied in the development of modern, high bypass ratio turbofan engines such as the GE90 and the Pratt and Whitney 4084. Both the GE90 and P\&W 4084 engines were recently flight certified by the FAA and are currently being introduced on Boeing's latest commercial aircraft, the Boeing 777 (see Figure 1).

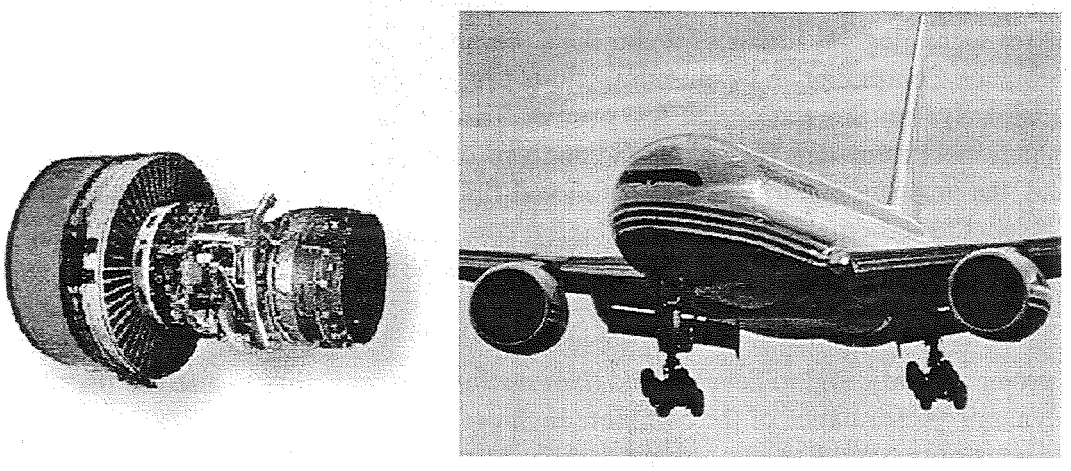


Figure 1. General Electric Energy Efficient Engine test hardware and current Energy Efficient Engine technology aircraft application (Boeing 777).

Description of the Allison AE3007 Engine

The Allison AE 3007 (pictured in Figure 2) is a 7,000-9,000 pound thrust-class turbofan engine designed for the growing regional jet and medium-to-large business jet markets. This engine is part of the Allison*common core* engine family, and is therefore closely related to the T406 (V-22 tilt-rotor aircraft) and AE 2100 (C-130J transport aircraft) turboprop engines. As such, the AE3007 engine benefits from the shared development and operational experience of its common core engine applications.

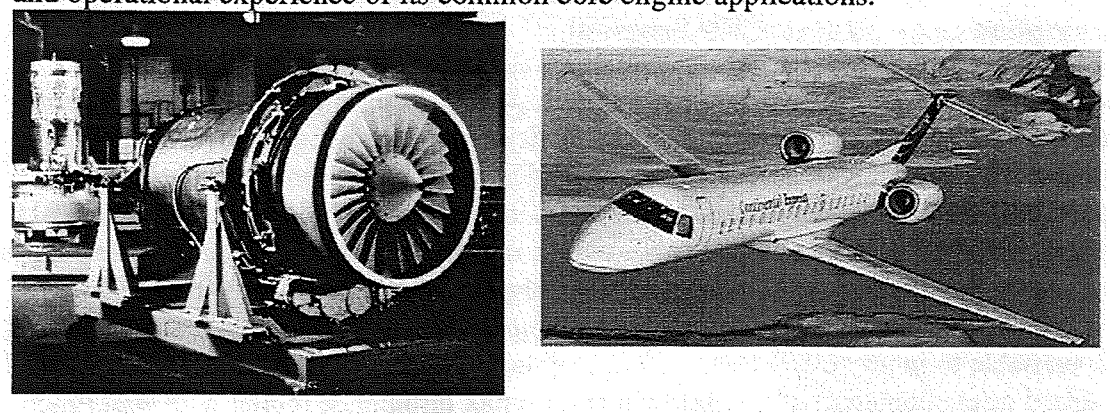


Figure 2. Allison AE3007 engine and installation on the Embraer RJ-145 regional airliner.

The AE3007 engine family features a dual spool (separate high pressure (HP) and low pressure (LP) shafts) configuration. The AE3007 engine cycle employs a moderate 5:1 bypass ratio, with an overall pressure ratio of 23:1. The engine's operating cycle balances the requirements of low fuel burn and operability/reliability margin while low emissions and low noise characterize the AE 3007 as a good neighbor. The AE3007 engine series (A, C, and H designations) provides power for a number of high performance aircraft. The AE3007A powerplant is featured on the Embraer (Empresa Brasileria de Aeronautica S.A.) RJ145 50-passenger regional jet, and the Embraer RJ135 35-passenger regional jet (see e.g. Figure 2). The AE3007C (a derated version of the AE3007A) was developed for the Cessna Citation X business jet. The AE 3007H was selected to power the Teledyne Ryan *Global Hawk* high altitude, long range unmanned surveillance aircraft.

Description of the National Cycle Program

The National Cycle Program (NCP) provides an architectural framework for NPSS project. The initial focus of the NCP is on the aerothermodynamic cycle process. It is a catalyst for establishing new standards for interfacing with tools of different disciplines. The NCP architecture provides the capability to zoom to models of greater fidelity, and will couple more directly to modeling tools for other disciplines. Representatives from government and the aeropropulsion industry determined that an object-oriented approach would meet and exceed the and simulation requirements of the aerothermodynamic cycle simulation process while also creating an extensible framework for the NPSS system. To ease the integration with external applications Common Object Request Broker Architecture (CORBA) was selected.

The NCP is the first step toward building a complete object-oriented architectural framework for NPSS. This framework successfully demonstrates the capability of the object-oriented paradigm to model a complex aeropropulsion process. Integration with external codes using CORBA has proved to be a viable high performance option to solving distributed code coupling problems. Detailed engine component hierarchies can now be created within the NCP architecture with components inside or outside its boundaries.

Description of the *ADPAC* CFD Program

The aerodynamic predictions for the cases described in this study were obtained using the *ADPAC* analysis code. The *ADPAC* code is a general purpose aerospace propulsion aerodynamic analysis tool which has undergone extensive development, testing, and verification [2] [3]. The *ADPAC* analysis solves a time-dependent form of the three-dimensional Reynolds-averaged Navier-Stokes equations using a proven time-marching numerical formulation. The numerical algorithm employs robust numerics based on a finite volume, explicit Runge-Kutta time-marching solution algorithm. Several steady-state convergence acceleration techniques (local time stepping, implicit residual smoothing, and multigrid) are available to improve the overall computational efficiency of the analysis. A relatively standard implementation of the Baldwin and Lomax turbulence model with wall functions is employed to compute the turbulent shear stresses and turbulent heat flux.

An attractive feature of the *ADPAC* code is the versatility and generality of mesh systems upon which the analysis may be performed. The*ADPAC* code permits the use of a multiple-blocked mesh discretization which provides extreme flexibility for analyzing complex geometries. The block gridding technique enables the coupling of complex, multiple-region domains with common (non-overlapping) grid interface boundaries through specialized user-specified boundary condition procedures. *ADPAC* supports coarse-grained computational parallelism via block boundary-specified message passing. Interprocessor communication is controlled by the Message Passing Interface (MPI) communication protocol. Parallel computations employed during this study utilizied a wide range of high speed processors, workstation clusters, and massively parallel computing platforms,.

Steady-state aerodynamic predictions for multistage turbomachinery are performed using a specialized boundary procedure known as a "mixing plane". The mixing plane strategy was developed to permit numerical simulations based on only a single blade passage representation for each blade row, regardless of the differences in circumferential spacing for each blade row. This simplification is afforded by circumferentially averaging data on either side of the interface between blade rows (the mixing plane), and then passing that information as a boundary condition to the neighboring blade row. An algebraic model representing the time-averaged effects of deterministic unsteadiness is employed to provide a more realistic simulation of complex turbomachinery flows.

LP Subsystem Simulation

Mesh systems for the EEE and AE3007 engines were generated using algebraic constuction techniques. Individual blade row meshes employ an H-type format with mixing plane boundary conditions employed between adjacent blade rows. Figure 3 illustrates the geometry and axisymmetric projection of the mesh system employed for the EEE simulation.

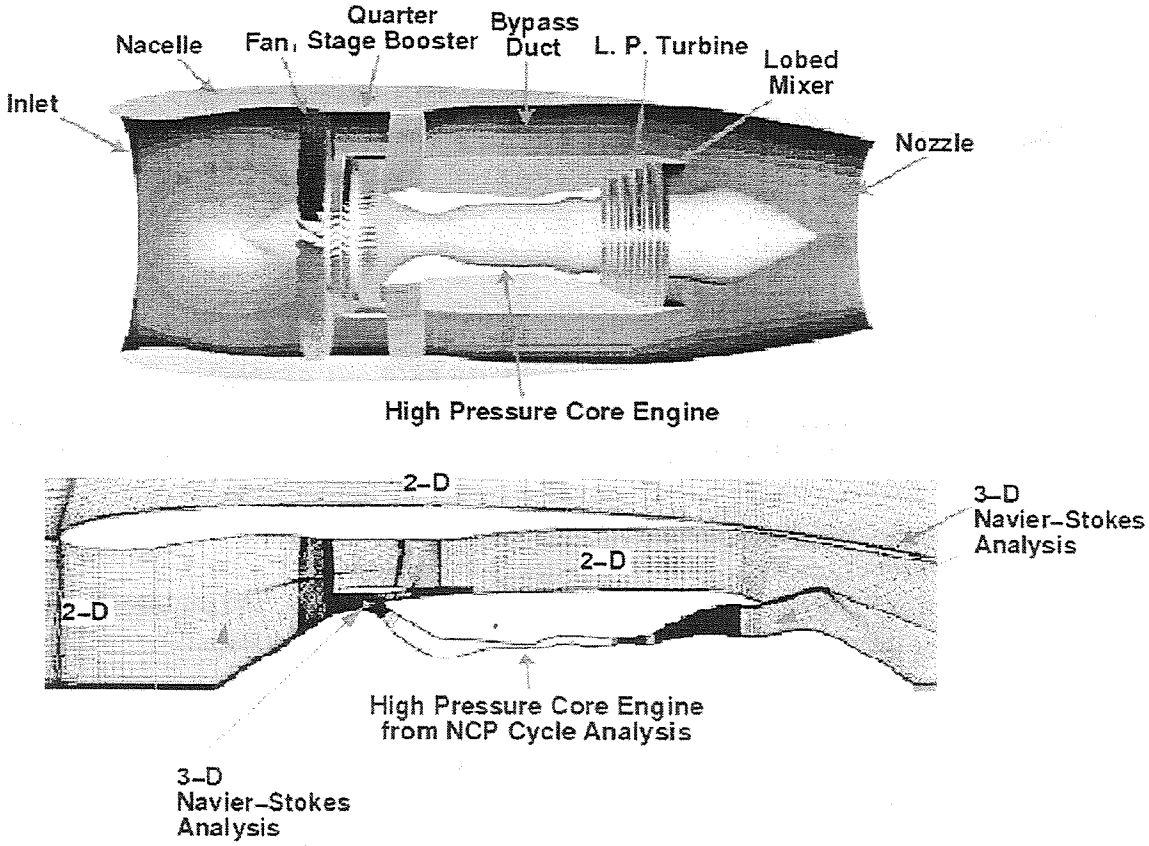


Figure 3. Energy Efficient Engine High Pressure/Low Pressure Spool Analysis illustrates coupling between 3-D CFD simulation (ADPAC) and engine cycle simulation (NCP).

The corresponding NCP model for the EEE engine is illustrated in Figure 4. Individual engine components are simulated in block fashion using the object-oriented system simulation capability afforded by the NCP architecture.

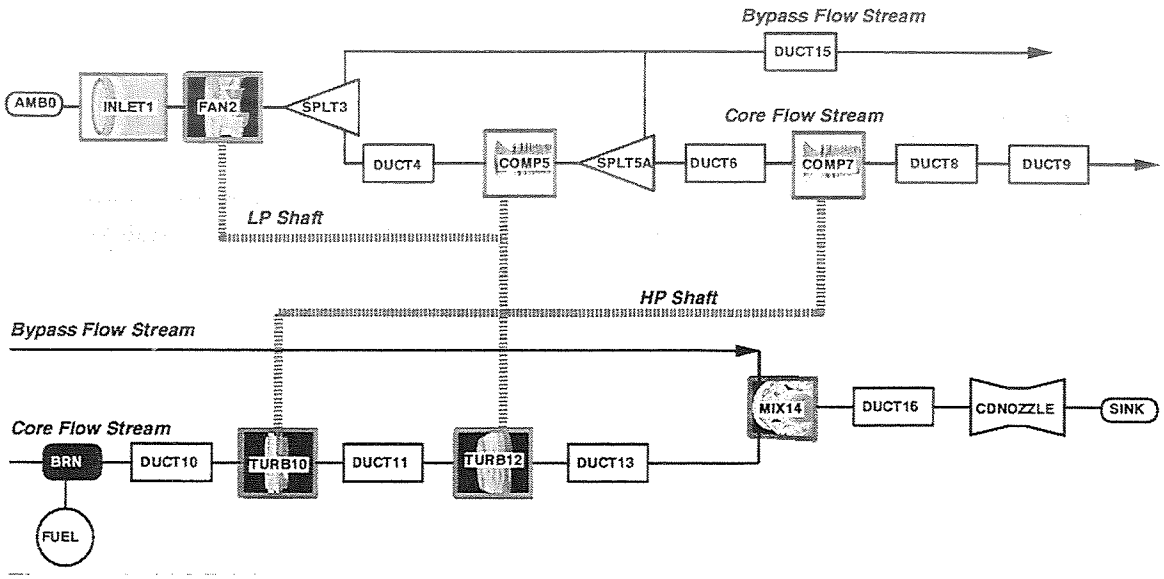


Figure 4. NCP block model diagram for the EEE engine.

Multi-disciplinary coupling is provided in both the NCP and ADPAC models by equating power requirements for common shaft-mounted components. For example, the equilibrium speed of the LP shaft is determined by equating the power requirements of the LP compression system with the power provided by the LP turbine system. A shaft power balance procedure has been developed which iteratively alters the LP shaft rotational speed until a proper power balance is provided. In practice, for the large scale CFD simulations, this iterative process must be underrelaxed slightly, and requires 5-6 global iterations until the component power balance is within a 1% tolerance.

Conclusions

A combined CFD/cycle gas turbine engine performance analysis procedure has been developed and is currently being applied to two modern gas tirbine engine designs. Multidisciplinary coupling is established in the large scale CFD simulations through a shaft power balance procedure. Coupling between the CFD simulations and the reduced order cycle analysis is accomplished through a CORBA interface controlled by an external JAVA application. This procedure will continue to be refined as part of the overall NPSS full engine analysis capability.

References

[1] Saunders, N. T., Colladay, R. S., and Macioce, L. E., "Design Approaches to More Energy Efficient Engines," AIAA and SAE 14th Joint Propulsion Conference, Las Vegas, Nevada, July 25-27, 1978.

[2] Hall, E. J., "Aerodynamic Modeling of Multistage ompressor Flowfields-Part 1: Analysis of Rotor/Stator/Rotor Aerodynamic nteraction," ASME Paper 97-GT-344, 1997.

[3] E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields-Part 2: Analysis of Rotor/Stator/Rotor Aerodynamic Interaction," ASME Paper 97-GT-345, 1997.

PARALLEL AEROELASTIC ANALYSIS USING ENSAERO AND NASTRAN

Lloyd B. Eldred*, Chansup Byun†, and Guru P. Guruswamy‡
Applied Computational Aerodynamics Branch
NASA Ames Research Center, Moffett Field, California 94035-1000

Abstract

A high fidelity parallel static structural analysis capability is created and interfaced to the multidisciplinary analysis package ENSAERO-MPI of Ames Research Center. This new module replaced ENSAERO's lower fidelity simple finite element and modal modules. Full aircraft structures may be more accurately modeled using the new finite element capability. Parallel computation is performed by breaking the full structure into multiple substructures. This approach is conceptually similar to ENSAERO's multi-zonal fluid analysis capability. The new substructure code is used to solve the structural finite element equations for each substructure in parallel. NAS TRAN/COSMIC is utilized as a front end for this code. Its full library of elements can be used to create an accurate and realistic aircraft model. It is used to create the stiffness matrices for each sub-structure. The new parallel code then uses an iterative preconditioned conjugate gradient method to solve the global structural equations for the substructure boundary nodes. Results are presented for a wing-body configuration.

Introduction

Accurate structural modeling of a real aircraft by discretization has constraints that are completely independent from those faced by the aerodynamics discipline. A structural model focuses on the main internal features of the aircraft: the wing's spars and ribs and the fuselage's bulkheads and stringers. An aircraft aerodynamic model focuses on critical aerodynamic features: regions of separation, shocks, etc. These major features of interest are completely unrelated to each other. An attempt at using common meshes is at best doomed to inefficiency, and more likely to failure. A much more efficient and powerful approach is to interface the highest fidelity single discipline technologies available. ENSAERO[1-3] implements the Reynolds averaged thin-layer Navier-Stokes equations. NASTRAN[4]'s element library allows the accurate finite element modeling of the aircraft components as plates, bars, and beams and the sub-structure based structural system solver allows for efficient parallel solution of the structural equations.

A fluid-structure interface calculation is performed on the aircraft skin. Fluid forces cause structural loading, causing deflection, which in turn changes the fluid field. Since the fluid surface grid does not, in general, correspond to the structural grid on the aircraft skin, an interpolation/extrapolation scheme is used to transfer the fluid loads to the structural nodes and the structural deflections to the fluid grid.

This work builds on earlier domain decomposition work by Byun and Guruswamy[1-3]. That work involved interfacing the Navier Stokes/Euler solver with structural models. The structural models included a modal model and a parallel finite element model, using a partitioning approach.

Approach

A high fidelity parallel static finite element capability, "ENSAERO-FE", has been developed. Parallel computations are performed on multiple substructures with an iterative scheme used to calculate the boundary values. A standard finite element package, in this case NASTRAN/COSMIC, is used as a preprocessor to generate the substructure stiffness matrices. The new parallel code solves the system of equations. Figure 1 shows how the structural solution is calculated in parallel.

The interactive parallel solution of the finite element system is strongly based on that proposed by Carter, et. al.[5]. It uses a preconditioned conjugate gradient method. That scheme has been adapted to run on the IBM SP-2 and SGI Origin 2000 at NASA, Ames Research Center using the MPI for interprocess communication.

To use this scheme, the full structure is broken into substructures. The finite element stiffness matrices are assembled for each substructure, but never for the full structure. The use of connectivity information allows data to be exchanged about nodes that are shared between two or more substructures. This method has been shown to be scalable and efficient[5].

ENSAERO-FE is dependent on an external code

*Research Scientist, MCAT Inc., AIAA Member, eldred@nas.nasa.gov, (650) 604-2402
†Senior Research Scientist, MCAT Inc., AIAA Member
‡Senior Research Scientist, AIAA Associate Fellow

to generate the substructure stiffness matrices. As this is a one time operation for linear structures, there is little to be gained by parallelizing it. And, there is no need to go to the effort and expense of duplicating standard codes that are readily available. In this case, NASTRAN/COSMIC[4] was used as a front end, although any similar code should be easily adaptable. This also allows access to the full range of standard preprocessing and CAD tools designed for NASTRAN, as well as use of the supply of existing data decks.

Once the user has used the front end program to create the substructure stiffness matrices, he builds input files describing the substructure connectivity and boundary conditions. Depending on the case being run, he may also set up load input files, or have the loads calculated by an attached aerodynamics code. In this case, ENSAERO-MPI[6] is used.

ENSAERO-MPI is an aeroelastic analysis package which couples the Reynolds averaged thin- layer Navier-Stokes equations with structural analysis. Its existing, limited, low fidelity simple finite element or modal structural capabilities were replaced by the new finite element code. ENSAERO's internal interpolation/extrapolation capability was adapted to exchange the aircraft aeroelastic data between the two disciplines.

The codes for the two disciplines are run in parallel on the IBM SP-2. ENSAERO uses one processor per aerodynamic zone. ENSAERO-FE similarly uses one processor per substructure. The number of processors used varies substantially from problem to problem, depending on problem size and desired performance. For this work, the aerodynamic code has typically used around ten processors and the structural code around five. The MPI library is used for communication between like codes as well as across disciplines. NASA Ames' MPIRUN library is used to manage processor allocation and some communications set-up chores.

Interpolation/extrapolation of loads and deflections is performed using an intermediary interface grid. This interface grid is made of the structural skin elements (internal structural elements such as spars are ignored). These skin elements are converted to triangular interface elements for ease of interpolation. A search algorithm locates fluid grid points that fall within each triangular interface element and computes the appropriate bilinear interpolation coefficients. Figure 2 illustrates the matching of the two domain grids. Very dense fluid grids near the wing tip and at wing mid span (near a control surface) result in a large number of fluid points per interface triangle in these regions.

Aeroelastic Validation

A wide variety of detailed aircraft configurations are being analyzed. The combination of ENSAERO with finite element structural analysis has proven to be an accurate and efficient tool.

The first configuration is a Boeing SST wing-body model. This is a relatively simple model with a rigid fuselage and a flat, trapezoidal wing. The aircraft was modeled using 8 fluid zones and 3 substructures, as shown in figure 3. Only the 4 fluid zones for the top half of the domain are shown; the bottom half is similarly modeled. Also, the structure is modeled in rectangular plates (QUAD4s), but plotted as triangular elements.

Figure 4 shows an aeroelastic solution. The right half of the figure colors the stuctural grid by substucture. The left half of the figure shows the pressure field on the aircraft surface. Both halves of the figure show the aeroelastically deflected aircraft shape. By careful comparison of the deflected structural and aerodynamic grids, the deflection/load interface code was validated.

To validate the new code in its entirity, this aeroelastic solution was compared to one using ENSAERO-MPI's older modal structural code. NASTRAN used the exact same structural model to compute the first 6 stuctural modes. The finite element code produced a leading edge tip deflections that was about 3.6% lower than the modal solution.

The second model being examined is a high-fidelity wing model, modeling the ARW-2 research wing [6]. The model includes the major internal details of the wing including ribs and spars. Final modeling is underway.

More detail on this work can be found in a related conference paper[8].

Conclusions

The interfaced ENSAERO-FE and ENSAERO-MPI codes are a powerful analysis tool. They're accurate, efficient, and allow the high-fidelity aeroelastic modeling of aircraft. The use of standard structural codes as a front end package provides efficient, easy user access to the codes.

The choice of NASTRAN as the code preprocessor provides a number of advantages including access to large libraries of tools, elements and data decks. Construction of a new custom parallel solver capability resulted in a custom, streamlined, parallel, high fidelity analysis capability.

Performance of the finite element code module is excellent for sufficiently large problems. For very small problems the communications overhead can reduce performance. Aeroelastic analysis of very flexi-

ble structures requires artificial damping to accelerate convergence.

Acknowledgments

This work was completed using the resources of the High Performance Computing and Communication Program (HPCCP) at NASA Ames Research Center. The work done by the first author was funded through NASA Ames Research Center Fluid Dynamics Analysis Contract NAS2-14109.

References

[1] Byun, C., and Guruswamy, G. P., "Wing-Body Aeroelasticity on Parallel Computers," *AIAA Journal*, Vol. 33, No. 2, Mar.-Apr. 1996, pp. 421-428.

[2] Guruswamy, G. P. "ENSAERO - A Multidisciplinary Program for Fluid/Structural Interaction Studies of Aerospace Vehicles," Computing Systems Engineering, Vol. 1, Nos. 2-4, 1990, pp.237-256.

[3] Byun, C., and Guruswamy, G. P., "Aeroelastic Computations on Wing-Body-Control Configurations on Parallel Computers," AIAA Paper 96-1389, April 1996.

[4] "NASTRAN User's Manual," NASA SP-222(08), June 1986.

[5] Carter, W.T., Sham, T.-L., Law, K. H., "A Parallel Finite Element Method and its Protoype Implementation on a Hypercube," *Computers & Structures*, Vol. 31, No. 6., pp. 921-934, 1989.

[6] Byun, C., and Guruswamy, G. P., "ENSAERO_MPI Parallel Multi-Zonal Version User's Guide," 1996.

[7] Sanford, M.C., Seidel, D.A., Erkstrom, C.V., Spain, C.V. "Geometrical and Structural Properties of an Aeroelastic Research Wing (ARW-2)," NASA T.M. 4110, April 1989.

[8] Eldred, L.B., Byun, C., Guruswamy, G. P., "Integration of High Fidelity Structural Analysis into Parallel Multidisciplinary Aircraft Analysis," presented at the 39th AIAA/ ASME / ASCE /AHS /ASC Structures, Structural Dynamics, and Materials Conference, Long Beach, CA, AIAA 98-2075, April 1998.

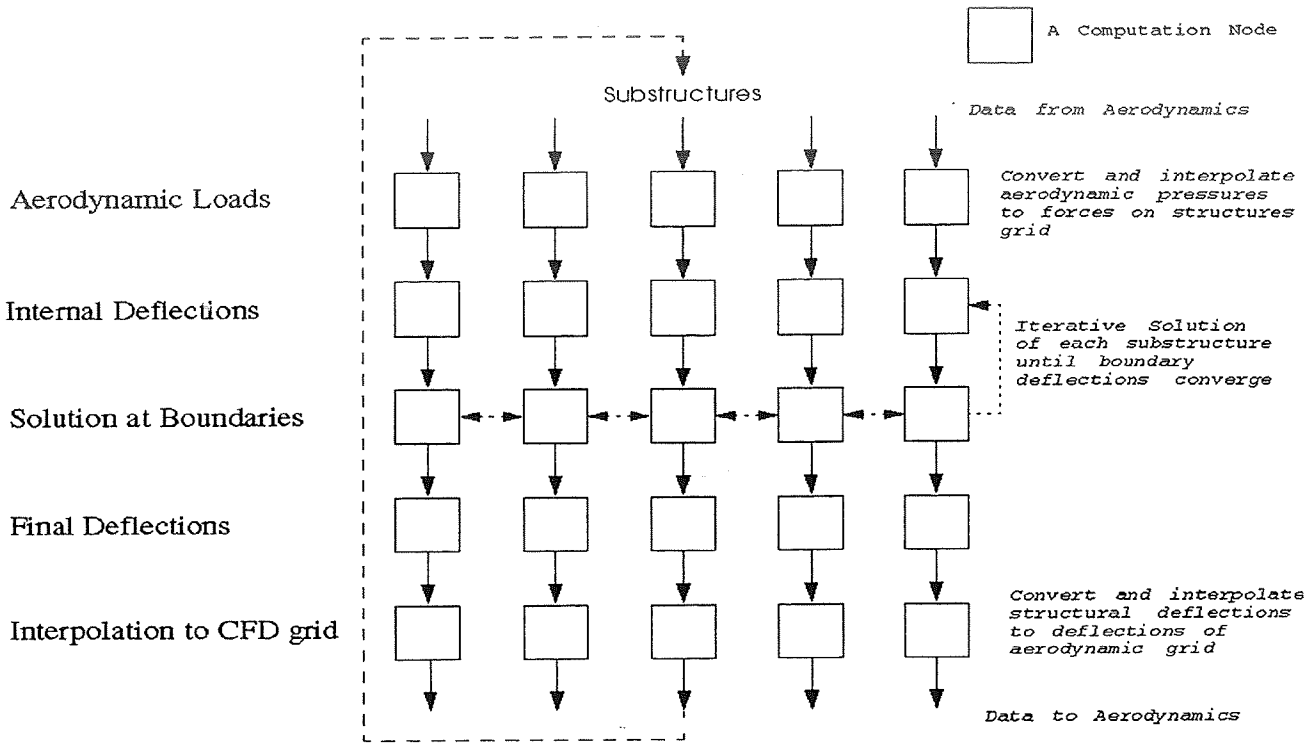


Fig. 1. Parallel Aeroelastic Structural Analysis using Substructuring

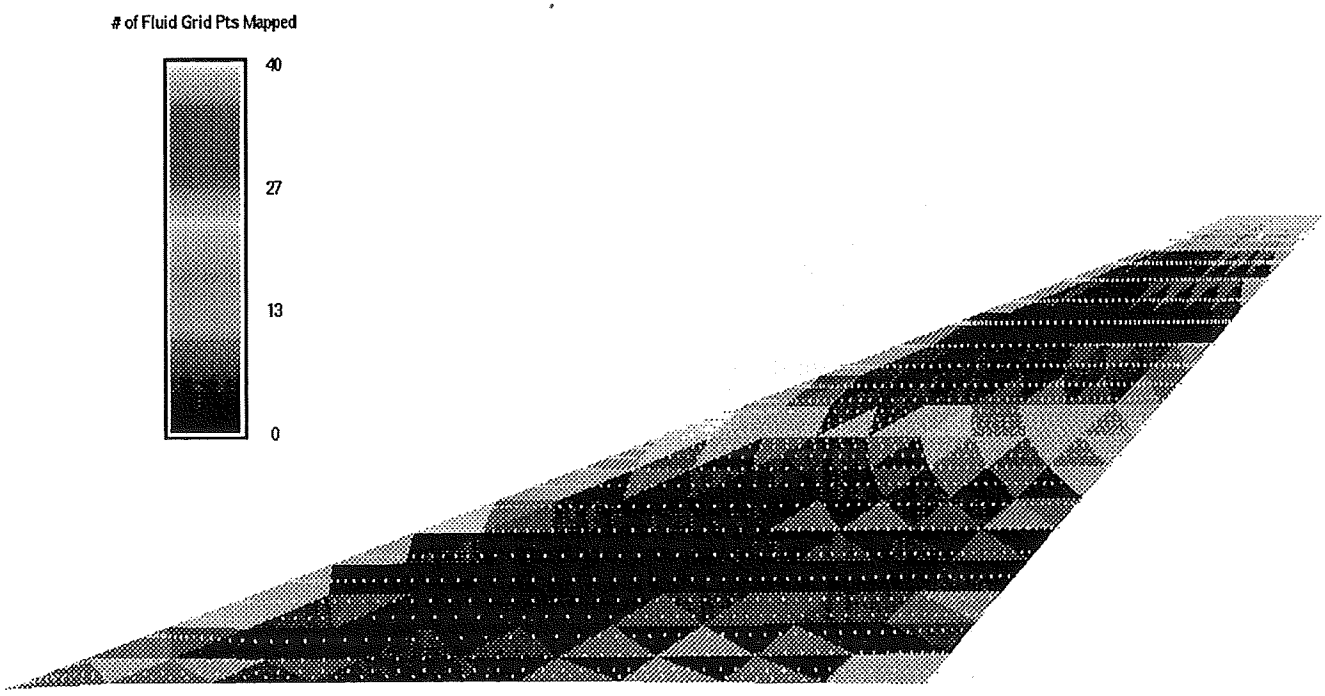


Fig. 2. Arrow Wing Structure-Fluid Grid Matching

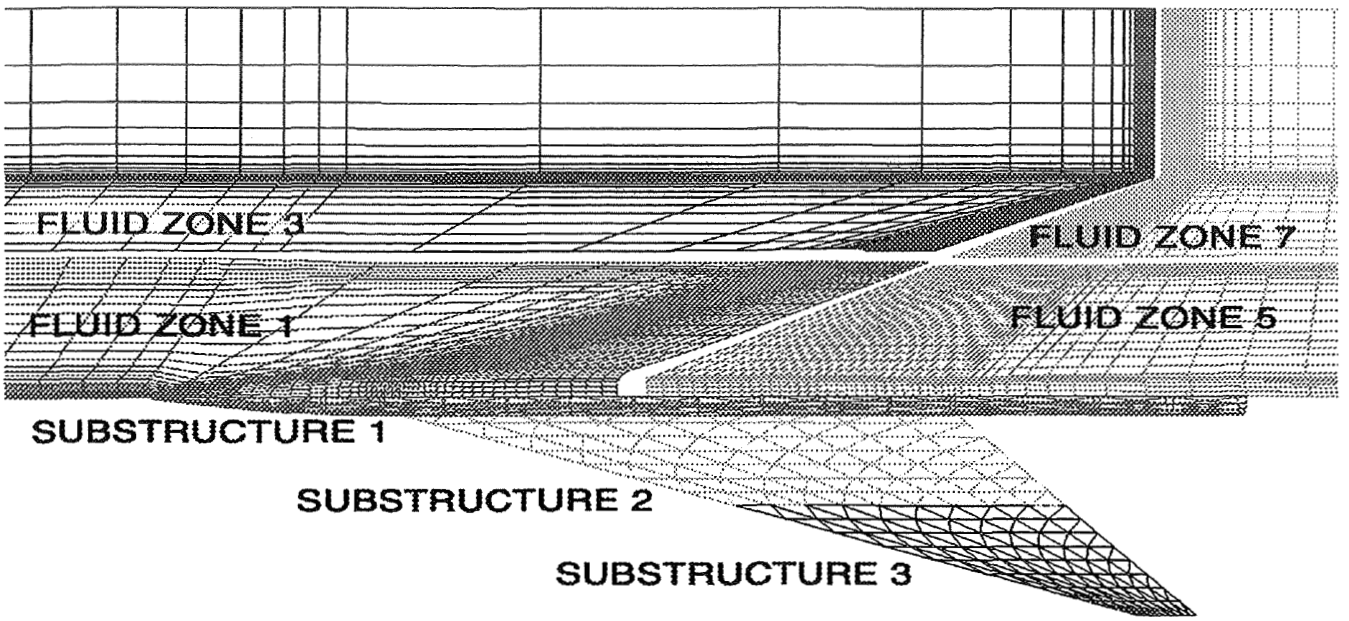


Fig. 3. Arrow Wing-Body Partial Fluid and Structural Grids

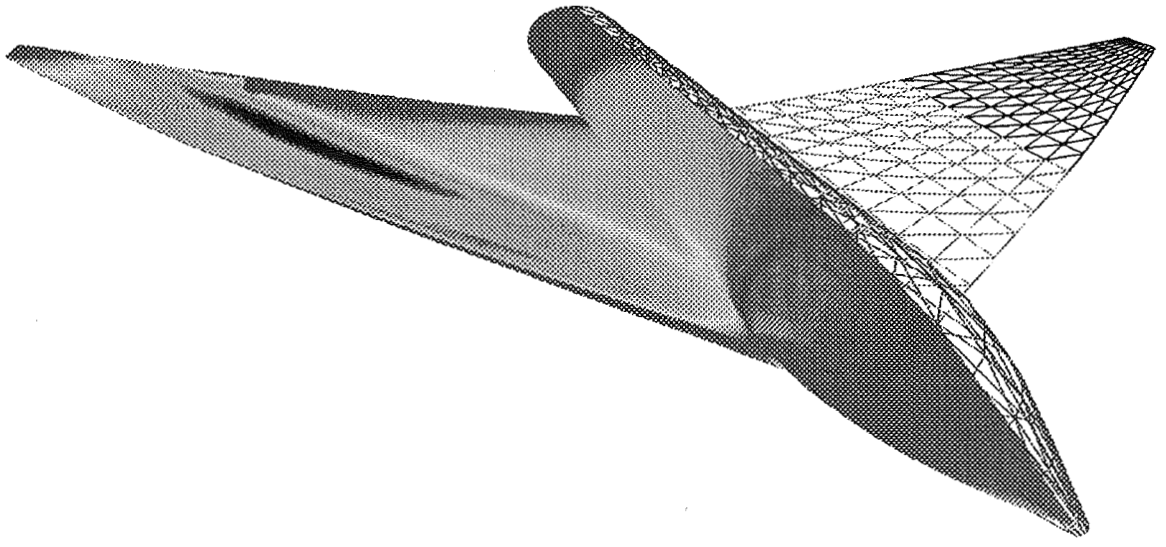


Fig. 4. Aeroelastic Solution for Arrow Wing-Body Configuration

PERFORMANCE AND APPLICATIONS OF ENSAERO-MPI ON SCALABLE COMPUTERS

Mehrdad Farhangnia, MCAT Inc, NASA Ames Research Center, MS 258-1, (650) 604-4496, farhangn@nas.nasa.gov
Guru Guruswamy, NASA Ames Research Center
Chansup Byun, Sun Microsystems Inc

Intoduction

The latest improvements and results generated by ENSAERO-MPI are presented in this talk. ENSAERO-MPI is a parallelized, high-fidelity, multi-block code with fluids, structures and controls capabilities developed at NASA Ames Research Center under the support of HPCC. It is capable of multidisciplinary simulations by simultaneously integrating the Navier-Stokes equations, the finite element structural equations as well as control dynamics equations using aeroelastically adaptive, patched grids. Improvements have been made to the code's robustness, moving grid capabilities and performance.

Multi-Zonal Parallel Implementation

The code is parallelized on a coarse grain level using the Message Passing Interface (MPI) for communication. The different disciplines are solved independently on separate nodes, with the flow domain partitioned further into a number of subdomains. There is also multi-parameter parallelization, where various parameter sets are run concurrently for a particular configuration. The partitioning and loading of the executables from each module onto the computational nodes is enabled using the MPIRUN library, which itself is based on MPI. This allows ENSAERO-MPI the portability to run on any shared or distributed memory, scalable computer supporting the MPI standard.

Adaptive Grids

Multidisciplinary simulation of an aircraft needs more advanced technology in moving grids because multi-zonal grids are common in complex geometries such as full aircraft configurations. For such grids used in ENSAERO-MPI, Trans-Finite Interpolation (TFI) based multi-block moving grids scheme is developed with non-matched block boundary interfaces. It provides a more general and flexible capability for aeroelastic applications of complex geometries. In addition, this moving grid capability is an efficient procedure for aeroelasticity simulations since the grid perturbation is done in parallel.

Applications

Several configurations have been analyzed with rigid and aeroelastic computations using ENSAERO-MPI on various parallel platforms. A wing-body, wing-body-empennage and a wing-body-nacelle topology of a high speed civil transport (HSCT) have been simulated and validated with various numerical and experimental results. Results of these computations can not be fully presented due to their proprietary nature, however, other configurations, such as the Boeing arrow-wing-body and the F-18/A full configuration will be presented. Figure 2 shows the CFD grid and pressure coefficient distribution of an 18 block, F-18/A geometry resulting from a high a Navier-Stokes calculation.

Scalability and Performance

A scalability and performance study of ENSAERO-MPI has been done on two different IBM SP2s, an SGI Origin 2000 and a Sun HPC 6000. A wing-body-empennage configuration, consisting of 1.25×10^6 grid points and a wing-body configuration with 5.1×10^5 grid points are used for evaluation. Results on the SGI have thus far been most encouraging with performance up to 100 MFLOPS/proc, including superlinear scalability. With the multi-parameter option, several cases can be computed concurrently resulting in potentially 25 GFLOPS performance on the 256 processor Origin 2000.

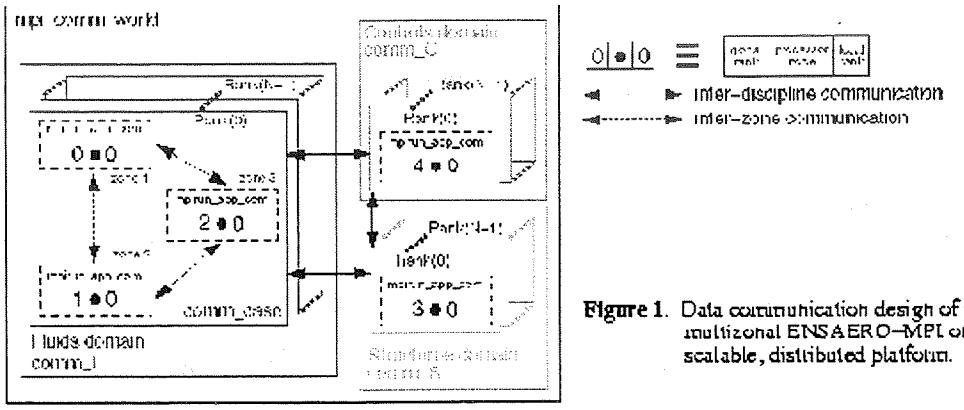


Figure 1. Data communication design of the multizonal ENSAERO–MPI on a scalable, distributed platform.

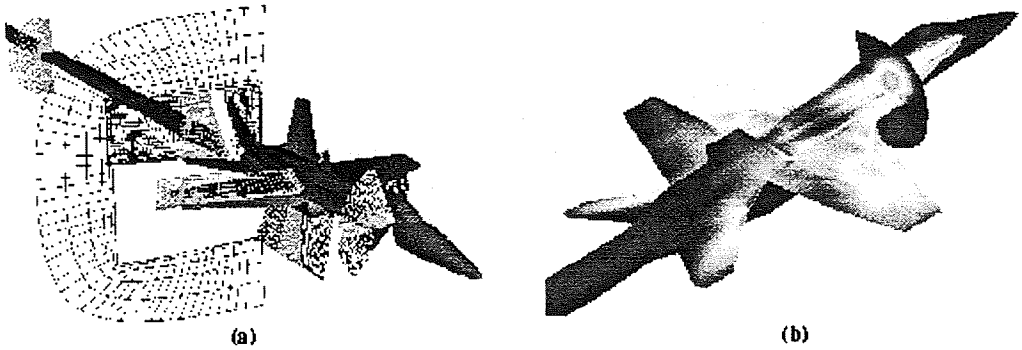


Figure 2. (a) An 18 block, 1.7 million grid point patched grid. (b) Pressure coefficient distribution.

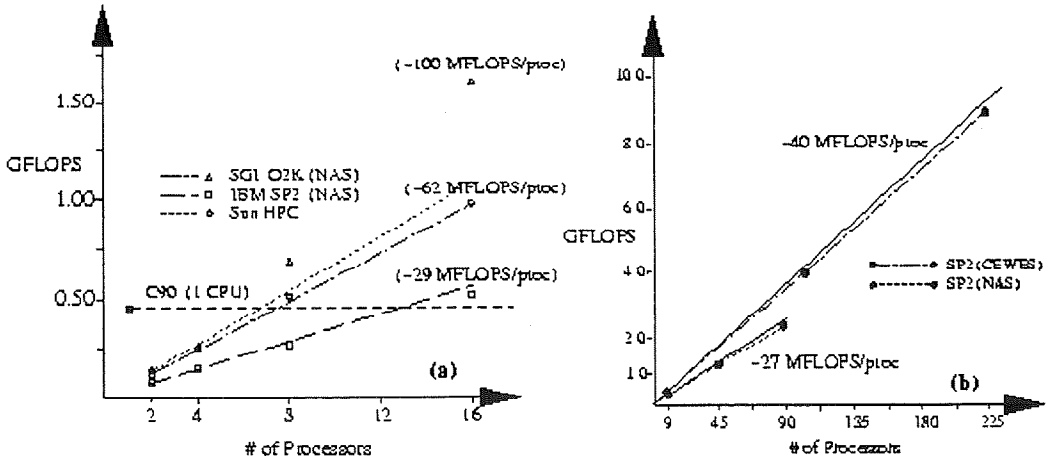


Figure 3. (a) Single–parameter performance of wing–body configuration, (b) multiple–parameter performance on several scalable platforms

530-39
018265

OVERAERO-MPI: PARALLEL OVERSET AEROELASTICITY CODE

Ken Gee
MCAT, Inc., MS 258-1, Ames Research Center, Moffett Field, CA 94035
(650) 604-4491, gee@nas.nasa.gov

Yehia M. Rizk
NASA/Ames, MS 258-1, Ames Research Center, Moffett Field, CA 94035
(650) 604-4466, yrizk@mail.arc.nasa.gov

366887
6 p.

1.0 Objective

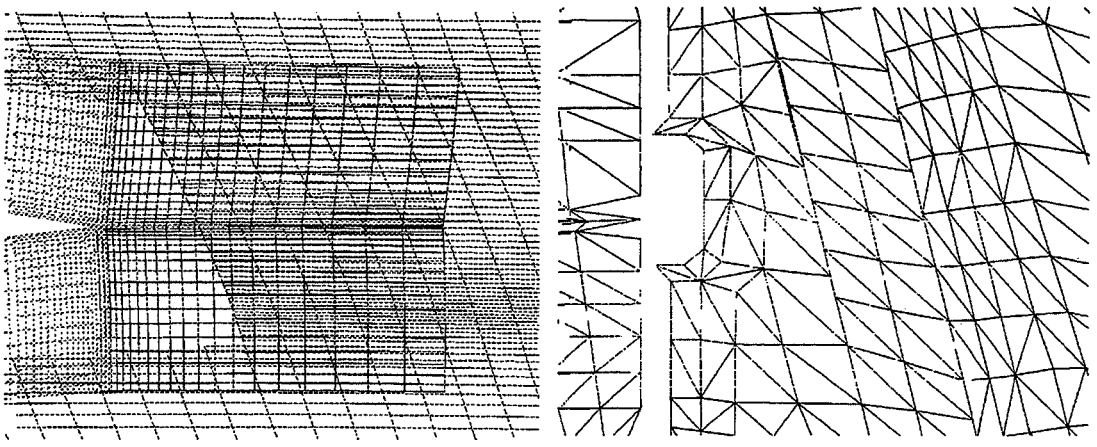
An overset modal structures analysis code was integrated with a parallel overset Navier-Stokes flow solver to obtain a code capable of static aeroelastic computations. The new code was used to compute the static aeroelastic deformation of an arrow-wing-body geometry and a complex, full aircraft configuration. For the simple geometry, the results were similar to the results obtained with the ENSAERO code and the PVM version of OVERAERO. The full potential of this code suite was illustrated in the complex, full aircraft computations.

2.0 Technology Impact

The flexibility of modern aircraft has an impact on the aerodynamic performance and design of the aircraft and can be predicted using computational fluid dynamics. The code suite presented in this paper provides a means for solving static aeroelastic problems for complex aircraft configurations modeled by overset grids on parallel computing systems. Overset grids simplify the modeling of complex aircraft geometries, which may include nacelles, pylons, and high-lift devices. Flows of interest are typically viscous and separated, requiring solution of the Navier-Stokes equations to accurately model the flow features. Such computations are resource- and time-consuming, requiring the use of vector supercomputers or massively parallel systems. The code suite provides an additional tool for use in the aircraft design process and can help improve the final aircraft design.

3.0 Technical Approach

The code suite consisted of the OVERFLOW-MPI code[1] and the modal structural analysis code that was part of ENSAERO.[2] The modal code was adapted for use with overset grids. In an overset grid system, grids of varying densities and topologies overlap, as shown in Fig. 1a. In addition, some grids can contain blanked regions. The structural grid has its own density and topology and may be unstructured, as shown in Fig. 1b. A binary tree search algorithm[3] was used to efficiently find neighbors between the two grid systems in order to interpolate surface pressure onto the structural grids and the surface deformation onto the fluids grids. Both fluids and structural surface grids were triangulated by the structural analysis code to simplify the search procedure and the bookkeeping of the donor data. The equation of the plane, defined by the three points of the donor grid triangle, was used to compute the interpolated data.



a) Overlapped, blanked fluids grids **b) Structural grid over same area**

FIGURE 1. Examples of grid overlap in overset fluids grid and between fluids and structures grids.

Interpolation of the deformation data onto the fluids grid surfaces was facilitated by treating each fluids grid surface as one of four types. Fluids grid surfaces were classified as flexible, referenced to the structural grid(s); flexible, referenced to a fluids grid(s); rigid, referenced to a fluids grid; and, rigid, non-moving. Deformation or translation data were obtained from the reference grid. This was sufficient to account for all fluids grid surfaces in the full aircraft case. The classification procedure was performed by the user.

The fluids volume grids were updated using the deformed fluids grid surfaces. The outer boundaries of most volume grids were allowed to be free-floating. This reduced the likelihood of crossed grid lines caused by the deformation. However, it introduced a slight error in the Chimera boundary interpolation data. This error was minimized if the surface deformation was small and if the grid outer boundary was sufficiently far from the surface such that the flowfield was close to freestream. The outer boundaries were fixed for volume grids with two or more surfaces on opposite faces, i.e., a surface at $l = 1$ and $l = l_{max}$. A linear decay function based on the arclength of the grid line was used to keep the opposing outer boundary fixed.

Loosely- and tightly-coupled versions of the code were developed. In the loosely-coupled code, a fluids solution was computed using the original fluids grids. The flow solution was then used to compute the deformation on the structural grids. The fluids volume grids were updated and the fluids solution continued using the updated fluids grids. The number of fluids iterations between structural code updates could be varied. In the tightly-coupled version, a structural update was computed after every fluids iteration. This could be used for time-accurate results at a later date. Tightly-coupled versions of the code were developed for both the OVERFLOW_PVM and OVERFLOW_MPI parallel overset flow codes.

The codes were used to compute the static aeroelastic deformation on an arrow-wing-body test case and a full aircraft case. The arrow-wing-body case was used to compare the accuracy and performance of the overset aeroelastic code with an existing patched aeroelastic code. The full

aircraft case demonstrated the ability of the overset aeroelastic code to handle large, complex grid systems in an efficient and consistent manner.

3.1 Arrow-Wing-Body Test Case

The arrow-wing-body test case, computed by Byun and Guruswamy[4] using the ENSAERO patched-grid aeroelastic code, was computed using the different versions of the parallel overset aeroelastic code. The arrow-wing-body geometry was a generic HSCT configuration, as shown in Fig. 2. It consisted of two grids totaling 510,400 grid points. The original patched grids were modified by overlapping the patched surfaces between the two grids to work with the overset codes. Fig. 3 shows a comparison of the grid densities of the wing portion of the fluids grid (Fig. 3a) and the structures grid (Fig. 3b). Note that the fluids grid required a much greater grid density to resolve the flow features, whereas the structural problem required a much coarser grid.

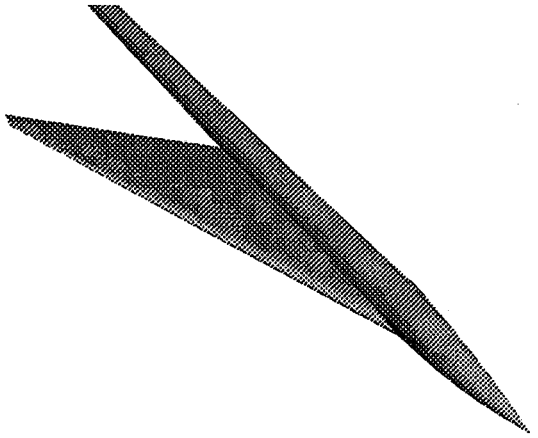
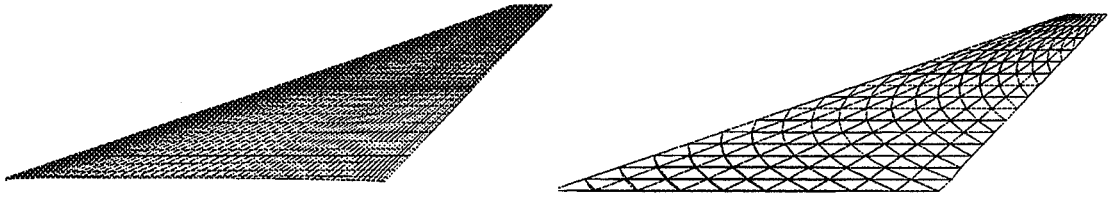


FIGURE 2. Arrow-wing-body surface geometry.



a) Arrow-wing-body wing grid portion (fluids) **b) Arrow-wing-body wing grid portion (structures)**

FIGURE 3. Comparison of arrow-wing-body grids.

The OVERAERO-MPI codes produced results similar to those obtained using the loosely- and tightly-coupled versions of OVERAERO-PVM, and the ENSAERO codes, as shown in Fig. 4. Differences in the convergence rate between the OVERAERO-MPI, OVERAERO-PVM, and

ENSAERO solutions were due to the different values of physical *dt* used in the structural code of each solution. For the OVERAERO-MPI solution, the physical *dt* was obtained using the mean aerodynamic chord (MAC) and the non-dimensionalization used in the OVERFLOW code. The OVERAERO-PVM solution used a smaller value for the reference length, leading to a smaller physical *dt* in the structures calculation. The value obtained from the ENSAERO code was based on the time scale of the structural response.

Differences in the converged value of the tip deflection were due to the different methods used to compute the wing area. The method used to compute the surface area in OVERAERO-MPI was generalized so that it could be applied to other configurations without modification. The method used in OVERAERO-PVM was similar to the method used in ENSAERO, and was specific to the arrow-wing-body grid topology. This resulted in the better match between these two solutions. The differences in the computed wing surface area led to differences in the normal force distribution over the wing surface, resulting in the difference in the final computed tip deflection.

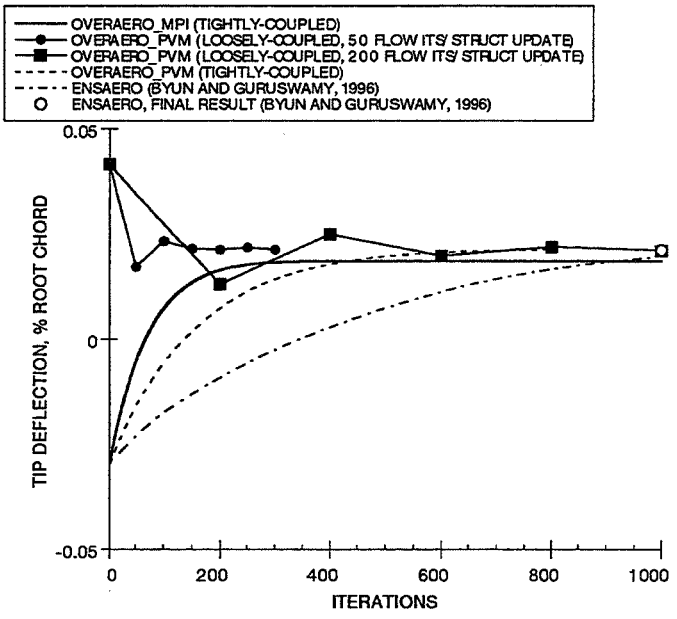
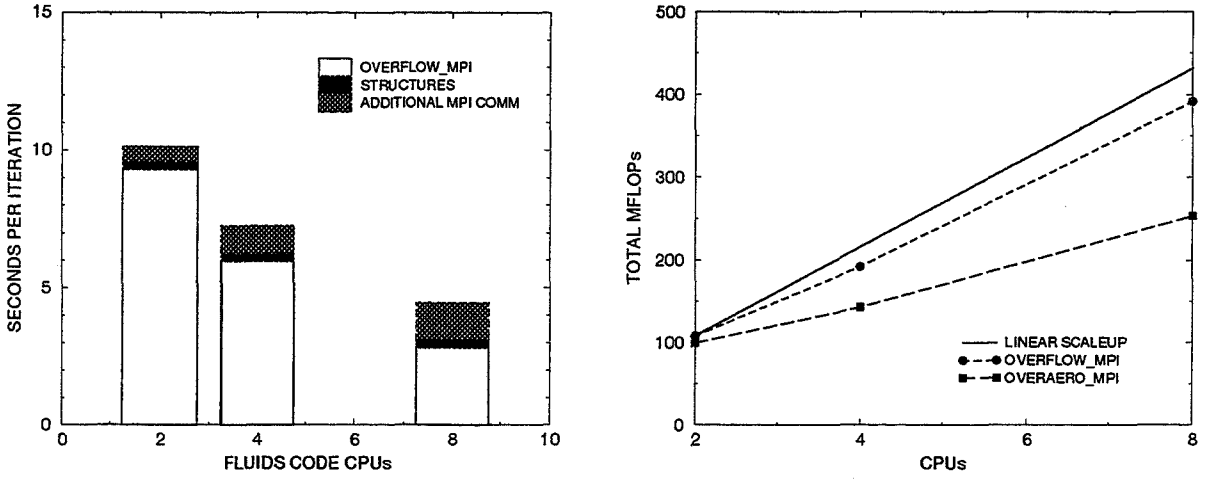


FIGURE 4. Comparison of OVERAERO and ENSAERO results for arrow-wing-body leading-edge tip deflection.

The performance of the OVERFLOW-MPI and tightly-coupled OVERAERO-MPI codes computing the arrow-wing-body problem on the Origin 2000 are shown in Figure 5. Figure 5a shows the walltime required for the baseline OVERFLOW_MPI code, the additional compute time required by the structures code, and the additional time required to send data between the fluids and structure node via MPI. The sum total is the time required by the tightly-coupled OVERAERO_MPI code. Since the structures portion ran on a single node regardless of the fluids code partitioning, the compute time required by the structures code was fairly constant. The communication time increased with the number of CPUs used due to the additional messaging required to first collect grid and flow data from the fluids worker nodes, and then distribute updated grid data to the fluids worker nodes. This caused a performance penalty as shown in Fig. 5b. The baseline OVERFLOW_MPI code scaled fairly well compared to the ideal linear increase.

However, the constant structures code compute time and additional MPI messaging time required cause the tightly-coupled OVERAERO_MPI code to scale with a much flatter slope, although there was an increase in performance with an increase in the number of CPUs used.



a) Walltime required per iteration **b) MFLOPs obtained**

FIGURE 5. Performance data for arrow-wing-body case.

3.2 Full Aircraft Configuration

The loosely-coupled version of OVERAERO_MPI was applied to a full aircraft configuration. The full aircraft grid system consisted of 150 grids totalling 32.3 million grid points. The grid system modeled the fuselage, wing, pylons, nacelles, and deployed high-lift system.

A rigid solution was obtained on the SGI Origin 2000 using OVERFLOW_MPI. A rigid flow solution was computed using 64 CPUs and required approximately 80 seconds per iteration. The solution required two subiterations per timestep to converge. This was due to the sensitivity of the solution to the order in which the Chimera boundary data was updated in the flow code. In the serial version of OVERFLOW, the Chimera boundary data was updated after each grid completed a time step. Thus, subsequent grids used a mix of n and $n+1$ level Chimera boundary data. However, in the parallel version of the code, the Chimera boundary data was updated at the end of the time step, which meant that all grids used n level Chimera boundary data. This change caused some grids to diverge unless two subiterations were computed during each time step for those particular zones.

A static aeroelastic solution was computed using the loosely-coupled OVERFLOW_MPI code. A structural update was computed after 100 flow iterations. A converged static aeroelastic solution was obtained after 8 structural updates, based on the tip deflection history. Loosely-coupled static aeroelastic solutions were also computed using the OVERFLOW_MLP code developed by James Taft.[5] The OVERFLOW_MLP code allowed for updates of the Chimera interpolation data after each grid time step, as was done in the serial version of OVERFLOW, or after the completion of the entire time step, as was done in OVERFLOW_MPI. The static aeroelastic solutions obtained using both methods agreed with the results obtained from the OVERFLOW_MPI computation.

Sample performance data for the various flow codes are listed in Table 1. For the OVERFLOW_MPI solutions, two subiterations were used only in those zones requiring it for convergence. Also, the partitioning scheme used to distribute the grids over the number of CPUs varied in each case, which explained why the performance does not scale with the number of CPUs for these two cases. For the OVERFLOW_MLP solutions, the serial update option required one process with 32 CPUs. It was found that 8 processes with 8 CPUs each provided the best performance in the parallel update option solution.

TABLE 1. OVERFLOW_MPI Performance on Full Aircraft Problem.

CODE	Number of CPUs	seconds/iteration	total MFLOPs
OVERFLOW_MPI (iter=2, some zones)	35	176	994
OVERFLOW_MPI (iter=2, some zones)	64	81	1183
OVERFLOW_MLP (serial update)	32 (1 x 32)	90	n/a
OVERFLOW_MLP (parallel update, iter=2, all zones)	64 (8 x 8)	87	n/a

4.0 Status of Technology

The overset aeroelastic codes presented in this abstract have been demonstrated to produce results similar to an existing patched aeroelastic code on a simple arrow-wing-body test case. The overset aeroelastic codes have also been applied to a large, complex aircraft configuration. The results indicated the code can successfully treat such a complex grid system.

Further testing is required to determine the accuracy of the method for the complex aircraft case. Further improvements to the code include automating the fluids surface grid classification process, improving the robustness of the interpolation scheme, and incorporating higher-fidelity structural methods into the code. Future plans include applying the code to other complex, full aircraft geometries.

5.0 References

1) Hatay, F. F., Jespersen, D. C., Guruswamy, G. P., Rizk, Y. M., Byun, C., and Gee, K., "A Multi-level Parallelization Concept for High-fidelity Multi-block Solvers," SC97: High Performance Networking and Computing, San Jose, CA., November, 1997.

2) Byun, C. and Guruswamy, G. P., "Wing-Body Aeroelasticity Using Finite-Difference Fluid/ Finite-Element Structural Equations on Parallel Computers," AIAA Paper 94-1487, April, 1994.

3) Aftosmis, M. J., "Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries," von Karman Institute for Fluid Dynamics Lecture Series 1997-02, March, 1997.

4) Byun, C. and Guruswamy, G. P., "Static Aeroelasticity Computations for Flexible Wing-Body-Control Configurations," AIAA Paper 96-4059-CP, August, 1996.

5) Taft, J., private communication, June, 1998.

Development and Validation of a Fast, Accurate and Cost-Effective Aeroservoelastic Method on Advanced Parallel Computing Systems

Sabine A. Goodwin & P. Raj
(770) 494-8587; (770) 494-3801
svermeersch@fs1.mar.lmco.com ; raj@mar.lmco.com

Lockheed Martin Aeronautical Systems
D/73-07, Z/0685
86 S. Cobb Drive
Marietta, GA 30063-0685

Progress to date towards the development and validation of a fast, accurate and cost-effective aeroelastic method for advanced parallel computing platforms such as the IBM SP2 and the SGI Origin 2000 is presented in this paper. The ENSAERO code, developed at the NASA-Ames Research Center has been selected for this effort. The code allows for the computation of aeroelastic responses by simultaneously integrating the Euler or Navier-Stokes equations and the modal structural equations of motion[1,2].

To assess the computational performance and accuracy of the ENSAERO code, this paper reports the results of the Navier-Stokes simulations of the transonic flow over a flexible aeroelastic wing body configuration. In addition, a forced harmonic oscillation analysis in the frequency domain and an analysis in the time domain are done on a wing undergoing a rigid pitch and plunge motion. Finally, to demonstrate the ENSAERO flutter-analysis capability, aeroelastic Euler and Navier-Stokes computations on an L-1011 wind tunnel model including pylon, nacelle and empennage are underway. All computational solutions are compared with experimental data to assess the level of accuracy of ENSAERO. As the computations described above are performed, a meticulous log of computational performance in terms of wall clock time, execution speed, memory and disk storage is kept. Code scalability is also demonstrated by studying the impact of varying the number of processors on computational performance on the IBM SP2 and the Origin 2000 systems.

ENSAERO Solver Description

ENSAERO is a multidisciplinary aeroelastic code which solves the Euler/Navier-Stokes equations coupled with structural equations. ENSAERO has time-accurate methods based on both central difference and upwind schemes[3]. Baldwin-Lomax and Johnson-King turbulence models are included for viscous flows. To represent the response of the structure due to external loads, the modal equations of motion are included in the analysis and a transfinite interpolation algorithm is applied to deform the grids for aeroelastic computations. The constituent parts of the applications in the parallel code communicate with each other using the Message Passing Interface[4] (MPI) protocol.

Model Descriptions and Results

Aeroelastic Research Wing (ARW-2)

As shown in Figure 1, this wing body configuration is composed of a supercritical airfoil section and has an aspect ratio of 10.3 and a leading edge sweep back angle of 28.8 degrees. The model was tested in the Transonic Dynamics Tunnel (TDT) at transonic Mach numbers and three different wind tunnel stagnation pressures[5].

The H-H type computational Navier-Stokes grid contains a total of 2.6 million nodes (196 in the streamwise direction, 112 in the spanwise direction and 60 normal to the surface). The surface and field Navier-Stokes grid for the ARW-2 configuration is illustrated in Figure 2. Grid points were clustered near the surface to insure a proper resolution of the boundary layer. The grid was split up into eight blocks of equal size (330,000 points each) to assure a load balanced computation.

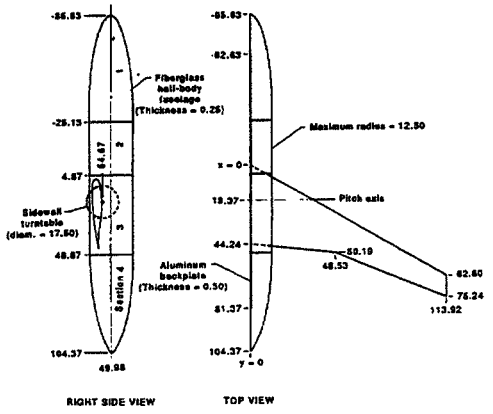


Figure 1 : ARW-2 geometry

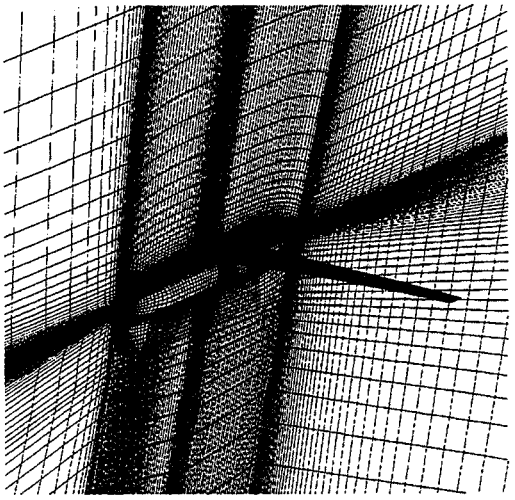


Figure 2 : ARW-2 Navier-Stokes grid

Aeroelastic computations at M=0.8, α=0 deg, Reynolds number = 4.2 x 10^6 and dynamic pressure (q) = 0.86 psi were performed. The first five structural modes were included in the analysis. The simulation was started from a rigid steady state solution followed by a static aeroelastic calculation with a large amount of structural damping. Because the structures computations require only a very small amount of CPU time compared to the fluids computations, structural deflections were calculated at every timestep. The interactions between fluids and structures modules were continued until no further change in the wing tip deflections was observed and therefore a static aeroelastic solution was obtained.

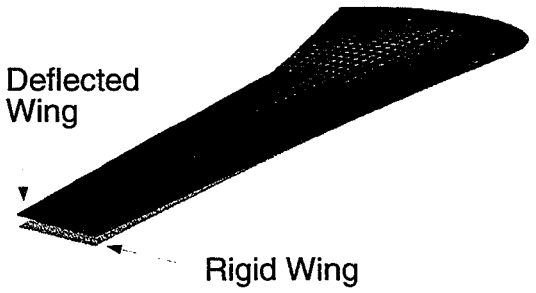


Figure 1 : Deflected and undeflected wing shape for the ARW-2

The time history of the generalized displacements for the first four modes shows that the first mode (first bending) is the dominating mode in the aeroelastic deflection. Figure 3 is an illustration of

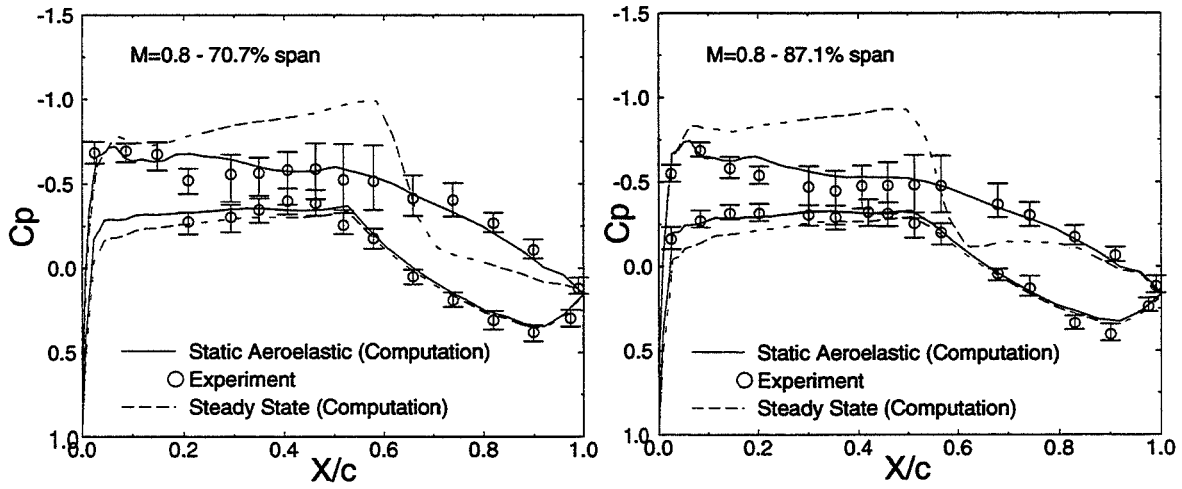


Figure 2 : Upper and lower surface pressures from steady state and aeroelastic solutions at 70 % and 87 % span for the ARW-2

the initial (undeflected) and final (deflected) wing shapes. Figure 4 shows the pressure coefficient (Cp) distribution at the 70.1% and the 87.1% span stations. Error bars indicate the Cp variation in the experiments due to unsteadiness in the flow. The steady state Cp differs significantly from the experimental data. One can clearly see that the rigid computations result in a shock on the upper surface of the wing while the experimental upper surface Cp's remain smooth. When the wing is allowed to deform due to the normal aerodynamic forces, surface pressures after the static deflections match the experimental data very well. Figure 5 shows the upper surface pressure for the steady state solution and the static aeroelastic solutions and it clearly indicates the significance of the aeroelastic effects for the ARW-2.

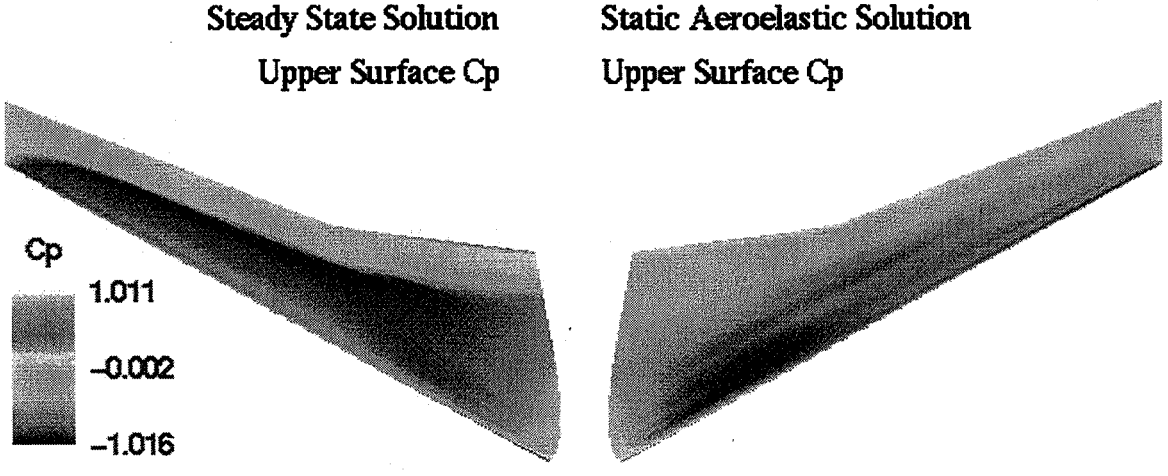


Figure 3 : Upper Cp from rigid (left) and aeroelastic (right) solution for the ARW-2

Table 1 : Performance results for the ARW-2 steady state solution at M=0.8 on IBM SP2 (2.5 million grid points)

No. Compute Nodes (on Sp2)	Wall Clock Time/Step (Seconds)	Total CPU Time/Step (Seconds)
4	60.0	240.0
8	30.9	247.2
16	15.9	254.4
32	8.7	278.4

The results of the ARW-2 test case using multiblock grids also provide a good illustration of the current performance capabilities of the ENSAERO code. To demonstrate scalability, i.e. computational speed-up with increasing number of processors, the grid was divided into four, eight, sixteen and thirty-two subgrids. The computational domain was decomposed by splitting the full grid such that each block had exactly the same number of grid points.

Table 1 shows the ENSAERO performance for the ARW-2 steady state solution using the IBM-SP2. Good actual scalability compared to theoretical scalability was obtained up to 16 processors. When using 32 processors, the communication overhead leads to a deterioration from linear speed-up.

Benchmark Models Supercritical Wing (BMSW)

This supercritical wing is the second in a series of three similar models that the Benchmark Models Program is testing in the TDT. The model is a rigid rectangular wing with a NASA SC(2)-0414 second generation supercritical airfoil. The chord of the BMSW is 16 inches and the span is 32 inches. The mount system in the tunnel is a flexible mount system called the Pitch and Plunge Apparatus (PAPA) and it provides a well defined, two-degree-of-freedom dynamic system on which rigid models encounter classical flutter in the TDT[6]. Figure 6 shows the planform and airfoil section of the model. An H-H grid with a total of 1.15 million grid points (128 points in the chordwise direction, 66 points in the spanwise direction and 65 points in the normal direction) was generated.

Computations are performed at M=0.77, α=0 deg, Reynolds number=1.055×10^6 and q=1.17 psi. Methods in the frequency and time domain are used to determine the flutter boundary for this wing. Several cycles of a forced harmonic oscillation (at different reduced frequencies) are calculated to obtain the generalized forces as a function of reduced frequency. Multiple time marching computations are required at various reduced frequencies to generate the GAF's for each structural mode (pitch/plunge). Computations for this geometry are underway at the time of writing.

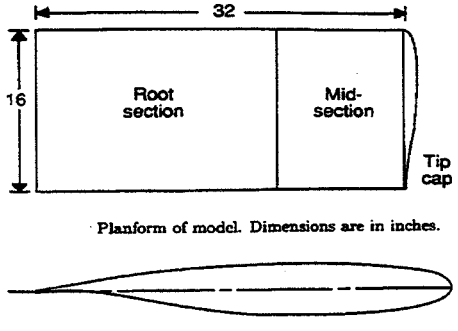


Figure 4 : Navier-Stokes grid for BMSW

Figure 7 shows the ENSAERO performance for the BMSW steady state and dynamic aeroelastic analysis. As for the ARW-2 testcase, good actual scalability was obtained up to 16 processors.

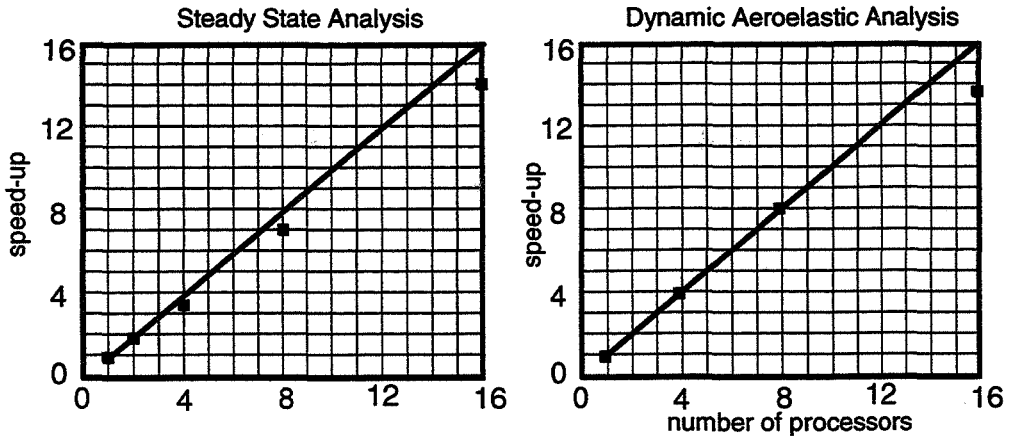


Figure 5 : Scalability results for BMSW on the IBM SP2

L-1011 Wind Tunnel Model

The L-1011 wind tunnel model is a 0.02 scale model of the Lockheed "Tristar" L1011-500 configuration. The L1011-500 is a wide-bodied, three engine, commercial aircraft which was developed and manufactured by the Lockheed-California Company.

An 83 block grid of a total of approximately 9 million grid points was generated. The grid topology is H-H around the pylon/nacelle, C-H around the wing and H-H in the farfield. Figure 8 shows the surface and field grid around the body and the pylon/nacelle. The spacing near the surface is such that the computations will result in a Y+ of order 1.

To demonstrate the ENSAERO flutter-analysis capability, aeroelastic Euler and Navier-Stokes computations are currently being performed at M=0.80, using the grid shown in Figure 8. The solutions and the flutter characteristics will be compared to experimental data available for these flow conditions. Figure 9 shows the steady state Euler solution which was obtained using 35 processors of an Origin 2000 system. The maximum time per step for these computations was 13.7 seconds and approximately 10,000 iterations were required to reach convergence. This computation required approximately 3.1 gigabytes of memory. The dynamic aeroelastic analysis for this case is underway at the time of writing.

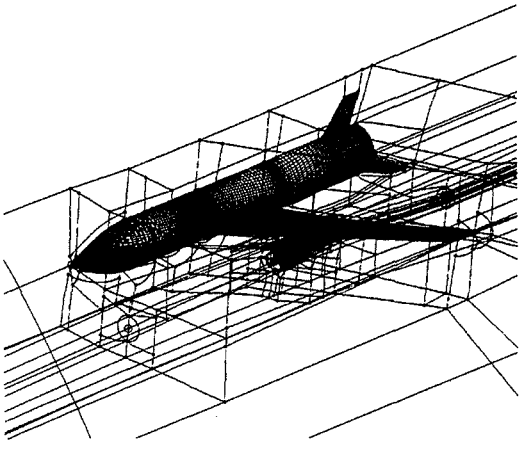


Figure 6 : Surface grid on L-1011 geometry

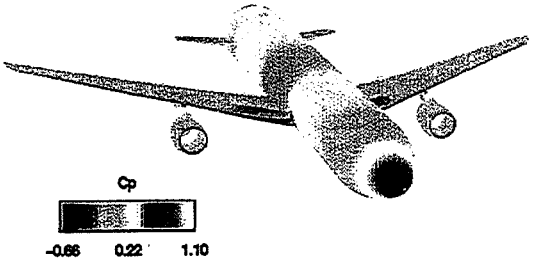


Figure 9 : Steady state solution on L1011 geometry at M=0.8

Concluding remarks

This work will help us obtain a more detailed understanding of the high performance computing environments so that they can be used cost-effectively to solve aerospace engineering problems. The resulting capability will significantly enhance the aerodynamic loads and flutter analysis methodologies crucial to achieving "optimal" structural design while meeting performance goals.

Acknowledgments

This research was performed under the NAS2-14092 contract sponsored by NASA-Ames Research Center with Dr. G.P. Guruswamy as the technical monitor. Thanks are due to Dr. Guruswamy of NASA Ames Research Center , Dr. C. Byun and Mr. M. Farhangnia for their support and councel. The use of the NASA-Ames IBM SP2 and SGI Origin 2000 system is gratefully acknowledged

References

1. Guruswamy, G.P. " User's Guide for ENSAERO - A Multidisciplinary Program for Fluid/Structural/Control Interaction Studies of Aircraft" NASA TM 108853, October 1994

2.Byun, C. and Guruswamy, G.P. "A Comparative Study of Serial and Parallel Aeroelastic Computations on Wings," NASA TM 108805, January 1994

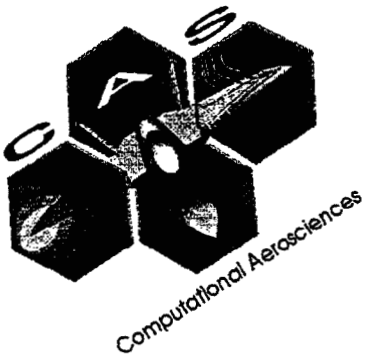
3. Obayashi, S., Guruswamy, G.P., and Goorjian, P.M., "Application of a Streamwise Upwind Algorithm for Unsteady Transonic Computations over Oscillating Wings, " AIAA Paper 90-3103, August 1990

4. Dongarra, J., Hempel, R., Hey, A., Walker, D. "A proposal for a User-Level, Message Passing Interface in a Distributed Memory Environment", ORNL/TM-12231, June1993

5. Eckstrom, C.V., Seidel, A.S. and Sandford, M.C., "Measurements of Unsteady Pressure and Structural Response for an Elastic Supercritical Wing" NASA TP 3443, November 1994

6. Dansberry, B.E., Durham, M.H., Bennett, R.M., Turnock, D.L., Silva, W.A., and Rivera, J.A., "Physical properties of the Benchmark Models Program Supercritical Wing." NASA TM 4457, September 1993.

OMIT THIS PAGE



Session 8:

Design/Engineering Environments

MOD TOOL (MICROWAVE OPTICS DESIGN TOOL)

Daniel S. Katz, Andrea Borgioli, Tom Cwik, Chuigang Fu, William A. Imbriale, Vahraz Jamnejad, and Paul L. Springer

Jet Propulsion Laboratory

(contact: Daniel S. Katz, 4800 Oak Grove Drive, MS 168-522, Pasadena, CA, 91104, voice: 818-354-7359, fax: 818-393-3134, e-mail: Daniel.S.Katz@jpl.nasa.gov)

I. Introduction

The Jet Propulsion Laboratory (JPL) is currently designing and building a number of instruments that operate in the microwave and millimeter-wave bands. These include MIRO (Microwave Instrument for the Rosetta Orbiter), MLS (Microwave Limb Sounder), and IMAS (Integrated Multispectral Atmospheric Sounder). These instruments must be designed and built to meet key design criteria (e.g., beamwidth, gain, pointing) obtained from the scientific goals for the instrument. These criteria are frequently functions of the operating environment (both thermal and mechanical). To design and build instruments which meet these criteria, it is essential to be able to model the instrument in its environments.

Currently, a number of modeling tools exist. Commonly used tools at JPL include: FEMAP (meshing), NASTRAN (structural modeling), TRASYS and SINDA (thermal modeling), MACOS/IMOS (optical modeling), and POPO (physical optics modeling). Each of these tools is used by an analyst, who models the instrument in one discipline. The analyst then provides the results of this modeling to another analyst, who continues the overall modeling in another discipline.

There is a large reengineering task in place at JPL to automate and speed-up the structural and thermal modeling disciplines, which does not include MOD Tool. The focus of MOD Tool (and of this paper) is in the fields unique to microwave and millimeter-wave instrument design. These include initial design and analysis of the instrument without thermal or structural loads, the automation of the transfer of this design to a high-end CAD tool, and the analysis of the structurally deformed instrument (due to structural and/or thermal loads).

MOD Tool is a distributed tool, with a database of design information residing on a server, physical optics analysis being performed on a variety of supercomputer platforms, and a graphical user interface (GUI) residing on the user's desktop computer. The MOD Tool client is being developed using Tcl/Tk, which allows the user to work on a choice of platforms (PC, Mac, or Unix) after downloading the Tcl/Tk binary, which is readily available on the web. The MOD Tool server is written using Expect, and it resides on a Sun workstation. Client/server communications are performed over a socket, where upon a connection from a client to the server, the server spawns a child which is be dedicated to communicating with that client. The

server communicates with other machines, such as supercomputers using expect with the username and password being provided by the user on the client.

II. User Interface

The initial MOD Tool screen requires the user to provide a username and password, and to select an old or new design on which to work. The client communicates with the server to check this data, and to reach the area of the database containing this design. At this point, the main MOD Tool screen comes up, allowing the user to work in one of six modes.

1. Design Mode

This mode allows the user to load, modify, and save a design. A design is primarily a set of physical optical elements (reflectors), but it also includes two non-physical elements, the location of a feed, and the location for outputs to be gathered. Reflectors are designed as conic sections cut with an arbitrarily-defined oval cutting cylinder. These conic sections may be either flat plates, paraboloids, ellipsoids, or hyperboloids. A feed location is defined by a point at the feed aperture, a direction in which the feed is pointed, and the major and minor radii of the aperture. An output system is defined by a point and a direction. The design information is shown in two forms: a table and a graphic window. The table presents coordinates of the points that determine the object (for a paraboloid, these would include the vertex, the focus, a point on the paraboloid and a point off the paraboloid to determine the axis of the cutting cylinder, and the major and minor radii of the cylinder) and other information that might be useful to the designer (again for a paraboloid, this includes the distance from the vertex to the focus). The graphics window shows this objects determined from this data in 2-D slices, and allows the user to move points on the screen. Changes in either form are reflected in the other (if a graphic point is moved, the table is updated to reflect the new data, and vice versa). Our current plans for the graphics frame in the design mode are to allow the user to work on any Cartesian plane (for example, the x-y plane at $z=1.4$ mm). The designs are examined and modified by the GUI client, but the data is loaded from and saved to the server's database.

2. Prescription Mode

This mode allows the user to specify and modify the objects that will be used in a given analysis. The user can choose objects and order a subset of the objects from the current design, or load an old prescription. Objects may be modified in location and/or orientation in order to perform a tolerancing analysis. Once the objects are chosen and possible modified, the user can save the prescription. The prescriptions are examined and modified by the GUI client, but the data is loaded from and saved to the server's database. See Figure 1 for an example of this mode.

3. Geometric Optics Analysis Mode

This mode allows the user to perform geometric optics calculations on the prescription and design that have been previously specified. This is done by converting the design/prescription

information into a MACOS input file, and running MACOS under control of the GUI. The conversion is performed by the server, which also runs MACOS. Results are transferred to and displayed by the client. This analysis is quite fast, and may be used for initial design work.

4. Physical Optics Analysis Mode

Physical Optics (PO) is a much more precise analysis method, which is used when the objects being analyzed are relatively small (size is in terms of wavelengths). This mode allows the user to set up and launch PO runs, which generally are fairly time consuming and are done on a supercomputer. Currently, the PO code used at JPL has been ported to the Cray J90, and HP SPP-2000, and Beowulf systems. Starting a PO job on a supercomputer involves the server converting the design and prescription information to a format the PO code understands, the client obtaining a username and password for the supercomputer from the user, and the server transferring the files to the supercomputer and starting the job. Additionally, the PO code can be used to analyze surfaces that have been structurally or thermally deformed by the server using mesh and load file which have been submitted (using the following two modes), calculating the coefficients of a bipolynomial surface for the deformation of each surface, and transferring this information to the supercomputer. In order to do this, the server strips the information about each surface from the mesh and load files, and uses MATLAB to calculate the desired coefficients.

5. Submitting a Mesh Mode

This mode is used by the structural engineer. Once he has created a mesh, either based directly on the design, or based on a CAD model which was built from the design, he uses this mode to submit it from the client to the server, and store it in the database associated with the design. MOD Tool currently uses FEMAP neutral files for meshes, which are unit-independent. Therefore, the structural engineer must also provide information about the units that were used in the mesh.

6. Submitting a Load Mode

This mode is also used by the structural engineer. Once a mesh has been created, it is normally deformed by either structural or thermal loading. MOD Tool currently accepts NASTRAN .f06 files that contain the deformation of each node of the mesh. This information is used by the PO code, as described in the Physical Optics Analysis mode.

III. Conclusions and Future work

MOD Tool is being built in conjunction with MIRO. The motivation for this is both to help the MIRO project by providing the analyses required in a timely manner as well as to help MOD Tool by providing a real instrument as a test case. The figures in this extended abstract all come from MIRO. The close involvement of the two projects has been very valuable, and while the measure of success for MIRO will not be seen for many years until it is flying and sending back

data, the measures of success for MOD Tool will be seen sooner, as MOD Tool itself is (hopefully) used and supported by more flight projects, and as the pieces that make up MOD Tool are used by other software projects.

Future needs that MOD Tool is designed to be able to meet include using analysis of a metrology input (the instrument as-built), and design optimization. As MOD Tool develops, these capabilities will be added, as will others that are have not yet been identified. MOD Tool has been designed as a framework to which many things can be added. It already includes the use of Tcl/Tk code on multiple platforms, Perl scripts, Fortran programs, and MATLAB code on a workstation, as well as Expect to allow use of networked supercomputers with proper accounting. One of the considerations in this work was developing as little as possible, and thus, reusing previously developed tools and components as much as possible. Figure 2 shows the overall MOD Tool process, which clearly contains many tools that can be used independently.

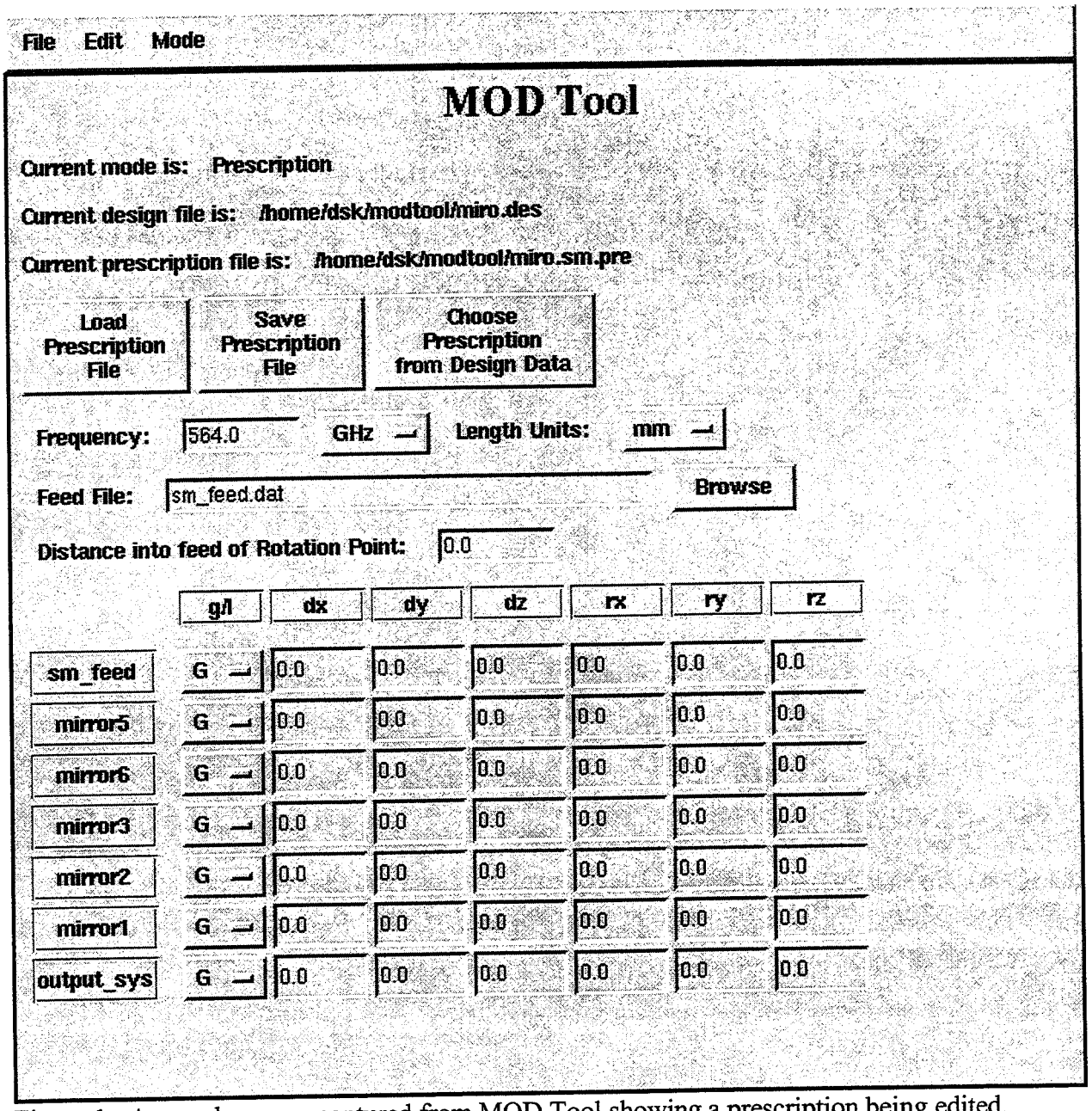


Figure 1. A sample screen captured from MOD Tool showing a prescription being edited.

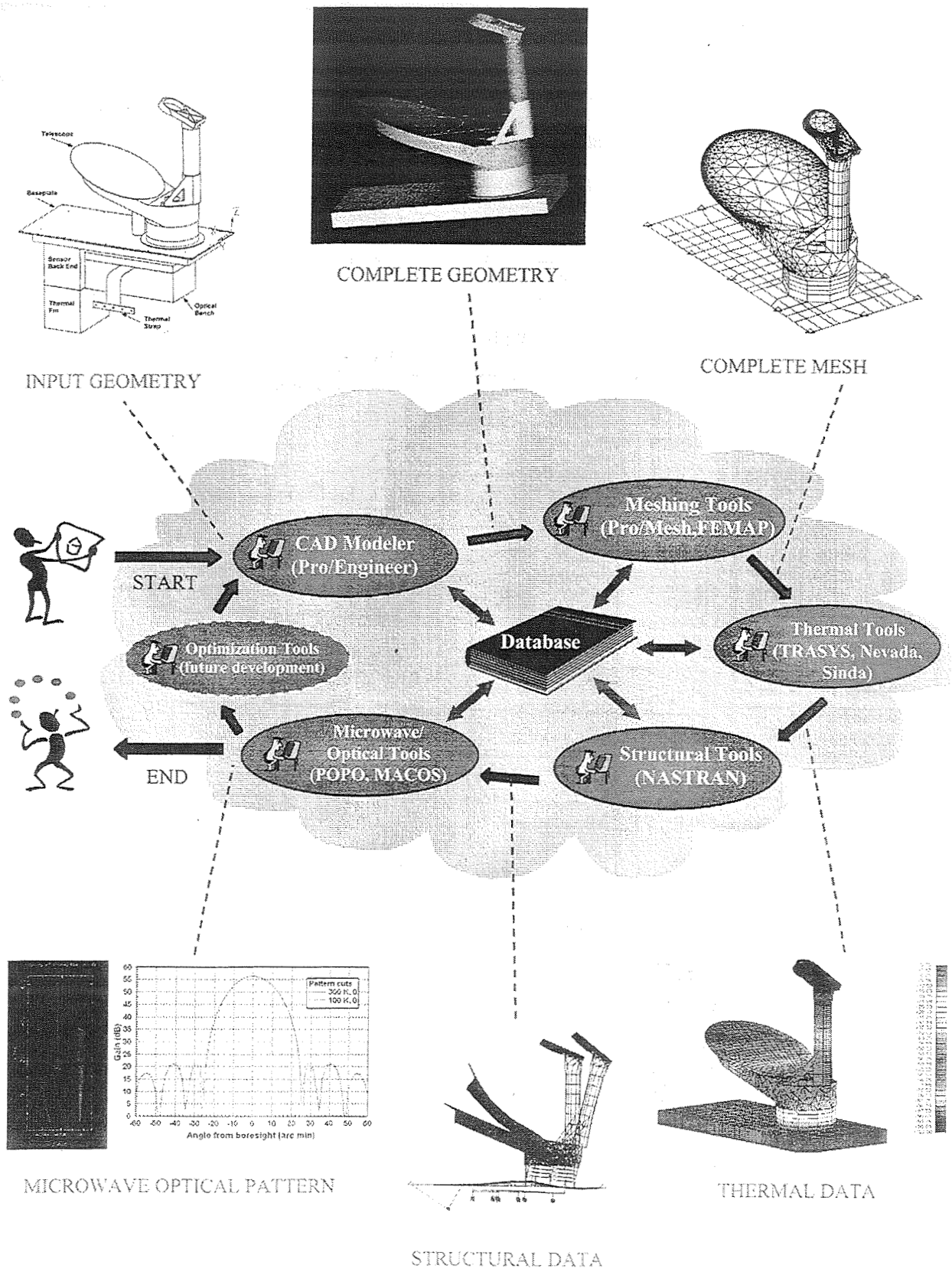


Figure 2. Conceptual process to be performed by MOD Tool

AN OBJECT ORIENTED FRAMEWORK FOR HSCT DESIGN

Raj Sistla, Augustine R. Dovi & Philip Su
Computer Sciences Corporation
3217 N. Armistead Avenue, Hampton, VA 23666
r.sistla@larc.nasa.gov (757) 766-8233

Introduction

Aircraft design is inherently iterative in nature and multidisciplinary in composition. The process is complicated by the fact that the focus and approach of each discipline can be quite distinct, and multiple invocations of the discipline programs are required to arrive at a feasible design. The usual result is a design procedure that is largely inflexible and computationally taxing. An earlier effort within the Framework for Interdisciplinary Design and Optimization (FIDO) project used Parallel Virtual Machine (PVM) to handle communications between discipline codes executing in a "host/slave" mode. This framework was sensitive to the host operating system and changing the analytical connectivity or switching discipline codes required major programming intervention.

The goal of the current framework is to provide a programming environment for automating the distribution of a complex computing task over a networked, heterogeneous system of computers. These computers may include: engineering workstations, vector supercomputers, and parallel-processing computers. They work on their individual parts of the design, in parallel whenever possible, and have access to centralized data. Each computational task is assigned to the most appropriate computer type. The present framework provides a means for automating the overall design process. It provides communication and control between components, which include the diverse discipline computations in a design problem and the system services facilitating the design.

The Framework

A multidisciplinary analysis and optimization system has been developed that is capable of concurrent analyses using several disciplines such as aerodynamics, structures, performance, propulsion and optimization. This system uses Common Object Request Broker Architecture (CORBA), a *client - server* paradigm, in an Object Framework [1] for the integrated design of a High Speed Civil Transport (HSCT) aircraft across a networked system of heterogeneous computers. The Beans Development Kit (BDK) is used to provide a JavaBeans-based graphical interface for user input, interactive and visual object connectivity, and for monitoring the progress of problem execution. Java's Database Connectivity, JDBC, is used by the client and the server objects to communicate with a central database. The primary objective of the design framework is to optimize the aircraft weight for given cruise conditions, range and payload requirements, subject to aerodynamic, structural, and performance constraints. The design variables include both structural thicknesses and geometric parameters defining aircraft shape. The framework provides the capability to switch between low-, medium-, and high fidelity codes with ease.

Common Object Request Broker Architecture (CORBA):

The Common Object Request Broker Architecture is a specification adopted by a consortium of industry representatives known as the Object Management Group (OMG) to define a framework for developing object-oriented distributed applications. In this model, an object is an encapsulated entity with a unique identity whose services can be accessed only through a well defined *interface*. Clients issue requests to objects to perform services on their behalf. The object implementation and location are transparent to the requesting client.

CORBA can be thought of as a "software bus" [Fig. 1] connecting various objects, both application and service, on a network of computers. Objects on the bus can be used by any other object on the bus, with the Object Request Broker (ORB) mediating the transfer of messages between them.

Object Request Broker (ORB):

The ORB is a software implementation of the CORBA specification [Fig. 2]. The key feature of the ORB is the transparency of how it facilitates the client-object communications. The client is not required to know where the target object resides, how and in what programming language it was implemented, or the operating system of the target host. When a client makes a request, it is not concerned whether that object is currently activated and ready to accept requests. The ORB transparently starts the object, if required, before delivering the request. The client does not need to know what underlying communication mechanism the ORB uses to mediate the message passing between the client and the server. All these enable the user to generate 'thin clients' i.e., all the number crunching is done on the server-side on computers most appropriate for the task. It also frees the application developer to concentrate more on the application domain issues and less about the low-level distributed system programming issues.

An ORB is one component of the OMG's Object Management Architecture (OMA). The others include the application objects, CORBA services, and CORBA facilities. Services include:

- Naming Service - which allows clients to find objects based on names,
- Trading Service - which allows clients to find objects based on their properties.

Different commercial implementations of the ORB must all be able to talk to each other using a standard network protocol called the Internet Inter-ORB Protocol (IIOP).

Within an object framework, each component communicates with others on a peer-to-peer basis. Each component is a client of other services and a server for the services it provides. Very often, a client for one request is a server for another. This architecture facilitates network programming by allowing the creation of distributed applications as sets of cooperating reusable objects that interact as though they were implemented in a single programming language on one computer.

Interface Definition Language (IDL):

Before a client can make a request to an object, it must know the types of operations supported by the object. An object's interface specifies the operations and types that the object supports and thus defines the requests that can be made to the object. Interfaces for objects are defined in the OMG Interface Definition Language (IDL). Interfaces are similar to classes in C++ and to interfaces in Java. IDL is a declarative language, not a programming language. It forces interfaces to be separate from object implementations. To use a discipline code, the user needs to know only what interfaces are implemented by that code.

Object Creation:

In the current implementation, Iona Technologies' implementation of the CORBA standard for the Java programming language, OrbixWeb, was chosen as the ORB. As a first step, all codes to be wrapped as objects are identified using a criterion such as reusability for their selection. Next, for each object, an interface file is written in IDL identifying the services offered by that object, the inputs, the outputs, and the types of errors the object can "throw". The object and its services (interpreted as function calls) are then implemented in Java. A service call is associated with the invocation of a discipline code. Discipline codes are mainly legacy codes written either in FORTRAN or C and are accessed through an intermediate function following Java's Native Interface (JNI) guidelines. Within the implementation file, a central relational database is queried for needed data and file information. Any required file management is done based on this file information. The third code component in this process of "objectifying" discipline codes is writing the "Server" class code. This component ties the implementation class to its IDL interface. Servers provide objects for use by clients and other servers.

A 'master' client program is then written to implement the design analysis algorithm by making service calls to the distributed application objects. When the master program is executed, the client calls are transferred to the ORB which then passes the function calls through the server code to the target object. Components of the ORB are implemented by the OrbixWeb daemons running on the client and the server hosts. After the target object execution is completed, responses are communicated back to the client via the ORB and as possible updates to the central database.

The Problem

This section describes the implementation of a simplified aircraft design optimization benchmark problem in the CORBA/Java based Object Framework. This problem is considered an excellent multidisciplinary optimization test case since it includes the interplay of multiple disciplines while carrying along only a small number of design variables, constraints, and a single objective function. The principal disciplines for the design problem are: aerodynamics, structures, propulsion, and performance. The design objective is to minimize the aircraft gross take-off weight for given cruise conditions, range, and payload requirements. The weight is minimized subject to aerodynamic, structural, and performance constraints such as limiting values of lift and drag, maximum stresses at critical points on the wing inboard and outboard panels, and range. Design variables include wing sweep, root chord, distance to break, and inboard and outboard skin thickness. Figure 3 shows the example HSCT model, without empennage and control surfaces, that was studied within this framework. Figure 4 describes the discipline segments and the information flow necessary for the design and analysis of an optimal HSCT configuration.

Implementation:

The discipline codes are "wrapped" using JNI methodology and encapsulated as objects in a CORBA Object Framework, for a network of Sun workstations. Sub-processes are also implemented as objects which use the lower level objects for their functionality.

The main program, the 'master' client written in Java, implements the design algorithm. It sends out requests for the services of the discipline objects residing on remote networked host computers. The BDK provides a graphical interface for user input, interactive visual object connectivity, and problem execution progress monitoring. To create the master client, selected object clients are implemented as JavaBeans and connected together using Java Studio, which is an interactive visual beans work environment. JavaBeans are depicted as icons (which can be animated). These can be equipped with "customizers", which are Motif style windows, that permit user interaction with the underlying object variables. User selections in these customizer windows override the default values programmed in the object definitions.

The OrbixWeb daemons must be up and running on the host and client computers before problem execution can begin. The ORB mediates the communication of the marshalled requests and starts the remote object, if required, before delivering the request. After the remote object completes servicing the request, the results are communicated back to the client by the ORB and as possible updates to the database.

At the start of the design process, initial run conditions are obtained from a central database and the various discipline segments are initialized. This initialization can be done in parallel on respective host computers because the disciplines are uncoupled at this stage. After initialization, the design optimization process cycles through analysis, gradient computation, and optimization phases until an optimum weight is obtained and the design variables have converged, or the specified limit on number of cycles is reached.

Analysis Phase:

The analysis phase begins with a calculation of drag polars using the medium-fidelity code Wingdes. Lift and drag values for a range of angles-of-attack are used in generating parametric representations of aerodynamic responses. All subsequent aerodynamic analyses for that design

cycle will utilize these drag polars to compute lift and drag. The next step in the analysis phase is the iteration for airplane weight convergence. The weight iteration loop begins with a static trim analysis where force balance is computed for two different load factors: a) load factor = 1.0 for drag calculation used in performance analysis, and b) load factor = 2.5 for loads calculation in structural design. The propulsion segment computes the current fuel flow rate. Next, the performance segment uses this flow rate to produce an estimate for fuel weight.

Structural analysis to determine structural weight is required only during the first iteration. A loads transfer program converts the aerodynamic pressure distribution to vertical forces on the structure at a trimmed angle-of-attack and a load factor of 2.5. The structural analysis program used in this problem is the Equivalent Laminated Plates Solution, ELAPS [Ref. 2]. The total weight is then computed as the sum of the fixed weights, structural weight, and the fuel weight. The aerodynamic, propulsion, and performance analyses are conducted iteratively until the total weight converges within a predefined tolerance.

Gradient Phase:

In the next phase, all the system response derivatives required by the optimizer are computed. The gradients of aerodynamic and structural constraints are computed using finite-differences. Gradients of the fuel weight with respect to design variables are obtained using a closed-form expression.

Optimization Phase:

Conmin with linear function approximations [Ref. 3] is used as the optimization program. The optimizer attempts to minimize the objective function, total airplane weight, subject to the aforementioned design constraints and computes an updated set of values for the design variables.

Results:

A typical HSCT configuration flying at Mach 2.4 at an altitude of 63,000 feet was analyzed using the analytical connectivity shown in Fig. 4. Figure 5 shows the variation of the structural design variables with cycle number while Figure 6 shows the variation of the airplane weight components with cycle number. These results are plotted along with similar results from an implementation of the current problem in the earlier Framework, FIDO.

These results indicate that the benchmark problem has been successfully implemented in the Object Framework and fully validated in comparison with the earlier implementation.

Conclusion:

Conceptual design systems like the one described here provide the aircraft designer with the analysis tools required to manage the complexity of multidisciplinary analysis and to unleash the computational resources required to design a realistic HSCT configuration. This approach is now being applied to a more realistic conceptual aircraft design problem using high fidelity analysis codes such as CFL3D for nonlinear aerodynamics, the Genesis structural analysis code, a detailed as built weights module, along with mission/performance and optimization codes using detailed structures, nonlinear aero- and linear aero- grids for multiple load conditions and configurations.

References

1. Steve Vinoski, "CORBA: Integrating diverse applications within distributed Heterogeneous Environments", IEEE Communications, Vol. 14, No. 2, February 1997.

2. Giles, G. L., "Equivalent Plate Analysis of Aircraft Wing Box Structures with General Planform Geometry", Journal of Aircraft, Vol. 23, No. 11, 1986, pp. 859-864.

3. Vanderplaats, Garret N., "CONMIN - a FORTRAN Program for Constrained Function Minimization User's Manual", NASA TM-X-62282, August 1973.

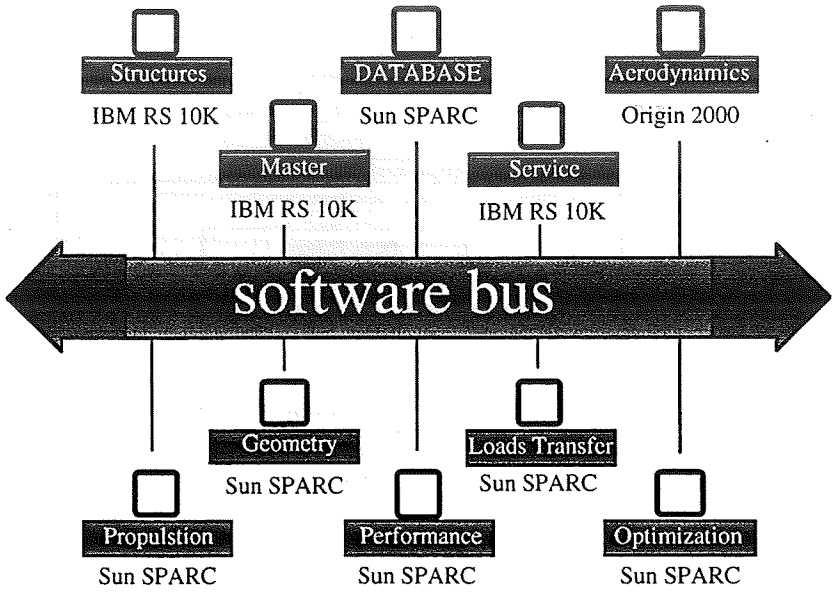


Figure 1. CORBA as a 'software bus'.

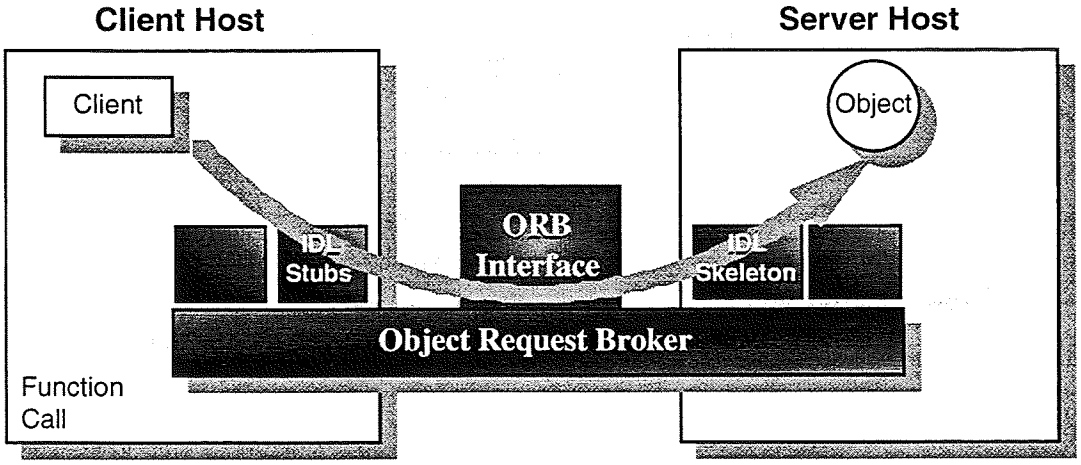


Figure 2. The Object Request Broker, ORB.

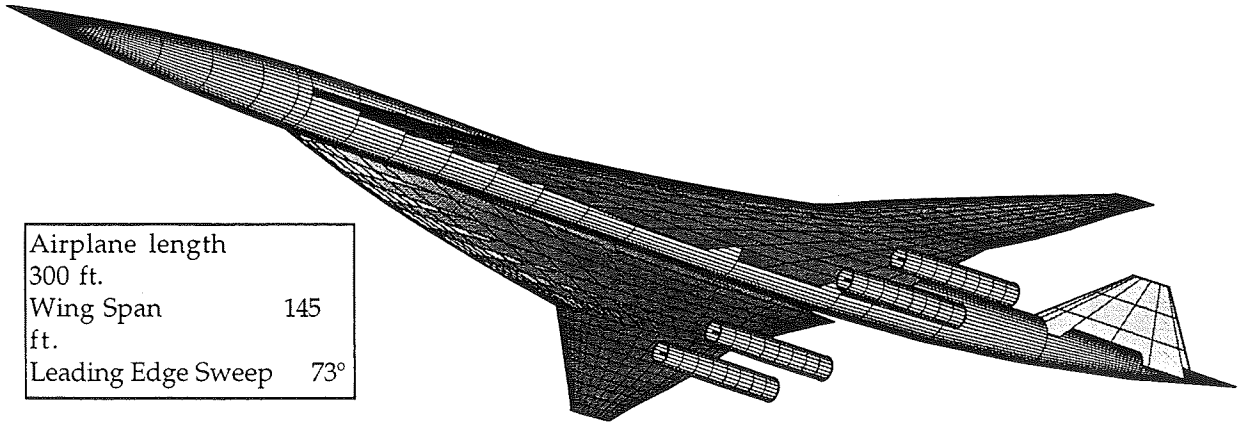


Figure 3. HSCT model problem and design point data.

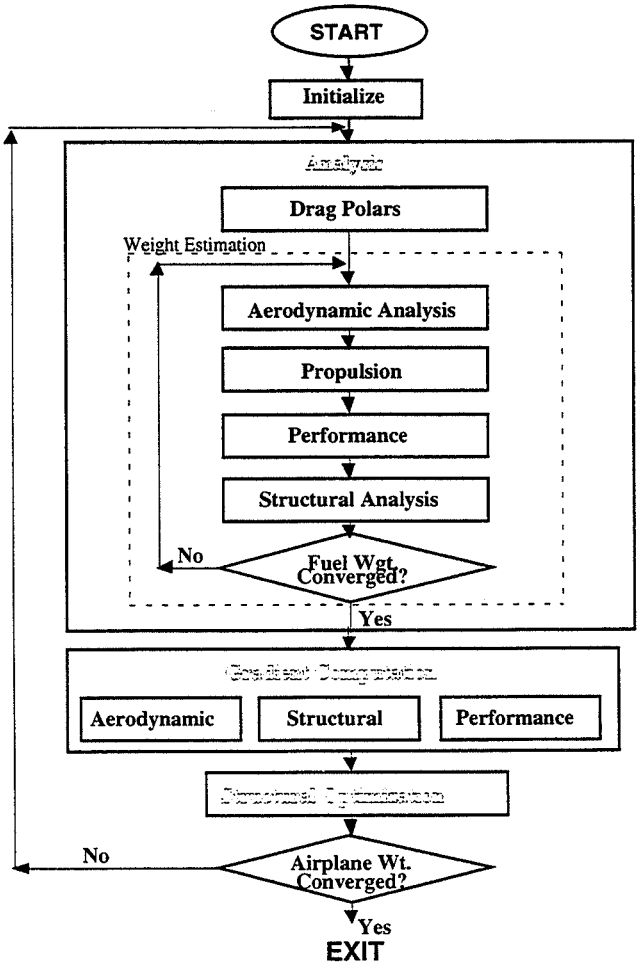


Figure 4. Flowchart for aircraft design optimization.

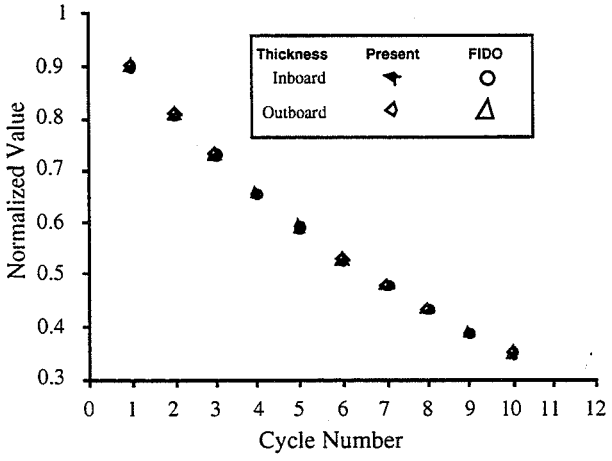


Figure 5. Variation of structural design variables.

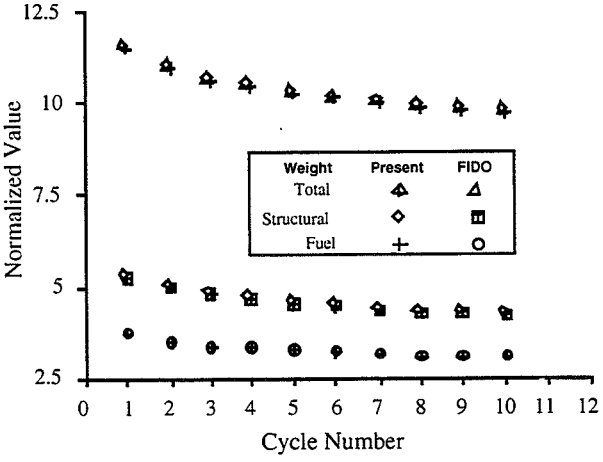


Figure 6. Variation of airplane weights.

NATIONAL CYCLE PROGRAM(NCP) COMMON ANALYSIS TOOL for AEROPROPULSION

G. Follen, C. Naiman, A. Evans/NASA Lewis Research Center
Gfollen@lerc.nasa.gov, **(216)433-5193**

Within NASA's High Performance Computing and Communication (HPCC) program, NASA Lewis is developing an environment for the analysis/design of aircraft engines called the Numerical Propulsion System Simulation (NPSS). NPSS is focused on the integration of multiple disciplines such as aerodynamics, structures and heat transfer with numerical zooming on component codes and computing/communication technologies to capture complex physical processes in a timely and cost-effective manner. The vision for NPSS is to be a "numerical test cell" that enables full engine simulation overnight on cost-effective computing platforms.

Through the NASA/Industry Cooperative Effort (NICE) agreement, NASA Lewis and industry partners are developing a new engine simulation, called the National Cycle Program (NCP), which is the initial framework of NPSS. NCP is the first phase toward achieving the goal of NPSS. This new software supports the aerothermodynamic system simulation process for the full life cycle of an engine. The National Cycle Program (NCP) was written following the Object Oriented Paradigm (C++, CORBA). The software development process used was also based on the Object Oriented paradigm. Software reviews, configuration management, test plans, requirements, design were all apart of the process used in developing NCP. Due to the many contributors to NCP, the stated software process was mandatory for building a common tool intended for use by so many organizations. The U.S. aircraft and airframe companies recognize NCP as the future industry standard for propulsion system modeling.

While NCP uses "Cycle" in its name, it by no means represents what cycle simulations had been characterized by in the past. The NCP was written to have all the features that currently existed in cycle simulations as well as having the ability to exercise the NPSS concepts of component code zooming, distributed/parallel processing and introduces through an API a means to conduct Multi-discipline simulations all from a "System" point of view. The NICE/NCP team found a way to come together into a pre-competitive environment to create a Common Analysis Tool for Aeropropulsion where the pooling of expertise and the sharing of the best features of all the companies could be leveraged and merged into one single effort. The NPSS/NCP team consists of representatives from:

NASA Lewis	General Electric Aircraft Engines	Pratt & Whitney
The Boeing Company	AlliedSignal Engines	Allison
Williams International	Teledyne Ryan Aeronautical	WPAFB
AEDC		

"BTP-Breaking the Paradigm"
NCP is being developed as the NPSS standard analysis framework based on the object-oriented paradigm featuring technologies like CORBA (Common Object Request Broker Architecture) by which aeronautics codes can be linked together in a collaborative environment. Executing a distributed simulation is now possible using the CORBA capabilities implemented within NCP. NCP is able to perform all the functions of a classical "cycle analysis code" as well zoom to higher order codes both within aerodynamic and structural class codes. Pictorially, what NCP has now allowed is illustrated by the following figure 1.

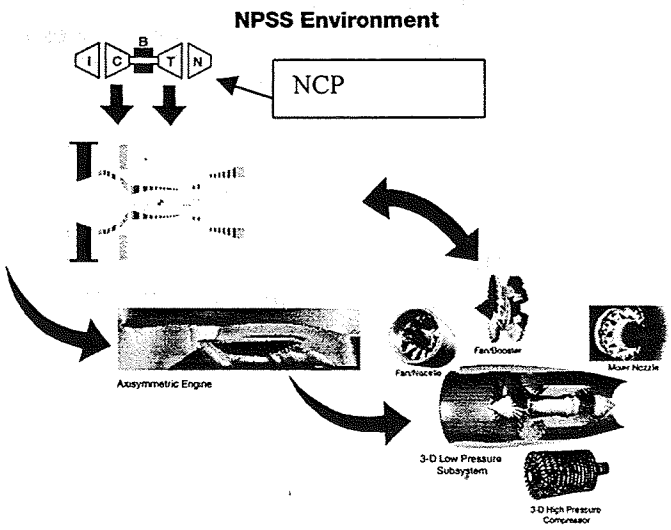


Figure 1. NCP and zooming

Historical Cycle Simulations

In a thermodynamic cycle system model of the engine, each component is represented by an overall (lumped) performance characteristic map. NCP is the model that is used to simulate the thermodynamic engine cycle. This model is obtained by matching the characteristics map of each component of the engine in the system model. During conceptual design of an engine, the performance of each component is estimated from an empirically derived database or from prior rig testing. In the case of a new component with no prior test history, the component performance is derived from a database and is a first approximation. In the conceptual design of an engine, component modeling codes are used to approximate the thermodynamic performance characteristics. These component performance characteristic maps are updated at a later point in the design cycle based on analysis of the components with Navier-Stokes flow codes, and eventually with test data.

In place within the Aeropropulsion Industry, many FORTRAN based simulation systems exist. Each of these systems mentioned here can potentially be replaced with NCP:

SOAPP – State of the Art Performance Program, initially developed at Pratt & Whitney in 1971.	**GSA**- research propulsion program used by Boeing written in the 1960's
ROCETS- modular rocket design and analysis system	**FAST**-in use at Allied Signal developed in the 1980's.
CWS-GE Aircraft Engines engine performance system, developed in the 80's	**ATEC**- Aerodynamic Turbine Engine Code used at AEDC, simulates dynamic behavior
RRAP-modular simulation system in use by Rolls Royce.	**TERMAP**- used by Allison Engine Company, namelist input format
NNEP- a simulation program currently supported by NASA Lewis Research Center	**BIEPP**-the Boeing installed Engine Performance Program.

The NCP Team has collected the "best of the best" in defining the requirements for a cycle analysis tool with extensions in this tool for zooming principles and establishing the effects of multi-disciplines. The use of a common tool and just as important, the knowledge of the physics within NCP, is invaluable to the NPSS/NCP team. Companies spent huge sums of money and exhaust many person hours in matching and integrating data from differing cycle systems with their own simulation systems. NCP offers many advantages over the current industrial practices which is characterized by as many different simulation systems as there are companies. A few advantages to an industry wide tool are:

- **This is not another cycle tool: NCP preserves the cycle system view and extends it to allow for the creation of better models;**
- Common tool reduces matching of answers when multi-company contracts are awarded;
- Company proprietary issues are still preserved because data used within the NCP is still held proprietary;
- Reduced costs in training and needed knowledge of other's systems;
- NCP was built based upon the object oriented paradigm that facilitates re-use of tested objects, sharing of created objects and permits the creation of new objects.

National Cycle Program Simulations

The NCP preserves what "Classical Cycle Simulations" have done and what they are needed for: Steady state aerothermal system performance, Calibration to measured behavior, Open loop transient performance predictions and Closed loop coupled aerothermal/Control system performance predictions. More importantly though, NCP was written to extend these features to include the integration of higher order CFD, multi-disciplines and parallel/distributed processing of each engine element while preserving an overall engine system view.

Common Objects within NCP

NCP simulations are built from the follow objects:

- Elements: Compressor, Turbine, Inlet, etc
- Subelements: Compressor Map, Turbine Map, Nozzle Map, etc
- Ports: Fluid, Fuel, Mechanical Rotation, Thermal Heat Transfer
- Flow Stations: Therm Library, JANAF Library, CEC
- Tables: 1st, 2nd , 3rd order Lagrangian
- Output Objects: Data dump, Page & window viewers
- NCP Input syntax allows for user defined elements and subelements and provides a macro language for user input

NCP's current delivery schedule is as follows and is illustrated in Figure 2.:

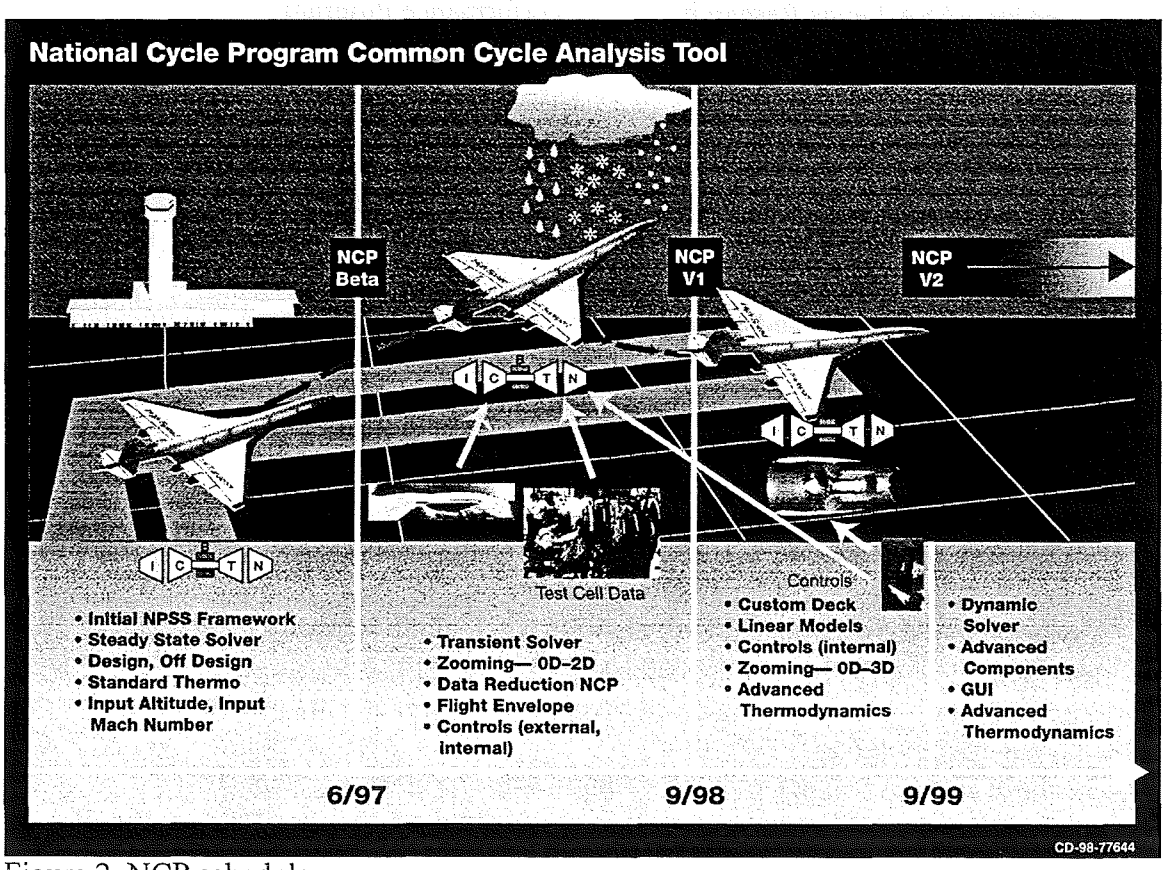


Figure 2. NCP schedule

In FY99, the NCP team will complete the major pieces of the Cycle system including Customer Deck generation, Dynamic solver capability and finish off the complete set of engine components. At the same time, NCP's development will shift to a major emphasis on Zooming. NASA Lewis is completing the connections between NCP and Wate through the NCP's External Element API definition. Additionally, Pratt and Whitney will define the object layer for connecting NCP to 1 and 2 Dimensional meanline and streamline codes for use in companies design process. NCP will also be connected to the

LAPIN inlet code which introduces a number of interesting engineering and computer science issues: Dynamic solver, geometry, multiple solvers.

Summary

The NCP offers a unique opportunity to the US Aeropropulsion Industry in that this group found a pre-competitive business area where they come together to build a common tool and push a new technology (NPSS). Many known and quantifiable benefits exist for the adoption of a common cycle analysis tool such as NCP. These benefits will be further extended as NPSS conducts "zooming" technology pushes from NCP to meanline/streamline codes through 3Dimensional Aero codes. These zooming activities are a focus of the NPSS project in FY99.

The NCP as part of NPSS is supported under the NASA High Performance Computing and Communications Program.

BIBLIOGRAPHY

1. "Numerical Propulsion System Simulation's National Cycle Program", A. Evans, G. Follen, C. Naiman, 1998 Joint Propulsion Conference AIAA-98-3113.

2. "The National Cycle Program: A Flexible System Modeling Architecture", R. Ashleman, T. Lavelle, F. Parsons, 1998 Joint Propulsion Conference AIAA-98-3114.

NCC - A MULTI-DISCIPLINARY DESIGN/ANALYSIS TOOL FOR COMBUSTION SYSTEMS

Nan-Suey Liu
NASA Lewis Research Center, MS 5-11, (216)433-8722, Nan-Suey.Liu@lerc.nasa.gov
And
Angela Quealy
Dynacs Engineering/ NASA LeRC, (216)977-1297,quealy@lerc.nasa.gov

NASA Computational Aerosciences Workshop 98 (8/25-8/27)

1 Introduction

A multi-disciplinary design/analysis tool for combustion systems is critical for optimizing the low-emission, high-performance combustor design process. Based on discussions between NASA Lewis Research Center and the jet engine companies, an industry-government team was formed in early 1995 to develop the National Combustion Code (NCC), which is an integrated system of computer codes for the design and analysis of combustion systems. NCC has advanced features that address the need to meet designer's requirements such as "assured accuracy", "fast turnaround", and "acceptable cost". The NCC development team is comprised of Allison Engine Company (Allison), CFD Research Corporation (CFDRC), GE Aircraft Engines (GEAE), NASA Lewis Research Center (LeRC), and Pratt & Whitney (P&W). This development team operates under the guidance of the NCC steering committee (Ref. [1]).

The "unstructured mesh" capability and "parallel computing" are fundamental features of NCC from its inception. The NCC system is composed of a set of "elements" which includes grid generator, main flow solver, turbulence module, turbulence and chemistry interaction module, chemistry module, spray module, radiation heat transfer module, data visualization module, and a post-processor for evaluating engine performance parameters. Each element may have contributions from several team members. Such a multi-source multi-element system needs to be integrated in a way that facilitates inter-module data communication, flexibility in module selection, and ease of integration.

2 Status

The development of the NCC beta version was essentially completed in June 1998. A brief description of the various elements follows.

Grid Generator

CFD-GEOM is an established geometry modeling and structured grid generation system developed and commercialized by CFDRC. Under the NCC development effort, it has been further developed to enable unstructured grid generation and direct CAD interfacing. An overview of this module can be found in Ref. [2]. It should be noted that other grid generators could be used with NCC if the mesh information is written in the Patran neutral format.

Main Flow Solver

The original flow solver (CORSAIR) was developed at P&W, under the present effort; it has been upgraded to CORSAIR-CCD, which is the current baseline gaseous flow solver for the NCC. CORSAIR-CCD is a 3-dimensional, Navier-Stokes flow solver based on an explicit four-stage Runge-Kutta scheme, using unstructured meshes, and running on networked workstations. The discretization begins by dividing the spatial computational domain into a large number of elements, which can be of mixed type. A central -difference finite-volume scheme augmented with a dissipative operator is used to generate the discretized equations, which are then advanced temporally by a dual time procedure for computing the low Mach number compressible flows. The history and capabilities of this solver, and some validation test cases are given in Ref. [3]. Its use in a design system is demonstrated in Ref. [4]; and ten benchmark test cases have been reported in Ref. [5].

Turbulence Module

Originally, the turbulence closure is obtained via the standard k-epsilon model with high Reynolds number wall function. Under the present effort, the following capabilities have been added: a low Reynolds number wall integration scheme, a new non-linear k-epsilon model for swirling flows, and a model which does not contain wall distance in its formulation. The details of these new models and their validations can be found in Refs. [6],[7],[8], and [9].

Turbulence / Chemistry Interaction Module

Several options are available in the NCC, they are: a single-step eddy breakup model for mixed-is-burned situations; an assumed Probability Density Function (PDF) model for finite-rate chemistry; and a model based on solving the transport equation for the joint PDF of scalars. The assumed PDF model uses an enthalpy variance transport equation, with an assumed beta-PDF for modulation of the forward and backward kinetic rate coefficients.

In order to better account for the interactions between turbulence and chemical reaction rates, a scalar PDF module for unstructured mesh and parallel computing has been developed and coupled with the main flow solver. This module solves a transport equation for the joint PDF of scalars (species, enthalpy, etc.) by using Monte Carlo techniques. The chemical reactions and their interaction with turbulence appear in a closed form in this transport equation and need not be modeled. Details of this module and its validation are given in Refs. [10] and [11].

Chemistry Module

Several reduced chemical mechanisms have been developed for use in NCC to compute NOX concentrations in the combustion of Jet-A and methane fuels. Refs. [5] and [12] describe the reduced mechanisms based on quasi-steady state and partial-equilibrium assumptions. A set of transport equations for the species retained in the resulting reduced mechanism needs to be solved along with other transport equations for flow variables. Alternatively, a code has been developed to automatically simplify full chemical mechanisms. The method employed is based on the Intrinsic Low Dimensional Manifold (ILDM) method. The resulting simplified mechanism is stored in look-up tables, which contain both reaction rate information and fluid properties. The implementation of the ILDM look-up tables in the NCC and its performance are discussed in Ref. [13]. It is shown that the ILDM kinetics approach results in significant reduction in CPU time.

Spray Module

In NCC, a Lagrangian based dilute spray model provides the solution for the liquid phase equations. Ref. [14] discusses its coupling with the gaseous flow solver and the scalar PDF module. A more detailed description of this module can be found in Ref. [15].

Radiation Heat Transfer Module

The present version of the module performs a finite-volume three-dimensional radiative heat transfer analysis using body-fitted structured meshes. Models for non-gray media including the non-gray interaction between certain gaseous species, as well as models for luminous radiation from soot, have been developed (Ref. [16]). A radiation module for unstructured mesh and parallel computing is currently being developed under the present effort.

Data Analysis

Several commercial software packages have been used for data visualization: Tecplot, Fieldview, Pv3_Gold, and CFD-View. A post-processor, CORPERF, has been developed to provide combustor performance parameters.

Integration Framework

Integration of the various modules is not fully modularized yet. The key NCC integration software technologies are the Data Transfer Facility (DTF) and the Multi-Disciplinary Computing Environment (MDICE) both developed at CFDRC. The DTF is used for static (file-based) coupling between modules, while the MDICE is used for dynamic (run-time) data exchange between NCC modules, as well as for integration between NCC and other flow and/or structural codes. A more detailed description is given in Ref. [2].

3 Parallel Performance

The objective of the NCC parallel processing effort is to reduce the overall turnaround time of a large-scale, fully reacting combustor simulation to 15 hours. The resulting code must continue to be portable to a wide variety of computing platforms and must run efficiently on a given target platform. Such a parallel improvement effort contributes significantly to the overall reduction in time and cost of the combustor design cycle.

The original solver (CORSAIR) was developed to run in a networked workstation environment using a proprietary message passing library (PROWESS). It was later ported to both PVM and MPI message passing libraries. The current NCC baseline flow solver (CORSAIR-CCD) may select any of the three message passing libraries at compile time. MPI has been used for this performance evaluation effort due to the availability of vendor optimized versions of MPI on target platforms of interest. CORSAIR-CCD was first ported to the IBM SP-2 and is now being further refined to run efficiently on an SGI Origin 2000. The code remains portable and efficient on a variety of parallel platforms including networks of personal computers. Some of the performance results have been reported in Refs. [1], [5], and [17].

A Lean Direct Inject (LDI) Combustor is being used to measure the performance of a typical large-scale combustor simulation. A 12 species, 10 steps reduced kinetics is being used to account for the amount of computational resources required for chemistry simulation. Current test cases consist of both a 444 thousand (444k) element and a 971k element geometry, however the problem size of interest is approximately 1.3 million elements. Until a geometry of this size becomes available the performance results of these smaller test cases are scaled up to estimate the performance of the larger problem.

Performance profiling tools have been used to identify where CORSAIR-CCD is consuming time and these sections of code have been systematically improved. Initial effort focused on streamlining the chemistry section of CORSAIR-CCD when profiling tools indicated this section consumed 54% of the execution time. Improvements such as eliminating unnecessary indexing and using more efficient operations yielded a 1.8x (1.8 times) improvement in performance. Replacing one statement which executed an exponentiation operation (a**0.25) with square root intrinsics (sqrt(sqrt(a)) resulted in a 1.3x improvement in performance alone. A more deadlock resistant communication pattern algorithm allowed increasing the number of processors in use resulting in additional performance savings. Finally the ILDM kinetics module was integrated with CORSAIR-CCD and was used as an option to provide the finite rate chemistry information, yielding a 4.7x improvement in performance for a large-scale, fully reacting combustor simulation.

Performance results from both the 444k element and 971k element LDI Combustor test cases currently indicate that a solution to a large-scale, fully reacting combustor simulation (1.3 million elements) can be achieved within an 18 hour turnaround. This is a performance improvement of almost 30x over the 1995 baseline and approximately a 175x improvement using this same baseline on a 1992-era parallel machine such as the Intel Paragon. CORSAIR-CCD currently achieves 1.4 GFLOPS using 32 processors on the SGI ORIGIN 2000 when running the 444k element LDI test case. The speedup is 24.2 and the efficiency is 76%. Effort is currently focusing on reducing the impact of message passing in order to improve the overall performance efficiency on the ORIGIN platform. Very recent benchmarks indicate a 15-hour turnaround is now achievable due to upgraded processor speeds of 250 MHz on the ORIGIN 2000.

4 Concluding Remarks

The development of the NCC beta version is essentially completed. Elements such as CORSAIR-CCD, Turbulence Module, and the Chemistry Module, have been extensively validated; and their parallel performance on large-scale parallel systems has been evaluated and optimized. However the scalar PDF Module and the Spray Module, as well as their coupling with CORSAIR-CCD, were developed in a small-scale distributed computing environment. As a result, the validation of the NCC beta version as a whole is quite limited and its overall performance on large-scale parallel systems is yet to be evaluated and optimized. Future work includes porting the NCC beta version to large-scale parallel systems to conduct validation cases of practical interests and to evaluate and optimize the overall parallel performance. In addition, the integration of NCC elements via DTF and MDICE will be continued to achieve a "plug-and-play" capability.

Acknowledgements

The NASA High Performance Computing and Communications Program (HPCCP) and the Smart Green Engine Program (SGE) have supported this effort.

REFERENCES

[1] Stubbs, R.M., and Liu, N.-S., "Preview of National Combustion Code," AIAA Paper 97-3114, July 1997.
[2] Harrand, V.J., Siegel, J.M., Singhal, A.K., and Whitmire, J.B., "Key Components and Technologies for the NCC Computing Framework," AIAA Paper 98-3857, July 1998.
[3] Ryder, R.C., "The Baseline Solver for the National Combustion Code," AIAA Paper 98-3853, July 1998.
[4] Brankovic, A., Ryder, R.C., and Syed, S.A., "Mixing and Combustion Modeling for Gas Turbine Combustors Using Unstructured CFD Technology," AIAA Paper 98-3854, July 1998.
[5] Chen, K.-H., Norris, A.T., Quealy, A., and Liu, N.-S., "Benchmark Test Cases for the National Combustion Code," AIAA Paper 98-3855, July 1998.
[6] Chen, K.-H., and Liu, N.-S., "Evaluation of A Non-Linear Turbulence Model Using Mixed Volume Unstructured Grids," AIAA Paper 98-0233, January 1998.
[7] Shih, T.-H., and Liu, N.-S., "A k-epsilon Model for Wall-Bounded Shear Flows," AIAA Paper 98-2551, June 1998.
[8] Shih, T.-H, Chen, K.-H., and Liu, N.-S., "A Non-Linear k-epsilon Model for Turbulent Shear Flows," AIAA Paper 98-3983, July 1998.
[9] Nikjoo, M., and Mongia, H.C., "Study of Non-Linear k-epsilon Model for Turbulent Swirling Flows," AIAA Paper 98-3984, July 1998.
[10] Raju, M.S., "Extension of the Coupled Monte-Carlo-PDF/SPRAY/CFD Computations to Unstructured Grids and Parallel Computing," AIAA Paper 97-0801, January 1997.
[11] Anand, M.S., James, S., and Razdan, M.K., "A Scalar PDF Combustion Model for the National Combustion Code." AIAA Paper 98-3856, July 1998.
[12] Kundu, K.P., Penko, P.F., and Yang, S.L., "Simplified Jet-A/Air Combustion Mechanisms for Calculation of NOX Emissions," AIAA Paper 98-3986, July 1998.
[13] Norris, A.T., "Automated Simplification of Full Chemical Mechanisms: Implementation in National Combustion Code," AIAA paper 98-3987, July 1998.
[14] Raju, M.S., "Combined Scalar-Monte-Carlo-PDF/CFD Computations of Spray Flames on Unstructured Grids with Parallel Computing," AIAA Paper 97-2969, July 1997.
[15] Raju, M.S., "LSPRAY-A Lagrangian Spray Solver-User's Manual," NASA CR-97206240, November 1997.
[16] Kumar, G.N., Mongia, H.C., and Moder, J.P., "Validation of Radiative Heat Transfer Computations Module for National Combustion Code," AIAA Paper 98-3985, July 1998.
[17] Liu, N.-S., Quealy, A., Kundu, K.P., Brankovic, A., Ryder, R.C., and Van Dyke, K., "Multi-Disciplinary Combustor Design System and Emissions Modeling," NASA CDCP-20011, Proceedings of the 1996 Computational Aerosciences Workshop, May 1997, pp. 49-54.

Gary Cole
NASA Lewis Research Center
21000 Brookpark Rd., Cleveland OH 44135
Gary.Cole@lerc.nasa.gov, 216-433-3655

Ambady Suresh and Scott Townsend
Dynacs Engineering Co., Inc.
2001 Aerospace Parkway
Brook Park, OH 44142
Ambady.Suresh@lerc.nasa.gov, 216-977-1384
Scott.Townsend@lerc.nasa.gov, 216-977-1080

INLET-COMPRESSOR ANALYSIS USING COUPLED CFD CODES

Introduction

Propulsion performance and operability are key factors in the development of a successful aircraft. For high-speed supersonic aircraft, mixed-compression inlets offer high performance but are susceptible to an instability referred to as unstart. An unstart occurs when a disturbance originating in the atmosphere or the engine causes the shock system to be expelled from the inlet. This event can have adverse effects on control of the aircraft, which is unacceptable for a passenger plane such as the high speed civil transport (HSCT).

The ability to predict the transient response of such inlets to flow perturbations is, therefore, important to the proper design of the inlet and the control measures used to prevent unstart. Computational fluid dynamics (CFD) is having an increasing role in the analysis of individual propulsion components. Isolated inlet studies are relatively easy to perform, but a major uncertainty is the boundary condition used at the inlet exit to represent the engine - the so-called compressor face boundary condition. A one-dimensional (1-D) Euler inlet simulation (ref. 1) showed that the predicted inlet unstart tolerance to free-stream pressure perturbations can vary by as much as a factor of about six, depending on the boundary condition used. Obviously, a thorough understanding of dynamic interactions between inlets and compressors/fans is required to provide the proper boundary condition.

To aid in this understanding and to help evaluate possible boundary conditions, an inlet-engine experiment was conducted at the University of Cincinnati (ref. 2). The interaction of acoustic pulses, generated in the inlet, with the engine were investigated. Because of the availability of experimental data for validation, it was decided to simulate the experiment using CFD. The philosophy here is that the inlet-engine system is best simulated by coupling (existing) specialized CFD component-codes. The objectives of this work were to aid in a better understanding of inlet-compressor interaction physics and the formulation of a more realistic compressor-face boundary condition for time-accurate CFD simulations of inlets. Previous simulations have used 1-D Euler engine simulations in conjunction with 1-D Euler and axisymmetric Euler inlet simulations (refs. 3,4). This effort is a first step toward CFD simulation of an entire engine by coupling multidimensional component codes.

Inlet-Engine Experiment

A sketch of the inlet-engine experiment conducted at the University of Cincinnati is shown in figure 1. The inlet is a constant-area annular duct. An acoustic pulse generator in the inlet is composed of an inflatable, flexible bump around the hub of the inlet and a unique mechanism internal to the hub for rapidly collapsing the bump. A bellmouth and constant area section of duct, upstream of the bump, are not shown in the sketch. The inlet is connected to a GE T-58 engine modified for cold operation. The engine has a ten stage compressor, preceded by a variable inlet guide vane (VIGV). The bump collapse results in two well-defined acoustic pulses (expansion waves), one traveling upstream and the other traveling downstream that interacts with the engine. Static pressure time histories were measured by high-response transducers located at the four axial stations shown in figure 1.

Inlet Simulation

The inlet, up to the engine face, and collapsing-bump portions of the experiment were simulated using NPARC (ref. 5), a general purpose CFD code capable of handling moving grids. Although the flow in the duct is axisymmetric, it was solved here as an Euler 3-D flow through a sector, since the inlet simulation is coupled to a 3-D turbomachinery simulation.

The inlet simulation used a single-block grid consisting of 186 x 33 x 13 points in the axial, radial, and circumferential directions, respectively. The NPARC default ADI algorithm was used to obtain the reference steady-state solution for the isolated inlet. The exit boundary condition was chosen to match the experimental inlet mass flow rate. A Newton iterative solution, which uses iterations of the steady-state algorithm, was used for the unsteady computations. This allows larger time steps than the default explicit algorithm for unsteady calculations. The bump height and collapse time used in the simulation were chosen to match the experimental pressure profile of the initial pulse, because of the uncertainty of the exact values during the experiment.

Engine Simulation

The engine was approximated by the first stage rotor and was simulated by using ADPAC (ref. 6), a turbomachinery code. The first stage rotor was gridded over one blade passage using a C grid. The domain was divided into seven blocks, each grid consisting of 18 x 33 x 33 points. A steady reference solution was also obtained for the isolated rotor simulation. The exit boundary condition of static pressure at the hub was adjusted to achieve the experimental mass flow rate.

The real engine has a variable inlet guide vane (VIGV) upstream of the rotor to permit optimal performance of the engine at off-design conditions. An unsteady simulation of both the VIGV and the rotor requires the solution of a rotor-stator interaction, an extremely complex and computationally intensive problem. It was desired to avoid this complexity, but the turning provided by the VIGV could not be ignored because without it the rotor solution was close to stall and very different from the experimental condition being simulated. As a first approximation, the problem was solved by imposing a turning angle on the flow at the rotor inlet.

In effect, the flow from the inlet is instantaneously turned by the full metal angle of the VIGV blade. This procedure conserves the mass flow rate but not the total energy.

The engine (rotor) computations were viscous, using the Baldwin-Lomax turbulence model. For the unsteady simulation, the ADPAC implicit unsteady algorithm was chosen because the time step restrictions of the explicit option are prohibitively expensive. An unsteady non-reflecting condition was used at the exit boundary to prevent the interference of reflections not relevant to the experiment. Initially a single-block solution was used, but ultimately the ADPAC domain was divided into seven blocks and run in parallel to achieve faster execution.

Code Coupling Approach

Once the isolated component solutions are obtained, it is necessary to connect the codes in some fashion to achieve the coupled inlet-engine solution. Since NPARC and ADPAC are both multi-block codes, the basic method used to couple the codes is similar to that used to couple two blocks of a single multi-block code. Even though both codes are multi-block, there are a number of issues that required resolution to make the coupling successful, including: NPARC is finite difference whereas ADPAC is finite volume; NPARC exchanges data once per time step, ADPAC once per Runge-Kutta stage; the types of variables exchanged between blocks are different.

The actual software mechanism used to couple the codes is the Visual Computing Environment (VCE, ref. 7) being developed by CFD Research Corporation. (The current software is called Multi-Disciplinary Computational Environment or MDICE and is intended to facilitate coupling of multi-dimensional/disciplinary codes.) The software consists of a graphical user interface and subroutine libraries that provide means to control the execution of one or more (possibly distributed) application codes and the communication between them, as well as grid generation and visualization tools. A script approach is used to drive the process and the PVM message-passing paradigm is used to transfer data.

Modifications to NPARC and ADPAC were obviously required to accommodate the code coupling and integration with VCE. The differences in the codes also required some approximations so that the coupled codes cannot claim to have the same accuracy as a single multi-block code. Therefore, the coupled-code accuracy was assessed by solving some well-known unsteady test cases with good results.

Results

Visualization software was used to monitor the flow field and pressure time histories at the four sensor locations during execution of the coupled inlet-engine simulation. A snapshot of the monitor screen near the end of the simulation is shown in figure 2. The time histories clearly indicate the passage of the expansion pulse at the four sensor locations. The flow visualization portion shows pressure contour plots. The entire NPARC domain is not shown, only a portion near the NPARC/ADPAC interface (indicated by the heavy vertical line at the beginning of the

duct turning). A direct comparison of the computational and experimental results are shown in figure 3 where the change in pressure at each station from its initial steady-state value divided by its initial steady-state value is shown as a function time. Both sets of data were filtered to eliminate frequencies above 2000 Hz. This was done to eliminate a 3000 Hz oscillation, believed to be due to transverse mode oscillations setup by the bump collapse. The oscillation is clearly visible in the station 1 runtime result beginning at about 0.008 second in figure 2, and it was in good agreement with the amplitude and frequency of the unfiltered experimental data. The computed and experimental incident waves at stations 1-3 are in especially good agreement both having the same "peak" amplitude of about -0.038 at all three stations. The time history at station 4 is of major interest because, in the time frame shown, it is the only one influenced by the reflection from the engine. It can be noted that the peak amplitude is about -0.043, greater than that for the other three stations. The peak values at station 4 show excellent agreement, but in the region of four milliseconds the agreement could be better. It is expected that the addition of more compressor stages to the simulation would improve the agreement in that time frame. However, adding more stages would significantly increase the complexity of the simulation and the associated execution time.

The coupled NPARC-ADPAC solution, shown in figure 3, took approximately 32.5 hours for a 10.4 millisecond transient. (Time equal zero corresponds approximately to the start of the bump collapse, which occurred after several milliseconds of initialization.) The computations were made in a distributed computing environment under nearly dedicated conditions with VCE and NPARC running on a two-processor machine and ADPAC on an eight-processor machine. All processors were SGI R10000s. Initially the simulation was run with a single ADPAC block and took several days to execute on two non-dedicated processors. Some timing benchmarks on nondedicated machines indicated that dividing the ADPAC domain into seven (almost) equal blocks would result in a speedup of about 4.5, which seems to be confirmed by the results. ADPAC still dominates the execution time. It is believed that additional parallelization of ADPAC and running on a dedicated shared-memory machine with native message passing will improve the speedup. Still, significant speedup is required for practical inlet analysis and design run times and begs for a simplified compressor face boundary condition that will accurately represent the engine. Such a boundary condition was recently proposed in reference 1.

Concluding Remarks

This investigation indicates that coupling inlet and turbomachinery CFD codes is a feasible approach to study inlet-engine interaction problems. A multi-block coupling offers a quick and relatively easy way to couple together two CFD codes for both steady-state and unsteady computations. This approach offers the possibility of including other specialized codes (e.g. combustor) to provide a full engine simulation. The computational results gave good agreement with the collapsing bump experiment, but significant speedup is required to make the approach practical as a design/analysis tool. The coupled NPARC-ADPAC codes could also serve as a test bed for exploring other flow perturbations of interest, such as convective temperature and tangential velocity disturbances, and for validation of simplified boundary conditions.

Acknowledgments

The authors would like to thank the General Electric Company for making available the T-58 flow path and blade geometries. Thanks are also due to Rod Chima and John Slater of NASA Lewis for their help with grid generation and other advice, Christopher Miller of Lewis and Ed Hall of Allison for help with ADPAC, Jim Schmidt of Dynacs for help with the T-58 blade design, and Anthony Opalski of the U. Cincinnati for filtering the simulation results in the same manner as the experimental data.

References

1. Paynter, G. C., Clark, L. T., and Cole, G. L., "Modeling the Response from a Cascade to an Upstream Acoustic Disturbance," AIAA paper 98-0953, January 1998.
2. Freund, D. and Sajben, M., "Reflections of Large Amplitude Pulses from an Axial Flow Compressor," AIAA paper 97-2879, July 1997.
3. Garrard, D., Davis, M. Jr., Wehofer, S., and Cole, G., "A One-Dimensional, Time Dependent Inlet / Engine Numerical Simulation for Aircraft Propulsion Systems," ASME paper 97-GT-333, June 1997.
4. Numbers, K. and Hamed, A., "Development of a Coupled Inlet-Engine Dynamic Analysis Method," AIAA paper 97-2880, July 1997.
5. Chung, K., Slater, J. W., Suresh, A., and Townsend, S., "NPARC v3.1 User's Guide," October 1997.
6. Hall, E. J. and Delaney, R. A., "ADPAC User's Manual," NASA CR 195472, May 1996.
7. "VCE Reference Manual, Version 2.6," CFD Research Corporation, October 1997.

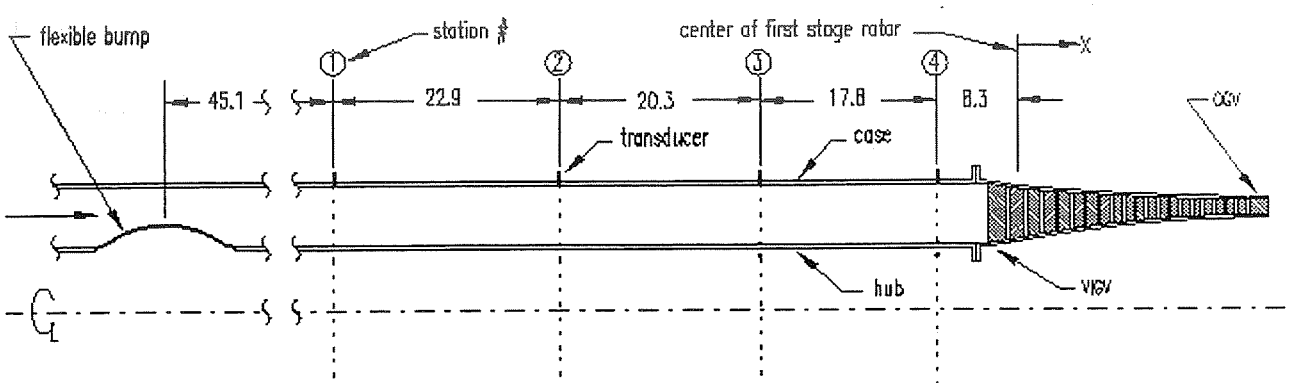


Figure 1. - Schematic of U. Cincinnati inlet-engine acoustic pulse experiment and pressure sensor locations (dimensions in centimeters).

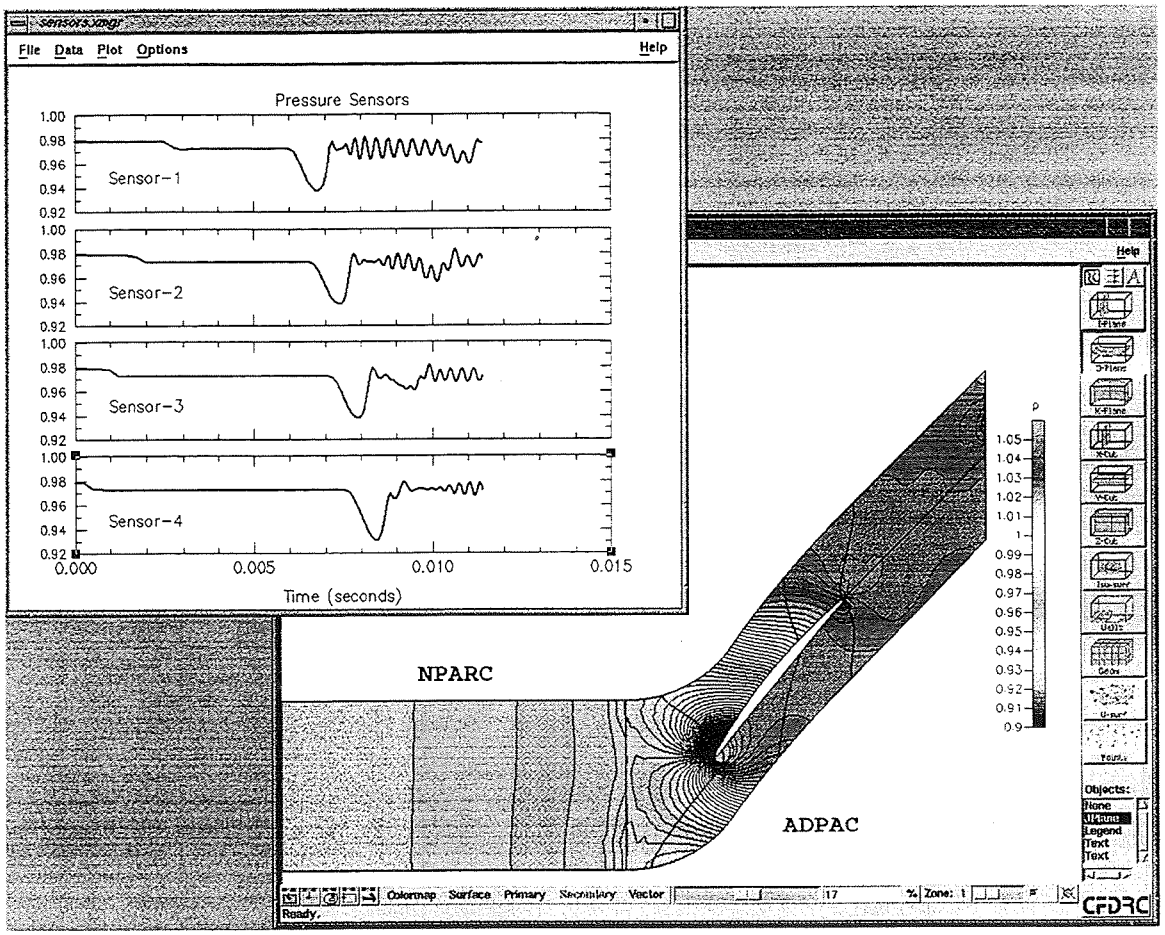


Figure 2. - Snapshot of (partial) NPARC/ADPAC pressure contour plots and pressure time histories at sensor locations during simulation of U. Cincinnati experiment.

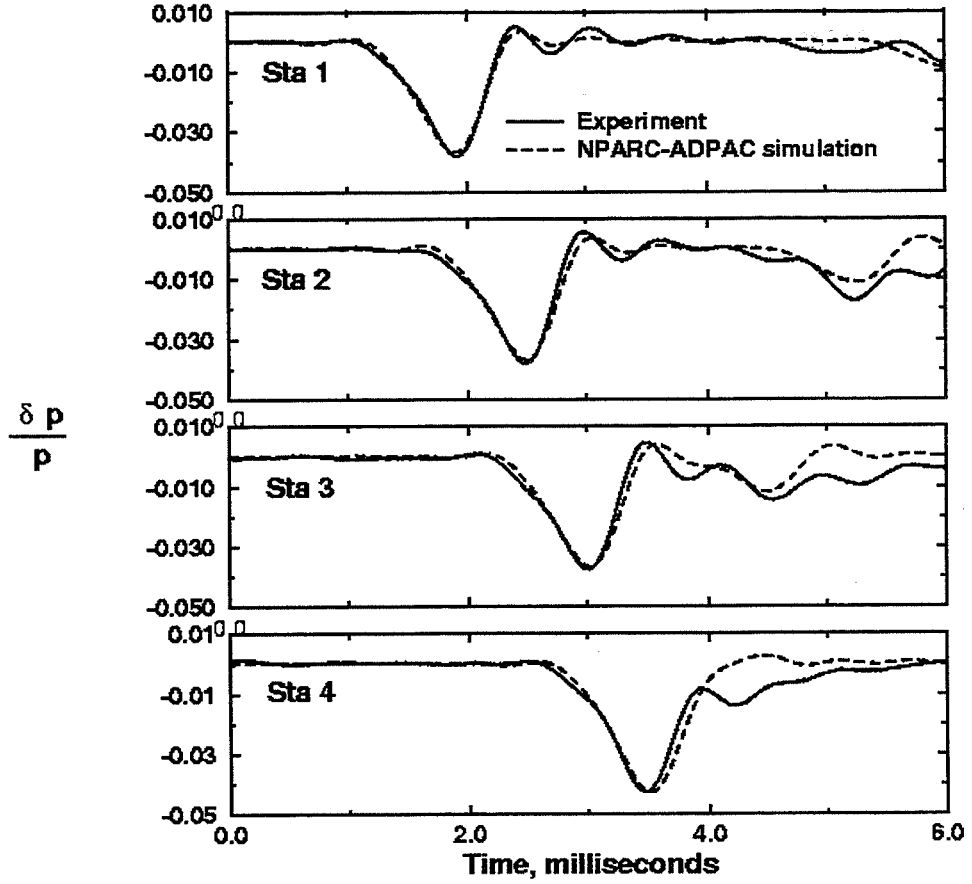


Figure 3. - Comparison of experimental and NPARC-ADPAC simulation results for inlet static pressures at four axial locations (Sta 1-4).

S37-31
018272

AEROSPACE ENGINEERING SYSTEMS AND THE ADVANCED DESIGN TECHNOLOGIES TESTBED EXPERIENCE

366898
81

William R. Van Dalsem, Mary E. Livingston, John E. Melton, Francisco J. Torres, and Paul M. Stremel
Aeronautical Information Technologies Division
NASA Ames Research Center
Moffett Field, California, United States 94035

Introduction

Continuous improvement of aerospace product development processes is a driving requirement across much of the aerospace community. As up to 90% of the cost of an aerospace product is committed during the first 10% of the development cycle, there is a strong emphasis on capturing, creating, and communicating better information (both requirements and performance) early in the product development process. The community has responded by pursuing the development of computer-based systems designed to enhance the decision-making capabilities of product development individuals and teams.

Recently, the historical foci on sharing the geometrical representation and on configuration management are being augmented:

- Physics-based analysis tools for filling the design space database;
- Distributed computational resources to reduce response time and cost;
- Web-based technologies to relieve machine-dependence; and
- Artificial intelligence technologies to accelerate processes and reduce process variability.

The Advanced Design Technologies Testbed (ADTT) activity at NASA Ames Research Center was initiated to study the strengths and weaknesses of the technologies supporting each of these trends, as well as the overall impact of the combination of these trends on a product development event. Lessons learned and recommendations for future activities are reported.

Motivation

Product development process improvement has become a pervasive theme. The motivations are long-standing and fundamental, as the communities that produce better products can, in general, enjoy a higher standard of living. There are, however, at least four specific reasons for the current intense interest in product development process improvement:

International Free Market

With the emergence of a worldwide free market supported by an international information-sharing system (e.g., the World Wide Web) and an effective international transportation system, an organization can no longer rely solely on locality, political boundaries, or marketing to maintain market shares. Product effectiveness is paramount, and to succeed, a product must be among the best in the world.

Workforce Challenges

Knowledge Reuse

Knowledge from past development events can often be of use in new product development activities. In the aerospace community, however, major design events occur infrequently. As a result, we are at times trying to reuse expertise attained 10, 20, or even 30 years ago. A timely example is the current interest in Earth reentry aerothermodynamics, an area last studied intensely many decades ago during the Space Transportation System, Apollo, Gemini and even Mercury programs.

Rapid Technology Refresh
Driven in part by the pace of the computer industry, engineering tools are becoming more complex and changing faster. For example, CAD systems are "better" than drafting boards, but it typically requires from 1-2 years to learn to use an advanced CAD system, and the systems are typically updated every 6-12 months. Because of the array of computer-based tools that a typical designer uses, the designer is challenged to just maintain proficiency with the tools of the trade.

Product Development Environment Dynamics
In even the simplest of product development events, one can easily imagine on the order of 100 interacting elements, including, for example, the various tools, facilities, members of the team, design requirements, and the design itself. In typical modern aerospace design events, one can easily imagine at least 1000 or even 10,000 elements in a design event. In the future, as we strive for even better products, one can easily imagine the number of distinct interacting elements increasing at least an additional order of magnitude.

Given N elements in a design event, one can compute from basic permutation mathematics (Ref. 2) the maximum possible number of element-to-element interactions:

$$\text{Interactions} \equiv \frac{N!}{2(N-2)!}$$

If one element of a design changes, it is possible that this change will alter some element-to-element interactions:

$$\text{Altered Interactions} = \frac{\alpha N!}{2(N-2)!}$$

The degree of influence of the change is characterized by α, where $0 \le \alpha \le 1$. If ΔN objects change per design event, the total number of altered element-to-element interactions is:

$$\text{Total Altered Interactions} = \sum_{\Delta n=1}^{\Delta N} \frac{\alpha N!}{2(N-2)!}$$

As an example, suppose the following:

N	Elements in design	6	
ΔN	Element changes	2	(α_1=0.80 and α_2=0.40)

The total altered element-to-element interactions are 18. Even in a very simple design domain (just 6 objects), with just a few changes in the included objects (2), one can encounter many changes in the interactions between pairs of objects (18). The magnitude of the product development event and the dynamics of the element-to-element interactions pose severe obstacles to effective product development processes.

Engineering Challenges
We have before us challenging engineering goals. Representative are NASA's Aeronautics and Space Transportation Enterprise technology goals (Ref. 1) that, for example, call for an order-of-magnitude reduction in the cost of space transportation within ten years and a 50% reduction in the cost of air travel within twenty years. In many cases, we have already extracted most of what can be obtained from single discipline or single sub-system refinement and optimization. We have also achieved significant, if loosely coupled, synergy between the various disciplines and sub-systems, and exploited the upstream use of manufacturing and operational considerations. Achieving these new goals is all the more challenging because of past engineering accomplishments.

Response

In response to these product development challenges, a range of technologies are being developed:

- Data creation (e.g., numerical simulation software)
- Data use/management (e.g., databases)
- Knowledge extraction (e.g., visualization, feature extraction)
- Process creation/management (e.g., design systems)

These tools are being developed by at least three types of organizations: non-profit academic and government organizations, manufacturing companies, and software firms.

This diversity of technologies and sources is exciting, but overwhelming. Within a manufacturing company, the focus is on improving the product development process with the minimum expenditure. Typically, this involves the procurement and integration of a subset of these tools. The challenge is to select, from the myriad of technologies and sources, a set of tools that together results in a cost-effective improvement of the product development process.

There are at least three specific obstacles to these efforts:

Nomenclature:

No standard vocabulary exists for describing the capabilities of many of the tools. It is therefore possible for two individuals to discuss the capabilities of a particular piece of software at length and still walk away from the discussion with completely different views of the software's capabilities. For example, consider the range of meanings of the following commonly used words:

- Automated
- Object-Oriented
- Seamless/Plug and Play
- Machine-Independent/Cross-Platform
- Multi-Disciplinary Analysis/Coupling

Both a simple spread sheet with analytic closed form functions and a system integrating CAD, Navier-Stokes fluid simulations and finite-element structural analysis are described by these commonly used words, but the two systems are of fundamentally different character and capabilities.

Social/Economic Pressures:

Software often cannot meet all of the customer's functional requirements. Due to the lack of a precise nomenclature, the customer frequently discovers these limitations only after purchasing and attempting to use the software. At this point, it is often prudent for the customer to invest further in the software rather than begin anew with another product. As a result, the software representative does not have an overwhelming motivation to acknowledge all functional limitations before purchase.

Discontinuous Migration Functionality and Costs:

New types of software are often prototyped by an academic or government organization. This software can be an effective first step into a new domain. For example, PLOT3D (Ref. 3), developed at NASA Ames Research Center, was one of the very first computational fluid dynamic visualization software systems and, in large part, paved the way for the current computational fluid dynamics visualization industry.

Such freeware is often unsupported, forcing the eventually transition to supported software. If the path of internal software is chosen there is usually a second transition to commercial software. Software transition may also be required when a particular vendor has fallen behind another vendor. Often the jumps between these different levels or suppliers of software can be costly in terms of both software licensing and reduced productivity. A representative history of software functionality versus time is presented in Figure 1.

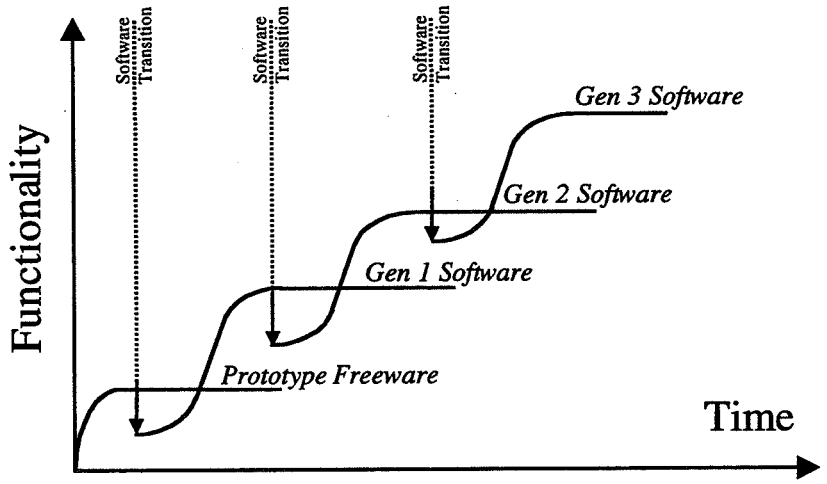


Fig. 1 Software functionality and cost with time.

For an organization that is attempting to improve its product development process, the lack of a precise nomenclature to define software capabilities, the social/economic pressures that can preclude up-front acknowledgment of software limitations, and the need to not infrequently upgrade the various elements of a design system all make the management of the typical aerospace engineering software system a continuous challenge,

ADTT

Given these challenges to the aerospace community, the Advanced Design Technologies Testbed (ADTT) was initiated with support from NASA's IT Base and HPCC programs as well as close collaboration with NASA's AST program. The objectives of this activity are four fold:

- Facilitate improvements in representative and important aerospace design processes.
- Provide a pragmatic proving ground for many of the most promising product development process tools and technologies.
- As required, develop bridging tools and technologies to facilitate integrated solutions.
- Disseminate the results of lessons learned to the aerospace and information technology communities

Context

Since the early 1970's, NASA Ames Research Center has had a major and sustained effort to improve the utility of computational physics tools and, more generally, computational software and hardware systems in the development of aerospace products. From this effort have grown innovative computational physics codes such as OVERFLOW (Ref. 4) and TIGER (Ref. 5), advanced computational facilities such as NAS, and ongoing focused programs such as HPCCP CAS. All of these are focused on exploiting advances in computer-based technologies to improve the product development processes.

In the early 1990's, NASA Ames Research Center re-invigorated a long-term interest in improving the effectiveness of experimental facilities within the product development process. This has resulted in the development of innovative instrumentation technologies (such as pressure sensitive paint) and improved operational processes, along with efforts to create a synergy between the experimental and computational elements of the design process. The IofNEWT (Ref. 6) activity facilitated the comparison of experimental and computational data during the wind tunnel test. The intent was to improve the interpretation and quality of the data, and identify fruitful directions for

further wind tunnel runs during a tunnel entry. Ensuing efforts, such as DARWIN (Ref. 7), have extended this approach to allow robust and secure real-time remote access to experimental data.

Initial Design Domain

All of the above efforts have, for the most part, been successful. The initial objective of the ADTT effort is to build the next natural evolution from these past successes. The current focus of the ADTT activity is to streamline the re-design of elements of a wind-tunnel model and the rapid sharing of that new design, allowing the manufacturing and testing of the new model elements during a wind tunnel test. If successful, this will allow a very rapid and synergistic use of computational and experimental technologies to rapidly evolve improved designs.

Engineering Requirements

A system is required that integrates:

- configuration and geometry manipulation and management;
- aerodynamic, structural, and system levels analysis; and
- data analysis and presentation.

The system must also enable access to other systems that capture and manage the experimental data and support rapid model manufacturing. Though not all desired functions are in place, such a system has been developed and tested, with the resulting new wind-tunnel model elements scheduled for testing in the near future.

Technology Evaluation Requirements

In order to adequately test and evaluate technologies for applicability to product development process improvement, they must be applied within realistic events. This testbed environment must allow the easy insertion of new tools and technologies. To be a representative testbed, it is important that the system can also simultaneously support access by individual software tool experts as well as overall product development experts, a requirement that requires an adaptive interface. A mature testbed environment should maintain baseline analysis and design functions so that all new technologies may be tested against baseline capabilities and metrics.

Existing Framework Technologies

A survey of potentially applicable frameworks showed that there are many products advocated as the generic solution to all product development processes, an example being Product Data Management (PDM) systems. Most existing frameworks are applicable to a well-structured process that can be managed and tracked, invoking relevant applications in a lock-step fashion. Existing frameworks generally support distributed users accessing centralized applications (e.g., CAD drawing manipulation or logistics management systems) and moderate size data sets. After many product reviews and even a few costly prototype activities, we concluded that for a dynamic, computationally-intensive (e.g., high-fidelity, physics-based computational analyses), and distributed product development event no existing framework has the required cross-platform capability, flexibility, and scalability.

Framework Approach

A ground-up approach was called for in which we could develop the flexibility needed to handle the changing requirements and environment while providing a stable schema for data storage and access. An underlying UNIX environment was able to provide a coherent framework for applications invocation and data sharing across multiple users. One of the goals of the system, however, was to distance the user from the specifics of the environment - directory structures, file names, UNIX commands, etc., and from the specifics of changes made to applications that might be reflected in launch commands or namelists. A graphical user interface and underlying process management capability was required to make this structure accessible to the user. The ADTT interface and architecture are designed to:

- Simplify user access to design tools and analysis applications.
  GUIs and underlying rule-based input advisory allow the user to interact with all applications at the parametric input level, rather than requiring expertise in the specifics of the applications or their invocations.
- Provide user guidance through the steps of the design process.
  Rule-based process management ensures the user is informed of options available given the state of the design.
- Provide the ability to generate large numbers of designs or analyses from a single input.
  Underlying application launch and post-processing capabilities allow designers to launch virtually unlimited numbers of jobs to the CAD systems, grid generation programs, or flow solvers. Rule-based systems build and coordinate input files, perform job routing and partitioning, and postprocessing and data storage.
- Provide job and design process status information to the user.
  At-a-glance information on the availability of requested information and status of analyses queued or running on any platform.
- Build a coherent data model for the organization and rapid access of analytical data.
  Store data files in unique and meaningful directory structures based on a data object model of the application.
- Allow remote access and collaboration.
  Ensure security for proprietary data and provide support of remote access to distributed applications within a dispersed heterogeneous computational environment. Through a secure Web site and COTS collaboration tools, ADTT supports access to design data and visualization capabilities.
- Provide flexibility in the addition and/or modification of applications within the system.
  GUIs, input files, data objects, and directory structures are built from text files. When a parameter changes, the system responds automatically.

Application Experience

The ADTT system is currently being used in conjunction with the wind tunnel test of a transport aircraft configuration. The standard high lift system consists of a leading edge slat and two trailing edge elements: a small vane flap and a larger main flap. Production cost could be reduced if a single trailing edge flap system of sufficient performance was developed. The ADTT system has been used to develop a single element design that does not compromise short field performance.

The redesign process began with the evaluation of several different single element flaps. Midway through this process, the decision was made to change the flap deployment kinematics due to system-level considerations. The flexible design of the ADTT simplified the tasks of regenerating the GUIs used to enter the appropriate deployment parameters and reconfiguring the CAD system to accommodate the new deployment schedules. A total of 36 designs were evaluated over a period of 2 weeks. The geometry manipulation, creation of input files, launching of codes, storage, and post-processing of the over 2200 datasets would have been extremely difficult to organize and accomplish without a system such as ADTT.

With the number of design variables involved and the overwhelming volume of data produced, it was imperative to develop a comprehensive method for comparing designs. While plots of aerodynamic coefficients versus angle of attack and surface pressure distributions were available, the number of configurations investigated quickly made it difficult to determine optimal designs using these traditional data presentation methods. A specialized interpolation process was developed to provide a flexible means of interrogating the many datasets subject to a variety of constraints. The interpolated results were then provided in a series of contour maps. Incorporating this new capability directly into the ADTT system allowed the developers to take advantage of the organized storage of the data and provide a consistent look and feel to the post processing. This interpolation procedure and the data presentation format were essential for

extracting the relevant design information from the large design space. The successful last-minute addition of this crucial data presentation format attests to the flexible design of the ADTT system.

Once the best design was determined, the CAD model used for the analyses was transferred to the manufacturing shop, where tool paths were generated and the corresponding part was machined. The customer, using an in-house technique, also independently confirmed the ADTT-developed predicted performance. This new flap is scheduled for evaluation as part of a current wind tunnel test. Once the experimental results from this design are available, the design process will be repeated, with the goal of producing a further improved design that will be manufactured and tested before the completion of the test program.

Lessons Learned

The initial application described here has demonstrated that an ADTT-like system does improve the speed and effectiveness of re-design and design trade-off studies. One important key to success with any aerospace system is to supply the required analyses, process, and data management capabilities within a framework that allows rapid, if not automated, adaptation to change. However, experience shows that the key to success is a very strong partnering between the aerospace system software designers and developers, the partners developing and maintaining the computational infrastructure, the partners developing intelligent systems technologies, and, of critical importance, the designers using and evaluating the system.

In the future, as the amount of generated data grows, we must find better ways to portray to the designers the state of both their own and their colleagues' designs, as well as allow the rapid comparison of designs, acknowledging that each designer may use different design metrics. We must work with the developers of advanced computational infrastructures to enable the scheduling of distributed computational systems (compute nodes, data storage systems, networks, and more) to optimize the productivity of the designers. For example, the computational infrastructure must be able to rapidly disperse and execute large numbers of jobs and must provide fast and reliable access to the results. To provide improved process advisory and streamlining to the users, using artificial intelligence support technologies, we must continue to develop tools and methods that facilitate the effective capture and organization of detailed design process and rationale knowledge.

Conclusions

In the ongoing effort to develop computational systems, such as ADTT, to support the product development process areas that require further research are continuously identified. Work to date suggests that significant development in the areas of data presentation, computational infrastructure, process assistance, and rationale capture and use are required.

The efficient extraction of information is of increasing importance as more high fidelity data becomes available. To be useful, the information must be: 1) easily accessible in near real time, 2) correlated against similar results in other data repositories, and 3) coalesced and presented in visual forms that are meaningful to the designers. Synergistic solutions to these three challenges must be constructed. Components of future systems for design space mapping will likely include:

- Archival database systems capable of cataloging thousands of high-fidelity simulations while providing efficient "at-your-fingertips" retrieval of individual datasets.
- Metadata architecture that enables the correlation of both computational and experimental information stored in heterogeneous databases.
- Data mining techniques that allow the perusal, analysis, correlation, identification, and summary extraction and presentation of design characteristics.
- New data presentations and filters that prevent designers from being overwhelmed by the sheer volume of information that results from a thorough examination of a multi-parameter design space.

Successful data mining methodologies for design space maps will need to be both applicable to broad classes of projects and easily customizable for specific designs. Techniques borrowed from disciplines such as digital image and signal processing may be useful.

The ability to launch virtually unlimited numbers of analyses also presents a new set of computational challenges. If we are to truly reduce the design cycle time, it is not enough to increase the capacity of job queuing - the bottleneck then becomes the availability of the resources rather than the time required for job setup. The ultimate goal is not to get the jobs into the queue faster but rather to make results available to the user more quickly. Reductions in run time through more intelligent job monitoring and intervention may provide a partial solution to this problem. However, the real breakthrough will be realized through access to a truly distributed, powerful computing environment (e.g., Computational Grids, Ref. 8). Revolutionary ways to organize and store the unprecedented flood of data that will be generated by these jobs and that must be rapidly accessible within the design environment are also required. Fast, distributed storage, data compression and rapid search techniques are among the relevant technologies for exploration.

Expert process assistance for the designer will provide the benefits of reduced design time with higher quality results, as well as historical reference for future designers and design applications. To accomplish this requires in-depth understanding of both the actual design process and, more importantly, the rationale behind the designers' decisions. To this end, technologies such as context understanding, mapping and recall, and other knowledge and rationale capture technologies are being developed as integral system components. When there is a more complete understanding of the principles that underly the design process, we can apply this knowledge, through planning and scheduling techniques, agent technology, reasoning techniques and other applicable technologies to assist the designer in further streamlining the process.

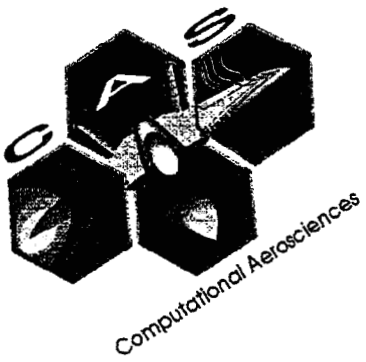
Acknowledgments

These activities have been made possible by the NASA IT Base, HPCC, and AST Programs, as well as the Aeronautics Design/Test Environment (ADTE) Project. Many engineers from Boeing and NASA Ames' Aeronautics, Information Systems, and Research and Development Directorates have worked together to explore the potentials of integrated design systems as reviewed in this paper. This introductory paper will be followed by future papers that will discuss in substantial details the work only outlined here.

References

1. NASA Office of Aeronautics and Space Transportation Goals: http://www.hq.nasa.gov/office/aero/oastthp/brochure/brochure.htm.
2. Kreyszig, E., Advanced Engineering Mathematics, Third Edition, John Wiley and Sons, Inc. New York, 1972, pg. 719.
3. Buning, P. G. and Steger, J. L., "Graphics and Flow Visualization in Computational Fluid Dynamics," AIAA-85-1507-CP, July 1985.
4. Renze, K. J., Buning, P. G., and Rajagopalan, R. G., "A Comparative Study of Turbulence Models for Overset Grids," AIAA-92-0437, January 1992.
5. Melton, J. E., Berger, M. J., Aftosmis, M. J., and Wong, M. D., "3D Applications of a Cartesian Grid Euler Method," AIAA Paper 95-0853, January 1995.
6. Koga, D. J., Schreiner, J. A., Buning, P. G., Gilbaugh, B. L., and George, M. W., "Integration of Numerical and Experimental Wind Tunnels (IofNEWT) and Remote Access Wind Tunnel (RAWT) Programs of NASA," AIAA-96-2248, June 1996.
7. Koga, D. J., Korsmeyer, D. J., and Schreiner, J. A., "DARWIN Information System of NASA—An Introduction," AIAA-96-2249, June 1996.
8. Foster, I. and Kesselman, C., "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publishers, San Francisco, 1998.

OMIT THIS PAGE



Session 9:

Numerical Optimization

Aerodynamic Shape Optimization Using A Combined Distributed/Shared Memory Paradigm

Samson Cheung
MRJ Technology Solutions

Terry Holst
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

Current parallel computational approaches involve distributed and shared memory paradigms. In the distributed memory paradigm, each processor has its own independent memory. Message passing typically uses a function library such as MPI or PVM. In the shared memory paradigm, such as that used on the SGI Origin 2000 machine, compiler directives are used to instruct the compiler to schedule multiple threads to perform calculations. In this paradigm, it must be assured that processors (threads) do not simultaneously access regions of memory in such a way that errors would occur. This paper utilizes the latest version of the SGI MPI function library to combine the two parallelization paradigms to perform aerodynamic shape optimization of a generic wing/body.

Optimizer

The *//*OWA parallel numerical optimizer,[1] which employs an unconstrained quasi-Newton method using a self-scaling BFGS algorithm,[2] is utilized in this study. Like all gradient optimizer algorithms, the present method requires two types of inputs, design-space sensitivity derivatives and line search information. The sensitivity derivatives tell the optimizer the impact of design variable changes on the objective function. The search direction information, obtained after each set of sensitivity derivatives is computed, helps the optimizer set step sizes, i.e., how much should each design variable be changed. In the present approach, derivatives of the objective function with respect to the design variables, are obtained using the finite-difference method. With this approach for computing design space sensitivity derivatives, constraints are easily added to the optimization process. For all computational results presented herein the following objective function is used:

$$OBJ = \frac{C_{D,i} + C_{D,V}}{C_L} + (C_L - 0.409)^2$$

In the above equation $C_{D,i}$ is the inviscid pressure drag coefficient computed from the CFD flow solver, $C_{D,V}$ is the viscous drag coefficient set equal to 0.0250 for all cases, and C_L is the inviscid lift coefficient also computed from the CFD flow solver. Note the penalty-type constraint on the lift coefficient built into this objective function definition that forces the lift coefficient to be near 0.409 (the unoptimized geometry's lift coefficient). During an optimization iteration, the distributed and/or distributed/shared memory paradigm is used to perform the function evaluations that are necessary for these derivative computations as well as the line search process used to find the optimal design point. Each processor is assigned to a particular design variable (see Fig. 1).

CFD Flow Solver

The TOPS[3,4] CFD code is utilized in this study to perform all objective function evaluations. This code is a three-dimensional full potential solver that utilizes a chimera zonal grid approach for handling complex geometries. It has the capability of producing complete numerical solutions about wing/body configurations (~200K grid points) in 1-2 min on a single SGI Origin 2000 processor. Each run consists of surface and volume grid generation; wing/fuselage line of intersection computation; chimera hole cutting, donor cell search, and interpolation coefficient computation; and, finally, the flow solver step. Each of these steps is automatically coupled and executed without user intervention. This makes the *//*OWA/TOPS coupling easy and efficient.

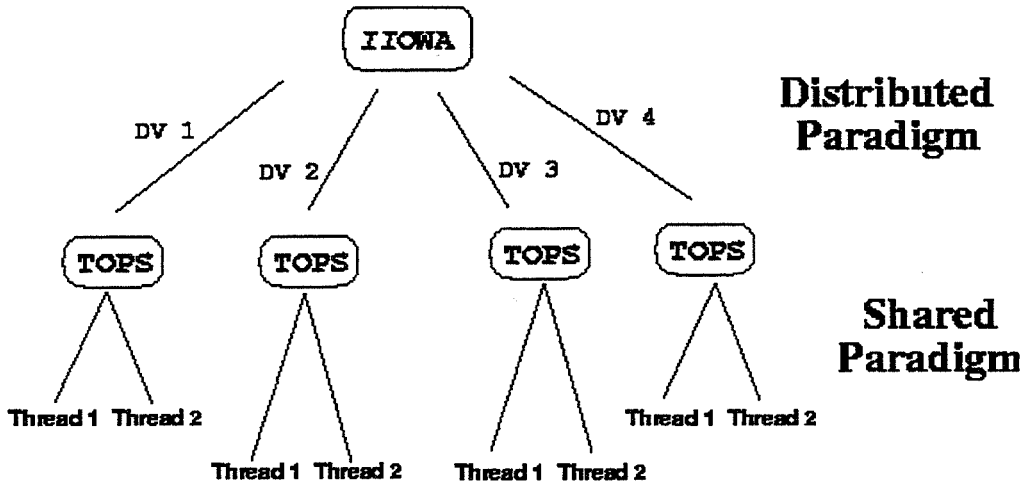


Fig. 1 Schematic of the aerodynamic optimization code system. The upper level (optimizer) uses the distributed memory paradigm and the lower level (CFD Code) uses the shared memory paradigm. Note: DV stands for design variable.

Additional improvements in wall-clock time are obtained using the shared memory paradigm for flow solver implementation. To efficiently parallelize the code using this paradigm, computationally intensive regions of code are selected for the creation of multiple threads (see Fig. 1). Care must be taken in this process so that the regions of code selected have the necessary data independence. Once these regions are identified, compiler directives, such as C$DOACROSS, are placed in the code to implement the multithread operations at FORTRAN do loops. A simple example is given below:

```
C$DOACROSS LOCAL(J,I,PX,PY,PZ)
      DO J=2,NJM
       DO I=2,NIM
          PX       = (PHI(I+1,J ,K+1)-PHI(I-1,J ,K+1)+PHI(I+1,J ,K)-PHI(I-1,J ,K))*0.25
          PY       = (PHI(I ,J+1,K+1)-PHI(I ,J-1,K+1)+PHI(I ,J+1,K)-PHI(I ,J-1,K))*0.25
          PZ       = PHI(I,J,K+1)-PHI(I,J,K)
          FKM(I,J) = RK(I,J,K)*(AZ3(I,J,K)*PZ+AZ5(I,J,K)*PX+AZ6(I,J,K)*PY)
       ENDDO
      ENDDO
```

The SGI FORTRAN compiler provides an option (-pfa) to perform Power Fortran Accelerator. This will conservatively multi-thread all the do loops in the compiled routines. Although it is very convenient, it may not be the best way to perform multi-thread parallelization. This is because some of the do loops being parallelized may not have enough work for multiple processors to share, especially when the additional overhead of multi-threaded parallelism is considered. Therefore, a more computationally efficient, albeit more tedious, approach is to add compiler directives, such as C$DOCROSS, by hand. Only computationally intensive do loops are executed in parallel. In the present implementation (as a test of this idea), the grid generation, chimera (hole-cutting, donor-cell search, and interpolation) routines and about one-third of the flow solver are not parallelized. The routines utilizing the C$DOCROSS directive are compiled with the -mp option; and before execution, the environment variable MP_SET_NUMTHREADS is set to the number of desired threads (processors).

Results

The RAE wing/body configuration is used as the initial condition in this optimization study. This geometry involves a symmetric wing mid-mounted onto an ogive-cylinder fuselage. A three-zone grid with a total of

210K grid points is employed to calculate each flow field (see Fig. 2). There are two inner grid zones, a C-H topology grid around the wing and a C-O topology grid around the fuselage. These two inner grids are generated using the HYPGEN grid generation code.[5] Once the wing grid is generated, the grid surface lying next to the wing/fuselage juncture is interpolated onto the fuselage surface using bi-cubic-spline interpolation. Thus, flow tangency boundary conditions are implemented in the wing grid along two grid surfaces, the wing surface and the fuselage surface. These two inner grid zones are connected to the freestream using the third grid zone, an outer, sheared-stretched, Cartesian-like grid. The flow conditions chosen for this study are $M_\infty = 0.84$ and $\alpha = 4°$. Once the design optimization is complete the initial and final geometries are rerun with a finer grid involving about 500K points. These fine grid results are then used for analysis and plotting.

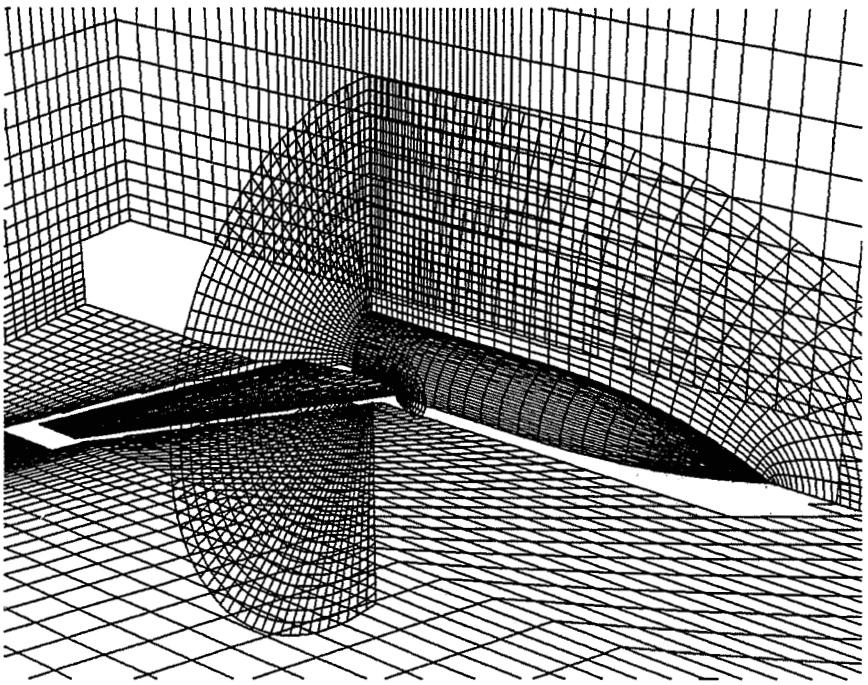


Fig. 2 Three-Zone grid about the RAE Wing A + cylindrical body B2

Case 1 (10 design variable, distributed case)

In the first case presented, ten design variables are used to discretize the design space. Geometric changes for only the upper wing surface are considered and are implemented using four thickness design variables and one twist design variable at each of two spanwise stations (root and tip). The thickness design variables are distributed on the upper surface at chordwise stations ranging between 0.15 and 0.65. The upper wing surface thickness distribution between 0.06 and 0.75 is held fixed during the design optimization process. This simple definition of the design space is chosen because the emphasis in this study is on implementation/parallelization efficiency, and not aerodynamic efficiency. The first optimization was performed using ten Origin 2000 CPUs (via MPI).

The total number of function calls (i.e., flow solver solutions from TOPS) was 385. The total wall clock time of this optimization run was about 66 mins, which is a speedup over the totally serial case of 8.1. There are two primary reasons this number is not closer to the ideal value of ten. First, at the beginning of the optimization iteration a single solution is required, which requires only a single processor. Second, during the line search, if a local minimum is detected, additional function evaluations are required to enrich the definition of the search direction. These extra line search enrichment solutions, in the present implementation, are performed serially, and thus, require only a single processor. A parallel implementation of the line search enrichment step is theoretically possible, but has not been implemented to date.

The result of the Case 1 optimization process, which required 18 optimization iterations, is about a 17% decrease in the objective function as can be seen from Fig. 3. The results of this optimization on the wing surface pressures and the wing shape are displayed in Figs. 4 and 5. Figure 4 shows the surface pressures at four wing stations ranging from 16 to 83% of the semi-span. The y/b=0.16 result (Fig. 4a) corresponds to the wing/fuselage juncture. Note that both the baseline (unoptimized) and optimized surface pressures are included in each of these plots. Generally, the transonic shock strength has been weakened at each station, while the lift has remained approximately constant. Figure 5 shows the wing surface shapes at the same wing semi-span stations as those displayed in Fig. 4. Note that the z/c coordinate has been multiplied by a factor of ten to better see the shape modifications chosen by the optimizer. Keep in mind that (except for twist) the airfoil shape on the entire lower surface and upper surface forward of 0.06 and aft of 0.75 was frozen throughout the optimization process. Generally, the airfoil thickness has been increased at the tip, decreased at the root, and a lift-neutral, "wash-out-type" twist distribution has been introduced.

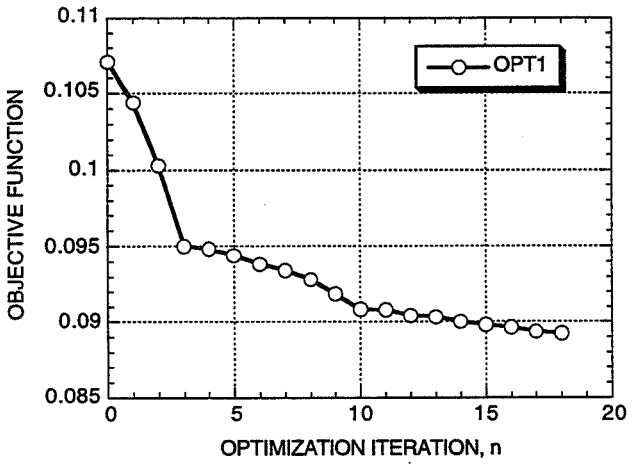


Fig. 3. Objective function versus optimization iteration for case 1 involving ten design variables and ten processors.

Case 2 (10 design variable, distributed/shared case)

The aerodynamic optimization for the next case is the same as the previous case, but the computation involves 20 processors and a combined distributed/shared implementation as described in the above CFD flow solver section. Each function evaluation is performed using two threads, i.e., MP_SET_NUMTHREADS is set to 2. The results of the numerical optimization, as expected, are identical to the above results for Case 1. The wall clock time is reduced to 43.7 mins. This is a factor of 1.50 speed-up over the previous 10 processor case and represents a speedup over the totally serial case of 12.2. Since only about half of the total code was affected by the compiler directives (its difficult to be precise on this point), this timing result is about as expected.

Case 3 (44 design variable, distributed case)

The aerodynamic optimization for the next case is likewise the same as the previous case, but the design space discretization is much finer involving a total of 44 design variables. Geometric changes for only the upper wing surface are considered and are implemented using ten thickness design variables and one twist design variable at each of four equally-spaced spanwise stations between the root and tip. In addition, a new perturbation grid generation option was utilized for this computation. Before the optimization run initiates, the baseline geometry volume grid for all three grid zones is saved in a file. Then, instead of using a "from scratch" grid generation for each function evaluation, the saved grid is read in and modified using the new perturbed wing surface geometry. The new perturbation grid generation option is about ten times faster than a call to HYPGEN and produces a grid very close to the original grid.

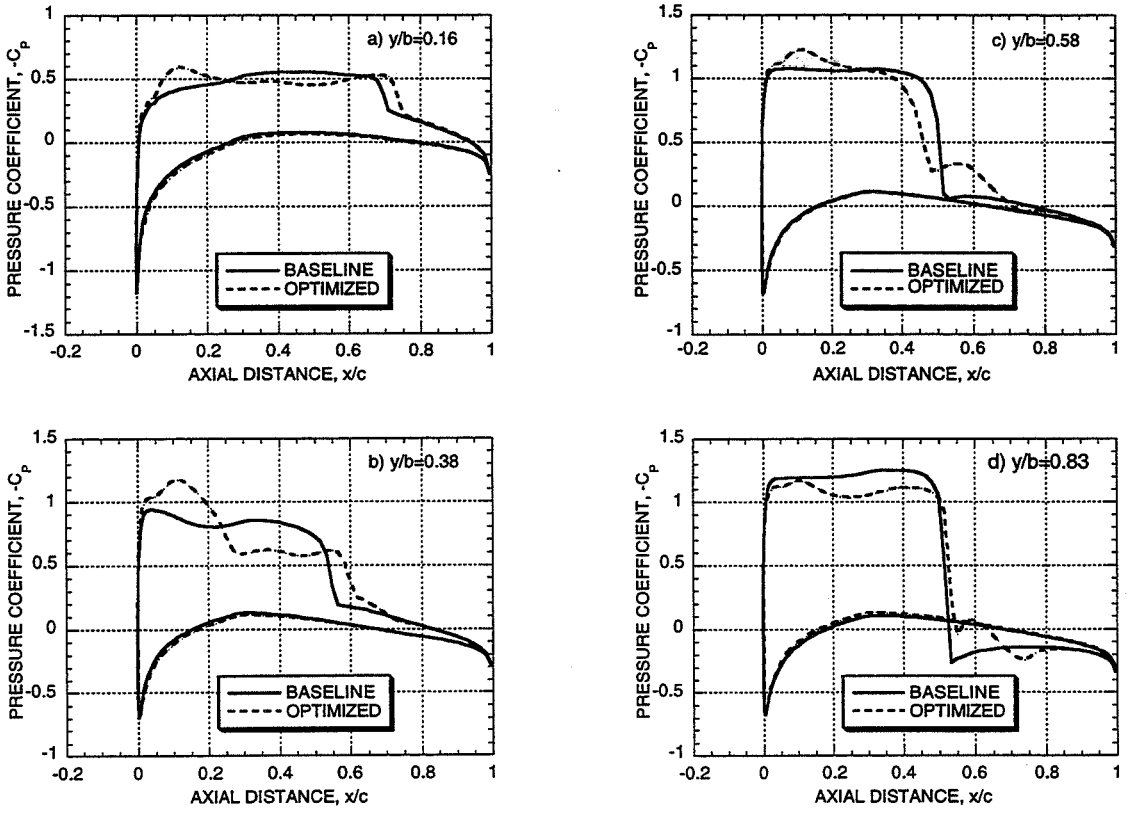


Fig. 4 Pressure coefficient distributions at several spanwise stations showing the effects of the wing shape optimization, ten design variables.

This optimization was performed on 44 processors and required a total of 20 design iterations, 1810 flow solver runs (function evaluations), and 65 mins of cpu time. This corresponds to a speedup over the totally serial case of 36.2 or about 82.3% of ideal. This particular computation required 6 serial function evaluations, i.e., TOPS computations run on a single processor with the other 43 processors sitting idle. These serial solutions (as previously mentioned) correspond to the initial solution and (for this particular case) 5 line search enrichment computations. If these 6 serial computations are removed from consideration, the computational statistics become 1804 function evaluations in about 57.2 mins. This corresponds to a speedup over the totally serial case of 41 or about 93% of ideal. Thus, it can be concluded that the *I*IOWA optimizer overhead for this computation is about 7% of the total computer time.

Conclusions

The TOPS full potential chimera aerodynamic analysis code has been coupled with the *I*IOWA gradient optimizer. The *I*IOWA/TOPS combination has been used to compute optimized wing shapes on a parallel computer using the distributed and distributed/shared paradigms with good success. The distributed cases showed good processing rates with speed-up factors ranging around 80% of ideal. The improvement in speed up for the distributed/shared case was not as good being only about 60%. The lower speed up for the distributed/shared case was due to only partial implementation of the shared paradigm, however, and requires more investigation.

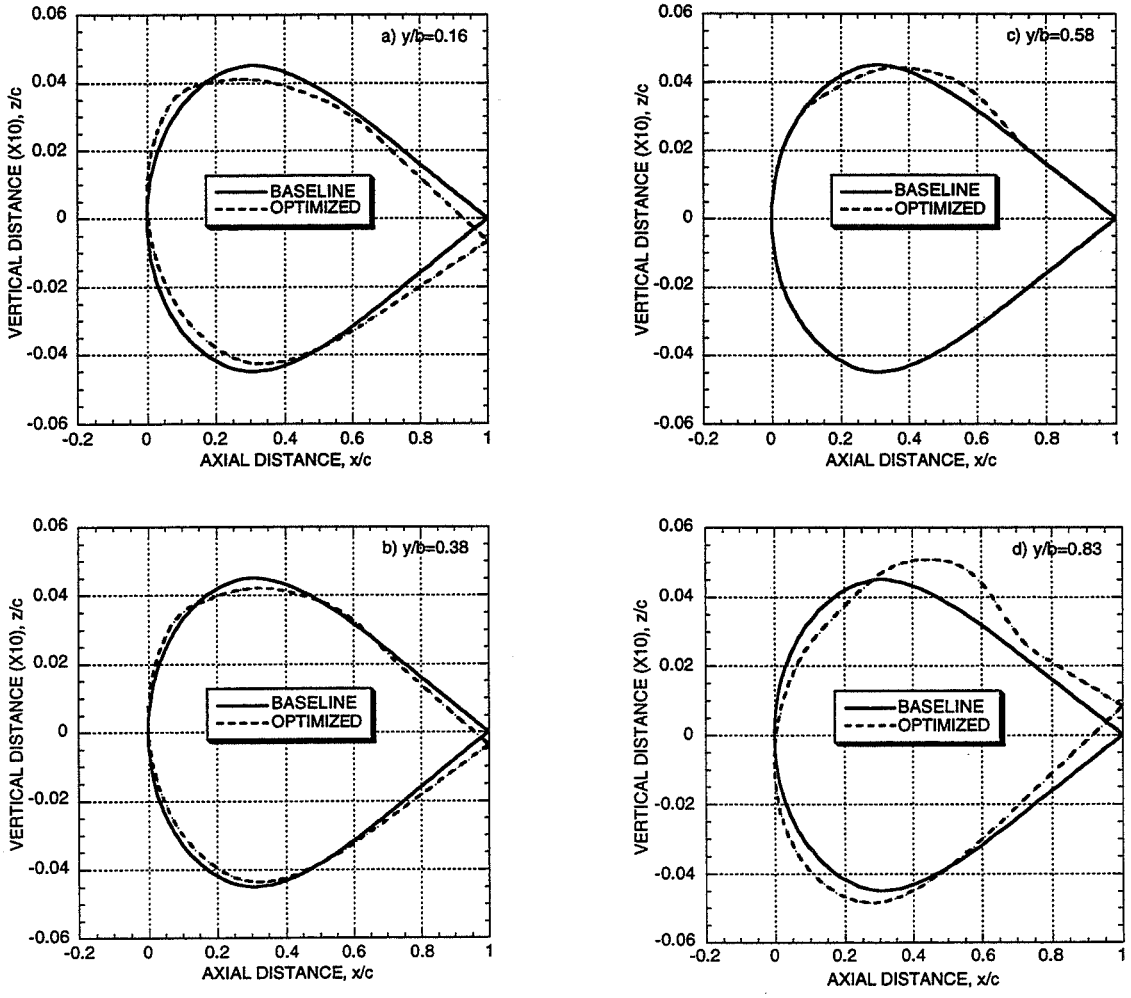


Fig. 5 Wing shape distributions at several spanwise stations showing the effect of optimization on the symmetric baseline wing. Note that the z-coordinate has been magnified by X10.

References

[1]Cheung, S., "Parallel CFD Design on Network-Based Computer," *J. of Aircraft*, Vol. 33, No. 3, May-June 1996.

[2]Luenberger, D., *Linear and Nonlinear Programming*, 2nd Ed, Addison-Wesley, pp. 261-288, 1984.

[3]Holst, T. L., "Full Potential Equation Solutions Using a Chimera Grid Approach," AIAA Paper No. 96-2423, June, 1996.

[4]Holst, T. L., "Multizone Chimera Algorithm for Solving the Full-Potential Equation," J. of Aircraft, Vol. 35, No. 3, May-June 1998, pp. 412-421.

[5]Chan, W. M., Chiu, I., and Buning, P. G., " User's Manual for the HYPGEN Hyperbolic Grid Generator and the HGUI Graphical User Interface," NASA TM 108791, Oct. 1993.

539-02
018274

HIGH-FIDELITY AEROELASTIC ANALYSIS AND AERODYNAMIC OPTIMIZATION OF A SUPERSONIC TRANSPORT

366900

Anthony A. Giunta
Postdoctoral Fellow, National Research Council
NASA Langley Research Center, Mail Stop 139, Hampton, VA 23681-2199, USA
Phone: 757/864-3176, E-mail: a.a.giunta@larc.nasa.gov

6P.

INTRODUCTION

A suite of modular government/commercial off-the-shelf (G/COTS) software packages has been created to perform high-fidelity aeroelastic analysis and aerodynamic optimization of aircraft configurations. While the current status of the software permits multidisciplinary analysis and single-disciplinary optimization, the goal of this research is to develop a high-fidelity multidisciplinary optimization (MDO) capability in which various MDO methods will be examined on realistic aircraft design problems. The existing MPI-based parallel computing capability in some elements of the G/COTS software is a key component in realizing the goal of high-fidelity MDO. In particular, the parallel computing capabilities allow the efficient calculation of sensitivity derivatives needed to perform gradient-based optimization. To demonstrate the utility of this modular G/COTS software approach, an aeroelastic analysis and aerodynamic optimization of a high-speed civil transport (HSCT) are examined.

SOFTWARE AND ANALYSIS MODELS

The software suite includes the following G/COTS elements: the Euler/Navier-Stokes solver CFL3D from NASA Langley (Ref. 1), the aerodynamic grid generator CSCMDO (Coordinate and Sensitivity Calculator for Multi-disciplinary Design Optimization) from NASA Langley (Ref. 2), the finite element analysis/optimization package GENESIS from VMA Engineering, Inc. (Ref. 3), the aerodynamic loads interpolation package FASIT (Fluids and Structures Interface Toolkit) from Georgia Tech and the U.S. Air Force (Refs. 4, 5), and the gradient-based optimizer DOT (Design Optimization Tools) from Vanderplaats Research and Development, Inc (Ref. 6). This collection of G/COTS software is loosely coupled using UNIX scripts and common geometry descriptions such as the PLOT3D format and the NASTRAN Bulk Data format. Currently, the suite is executed serially on a network of UNIX workstations but parallel computing is possible with both CFL3D and GENESIS. Figure 1 shows the arrangement of the software modules when performing aeroelastic analysis and Figure 2 shows the software organization used in aerodynamic optimization. Note that the block entitled "Geometry" denotes the use of an in-house parametric model of the HSCT configuration employing 104 variables (64 planform and shape, 40 internal structure).

The aerodynamic volume grid is created using the data in the parametric HSCT model along with CSCMDO and a baseline volume grid of the initial HSCT configuration. Here, a transfinite interpolation scheme is used in CSCMDO to produce a new volume grid for each iteration of the aeroelastic analysis or aerodynamic optimization. Thus, each new grid is a perturbation of the baseline HSCT volume grid. The aerodynamic volume grid is a two-block C-O mesh with approximately 300,000 grid points and overall dimensions of 121x41x61 (Fig. 3). Using

CFL3D, an inviscid aerodynamic analysis requires 1.3 CPU hours on a Silicon Graphics (SGI) workstation (195 MHz IP30 processor). Subsequent CFL3D analyses make use of the restart capability in the flow solver and require approximately 16 CPU minutes.

The finite element structural model is generated based on the 40 sizing variables. Of these, 26 of the variables are skin panel thickness values (13 panels on each of the upper and lower body surfaces), 12 of the variables are spar cap areas, and two of the variables are rib cap areas. The finite element structural model originally was developed by Balabanov (Ref. 7) and consists of 1130 elements with 1254 degrees of freedom (Fig. 4). Using GENESIS, a single structural analysis of this model requires about 50 CPU seconds on an SGI workstation.

For the aeroelastic analysis process, FASIT is used to calculate the aerodynamic loads from the CFL3D data and to interpolate the loads from the surface of the aerodynamic grid to the surface of the finite element structural model. FASIT is ideally suited for this loosely coupled approach to computational aeroelasticity, as it accepts several widely used file formats employed by computational fluid dynamics solvers and finite element analysis software. FASIT provides a suite of interpolation schemes for transferring the aerodynamic loads from the aerodynamic surface grid to the structural surface grid. The thin plate spline method of Duchon (Ref. 8) was used in this study, as recommended in the FASIT User's Manual (Ref. 5). Note that the interpolation methods in FASIT conserve the total force and moments for all three axes when transferring loads from the aerodynamic grid to the structural grid. The loads transferred to the structural grid are written in the NASTRAN Bulk Data format. Since this NASTRAN format is compatible with the GENESIS input format, no translation is needed between FASIT and GENESIS. The computational expense of running FASIT is approximately 10 CPU seconds.

The structural deflections in an aeroelastic analysis are transferred back to the aerodynamic surface and volume grids using some in-house software that calculates the change in each of the 64 HSCT external shape parameters. With an updated parametric description of the HSCT, a new aerodynamic volume grid is generated using CSCMDO, and the aeroelastic analysis process starts another iteration. A similar procedure is used in the aerodynamic optimization where the optimizer specifies changes in the variables and a new aerodynamic volume grid is generated based on the updated HSCT geometry.

AEROELASTIC ANALYSIS AND AERODYNAMIC OPTIMIZATION

The static aeroelastic analysis case is performed at 1.0g Mach 2.4 cruise conditions for a fixed angle-of-attack of 3.5 degrees (with respect to the fuselage centerline). A constant factor under-relaxation method converges the aeroelastic analysis in six iterations and 2.7 CPU hours. Without relaxation, the aeroelastic analysis procedure converges slowly and was terminated after 19 iterations (Fig. 5). The wing deflection is shown in Figure 6 where airfoils at the wing root, leading-edge break, and wing tip are shown for the initial undeformed shape and the final deformed shape. The elastic structural model results in a lift reduction of 5.7 percent as compared to the rigid model. Note that no structural sizing is performed during the aeroelastic analysis. However, prior to the aeroelastic analysis the structural elements are sized to meet von Mises yield stress criteria and local buckling constraints for a 2.5g pull-up maneuver at Mach 2.4.

The aerodynamic optimization problem employs 10 of the 64 parameters that define the wing planform and shape. The planform variables are the inboard and outboard leading edge sweep

angles (Λ_I, Λ_O), chord lengths at the leading edge break and wing tip (C_{break}, C_{tip}), the spanwise location of the leading edge break (Y_{break}), and the semispan (Y_{tip}). The airfoil shape variables are the thickness-to-chord ratio (t/c) at the leading edge break and the wing tip ($(t/c)_{break}$, $(t/c)_{tip}$), the leading edge radius parameter (R_p), and the chordwise location of maximum thickness on the wing (C_{max}). Camber and twist parameters are held fixed during this optimization case. The optimization problem is formulated to minimize drag subject to two constraints. One constraint requires the lift to meet or exceed 210,000 lb, and the other constraint specifies a minimum chord length at the leading edge break to maintain a reasonable planform geometry. Using the SQP optimizer in DOT, the aerodynamic optimization yields a drag reduction of about three counts (-5.8 percent) while meeting both of the constraints. The computational cost of this optimization is approximately 16 CPU hours on an SGI workstation. The initial and optimal variable values are listed in Table 1 and the corresponding wing shape improvements are shown in Figures 7 and 8. Note that a single processor workstation was used for the optimization and that sensitivity derivatives were computed using forward finite difference methods. Thus, these results represent a worst-case evaluation of computational costs. Future aerodynamic optimization and MDO investigations will benefit substantially from the use of the parallel version of CFL3D to efficiently compute the needed sensitivity derivatives.

CONCLUSION

In progress toward the goal of high-fidelity MDO for aircraft configurations, an initial capability has been developed to perform multidisciplinary analysis for static aeroelastic problems. The solution of the coupled aerodynamic-structural system is an important precursor to performing aircraft MDO. In addition to the multidisciplinary analysis capability, a single-disciplinary aerodynamic optimization capability was developed to demonstrate the methods needed to interface high-fidelity analysis software and optimization software. The use of modular G/COTS software facilitates the exchange or replacement of analysis codes according to the needs of the user.

Future use of the parallel computing capabilities in CFL3D and GENESIS will allow for the efficient calculation of sensitivity derivatives needed in aircraft MDO studies. These coupled system MDO cases will involve the combined sizing and shape optimization of an HSCT configuration subject to aerodynamic, structural, and performance constraints. Several methods for solving MDO problems involving coupled systems will be investigated including those based on Global Sensitivity Equations as well as Collaborative Optimization techniques.

REFERENCES

1. Taylor, A. C., III, Oloso, A., and Newman, J. C., III, "CFL3D.ADII (Version 2.0): An Efficient, Accurate, General-Purpose Code for Flow Shape-Sensitivity Analysis," AIAA Paper 97-2204, 1997.
2. Bischof, C. H., Jones, W. T., Mauer, A., and Samareh-Abolhassani, J., "Application of Automatic Differentiation to 3-D Volume Grid Generation Software," *Proceedings of the 1995 ASME Int. Mech. Engr. Congress and Exposition*, volume FED 232, San Francisco, CA, pp. 17-22, 1995.
3. GENESIS User's Manual Version 4.0, Vanderplaats, Miura and Associates, Inc., Colorado Springs, CO., 1997.

4. Smith, M. J., Hodges, D., and Cesnik, C., "An Evaluation of Computational Algorithms to Interface Between CFD and CSD Methodologies," Flight Dynamics Directorate, Wright Laboratory, Wright-Patterson Air Force Base, OH, Report WL-TR-96-3055, 1995.
5. Smith, M. J., Hodges, D., and Cesnik, C., "Fluids and Structures Interface Toolkit (FASIT), Version 1.0," Georgia Tech Research Institute, Atlanta, GA, Report GTRI A-9812-200, 1996.
6. DOT User's Manual Version 4.20, Vanderplaats Research and Development, Inc., Colorado Springs, CO, 1995.
7. Balabanov, V. O., *Development of Approximations for HSCT Wing Bending Material Weight Using Response Surface Methodology*, Ph.D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1997.
8. Duchon, J., "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," Springer-Verlag, Berlin, eds. W. Schempp and K. Zeller, pp. 85-100, 1977.

Table 1. Initial and optimal values for the 10 variable HSCT aerodynamic optimization case.

Variable	**Initial Value**	**Optimal Value**
Λ_I	74.0 °	75.0 °
Λ_O	45.0 °	45.0 °
C_{break}	42.4 ft	44.7 ft
C_{tip}	9.3 ft	10.0 ft
Y_{break}	28.6 ft	28.1 ft
Y_{tip}	67.3 ft	70.0 ft
$(t/c)_{break}$	2.15 %	1.50 %
$(t/c)_{tip}$	2.36 %	1.50 %
R_p (nondimensional)	4.5	3.5
C_{max}	45.0 %	46.0 %
Results	**Initial Value**	**Optimal Value**
$C_D \times 10^4$	52.74	49.70
Lift	202,000 lb	210,000 lb
Planform Area	4,610 ft^2	4,745 ft^2

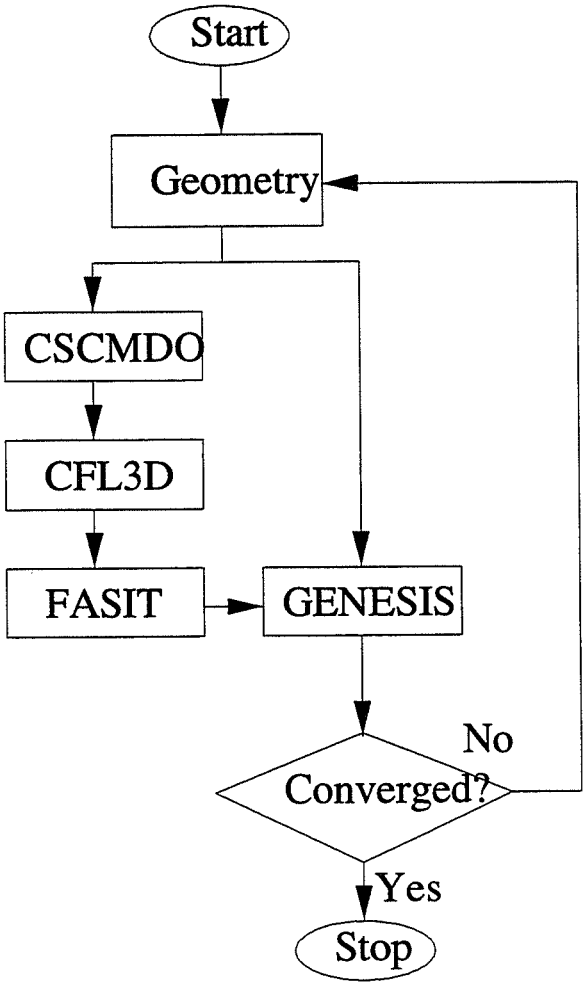


Figure 1. The software organization used in HSCT static aeroelastic analysis.

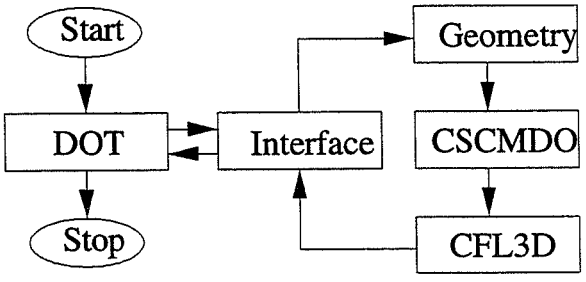


Figure 2. The software organization used in HSCT aerodynamic optimization.

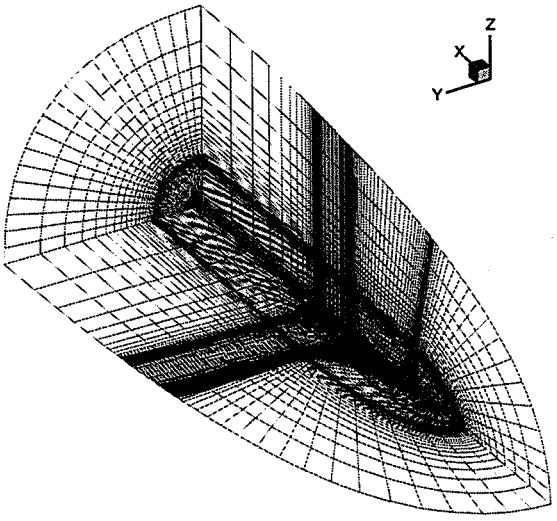


Figure 3. One of two blocks in the aerodynamic model showing the starboard wing and fuselage along with the x-z plane of symmetry and the exit plane.

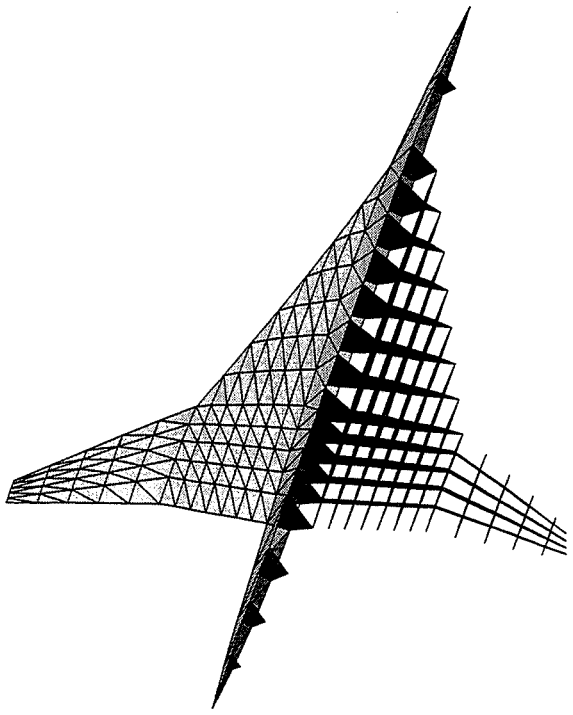


Figure 4. Structural model of the HSCT showing the wing skin elements (port) and the rib/spar elements (starboard).

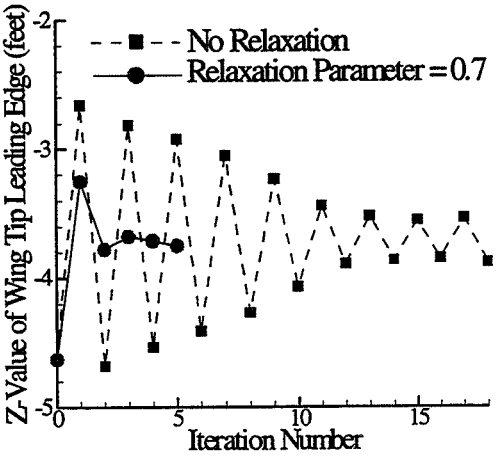


Figure 5. The convergence history of the aeroelastic analysis both with and without under-relaxation.

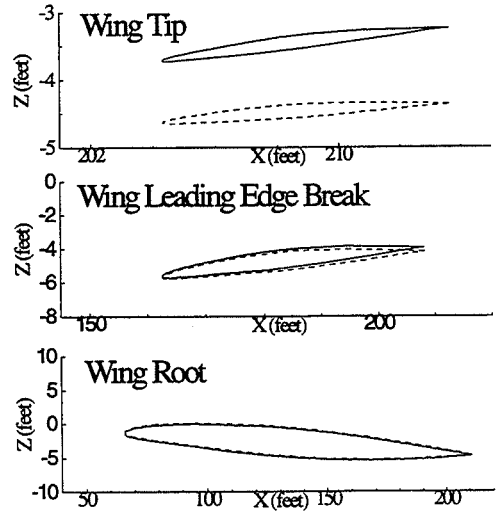


Figure 6. The initial undeformed (dashed) and final deformed (solid) airfoil sections obtained from the static aeroelastic analysis.

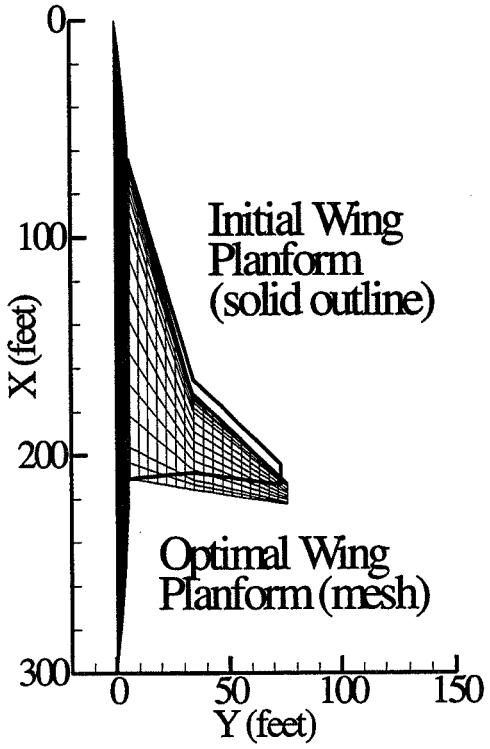


Figure 7. The initial (solid) and optimal (mesh) HSCT planforms in the aerodynamic optimization.

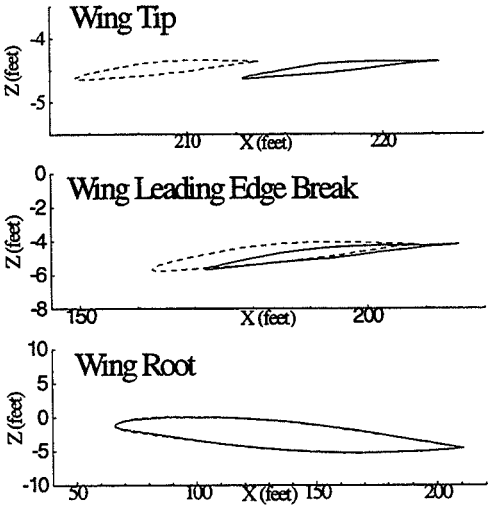


Figure 8. The initial (dashed) and optimal (solid) airfoil sections obtained from the aerodynamic optimization.

PARALLEL COMPUTATION OF SENSITIVITY DERIVATIVES WITH APPLICATION TO AERODYNAMIC OPTIMIZATION OF A WING

Robert T. Biedron (*r.t.biedron@larc.nasa.gov 757-864-2156*)
Jamshid A. Samareh (*j.a.samareh@larc.nasa.gov* 757-864-5776)
Lawrence L. Green (*l.l.green@larc.nasa.gov 757-864-2228*)

NASA Langley Research Center
Hampton, VA 23681-2199

ABSTRACT

This paper focuses on the parallel computation of aerodynamic derivatives via automatic differentiation of the Euler/Navier-Stokes solver CFL3D. The comparison with derivatives obtained by finite differences is presented and the scaling of the time required to obtain the derivatives relative to the number of processors employed for the computation is shown. Finally, the derivative computations are coupled with an optimizer and surface/volume grid deformation tools to perform an optimization to reduce the drag of a three-dimensional wing.

INTRODUCTION

Recently researchers have shown a great deal of interest in the application of advanced CFD methods to aerodynamic optimization, for both single-discipline and multidiscipline applications. Central to any aerodynamic optimization problem is the evaluation of solution derivatives with respect to the chosen design variables. Differentiation of the CFD source code used to obtain the solution gives *exact* derivatives of the discrete equations, without the step size problems of finite differences. Although quite tedious to perform by hand, exact differentiation of a source code is readily accomplished using an automatic differentiation (AD) tool such as ADIFOR[1]. The computational time for AD derivatives scales with the number of design variables, and the computational time may be prohibitive for large number of design variables. One way to reduce the effective computation time (wall time) is to subdivide the computational domain and compute each subdomain on a different processor. For this approach to be useful, the computational code must scale well with increasing number of processors.

The CFD code used for this study, CFL3D[2], has been widely used for aerodynamic analysis on complex configurations. One version of the code (CFL3Dv4.1hp) has recently been ported to parallel computer architecture via the use of MPI protocols. Studies have indicated good scaling on Origin 2000 testbeds for Euler and Navier-Stokes solutions[3]. Even more recently the parallel code has been passed through the ADIFOR automatic differentiation tool to generate code capable of computing both the solution and the gradient of the solution with respect to geometric design variables.

PARAMETERIZATION AND DESIGN VARIABLES

For aerodynamic optimization, a parameterized surface definition that relates the shape to geometric design variables is required. In many instances, a computational grid defining the baseline shape of the configuration is readily available, but a parameterization of the surface is not.

A surface parameterization scheme that overcomes this difficulty has recently been developed by the second author[4]. The method is a free-form deformation approach very similar to morphing techniques used in computer animation. It can simulate planform, twist, dihedral, thickness, and camber variations. In a sense, the model is treated as putty or clay in areas where it can be twisted, bent, tapered, compressed or expanded, but retains the same topology. The method is equally applicable to computational structures grids, and thus is ideally suited for aerostructural calculations.

An existing grid defining the ONERA M6 wing[5] was parameterized with 52 parameters. Of those 52 parameters, 31 were chosen as design variables: 5 planform, 4 twist, 4 shear, 9 thickness and 9 camber. Figure 1 shows the locations of the design variables chosen for the wing optimization.

COMPARISON WITH FINITE DIFFERENCES

As a validation that the AD code produces the correct derivatives, comparisons with central finite differences (FD) were made using double-precision arithmetic. The AD derivatives and finite differences were computed for inviscid flow over the M6 wing with the design variables described above. The flow conditions were Mach 0.84 and . A coarse grid of dimensions 97x17x17 in the streamwise, spanwise and normal directions, respectively, was used for the derivative validation studies. For the FD results, residuals were driven to machine zero; for the AD results, computations were stopped when derivatives no longer varied in the fifth decimal place. All finite differences were computed using a step size of 10^{-6}. Experience has shown that single precision is sufficient for inviscid analysis, e.g. negligible difference in force coefficients between single and double precision. The AD calculations were repeated with single precision to see if the same would hold true for derivatives of the force coefficients.

The results are summarized in Table 1 for several representative design variables. Similar results are obtained for other derivatives and other force coefficients. It is evident that the AD code does in fact produce the correct derivatives. Note that the AD results are the same for both single and double precision, at least to 4 decimal places, a result typical of other derivatives as well. Thus, the AD code can be used reliably with single precision, at least for the inviscid flow considered here. The advantage is that the code runs approximately 40% faster in single precision. Although not shown, it should be noted that when used with single precision, finite differences could not be obtained with better that approximately one percent error as compared to double precision. Furthermore, different design variables required different step sizes to obtain even that level of accuracy.

SCALING STUDY

The scaling study was carried out for inviscid flow over a High Speed Civil Transport (HSCT) configuration at Mach 2.4. The grid used was comprised of approximately 540,000 cells in 64 equal sized blocks. The surface was parameterized with 27 design variables in a manner similar to that used for the M6 wing described above. For the scaling studies, 100 three-level multigrid iterations were used, resulting in derivatives that remained unchanging with iteration number through the fifth decimal place. The computations were carried out in single precision. The results were obtained on Origin 2000 computers, using from 1 to 32 compute processors (an extra processor functions as the host, performing I/O tasks which consume relatively little CPU time). Each case was run at least twice to try to account for run-to-run variations due to system load.

The scaling results are shown in Figure 2. Computing only the solution (no derivatives) required 0.787 hours on a single processor, dropping to 0.023 hours on 32 processors. Computing the 27 AD derivatives along with the solution required nearly 33.5 hours on a single processor, dropping to 1.05 hours on 32 processors. The speedup was essentially linear for both solution and solution plus gradient calculations.

WING OPTIMIZATION

As an application of the parallel AD code, an aerodynamic optimization of an ONERA M6 wing was carried out. The objective of the optimization was to minimize the drag while maintaining the same lift as the baseline design. As for the derivative validation, inviscid flow at Mach 0.84 and was used, however a finer grid of dimensions 197x33x33 was employed for the optimization. The design variables used were the 31 shown in Figure 1, although for the current study the planform variables were constrained so that they did not change during the design, resulting in a fixed wing area. This eliminated the need for an additional code to calculate the wing area and derivatives of the wing area with respect to the design variables. Also, to prevent negative cell volumes near the tip, thickness variables Th3, Th6, and Th9 were constrained so as not to change. Design variable limits were arbitrarily chosen as follows: twist, +/- ; all others, +/- 1 percent span.

The optimizer used for this work is a modified version of the CONMIN code[6] known as JOPT7. Within each optimization cycle, the solution and gradient data provided to the optimizer are used to determine a linear approximation to the objective function and constraints used in the 1D line searches. This makes each line search much faster, but the linear approximation is only valid with a small region of the current solution. User-defined move limits for the design variables are required to insure that the optimizer searched only where the current linear approximation was reasonable.

The solution and design-variable changes suggested by the optimizer were incorporated into the surface model using the geometry deformation scheme mentioned earlier. Next, an AD version of the CSCMDO code[8] was used to propagate the difference between the old and new surfaces smoothly throughout the volume grid, determining the grid sensitivities in the process.

Figure 3 shows the design cycle history for both lift and drag. In this optimization, the angle of attack is fixed, and it was found that in order to move away from the current design, the constraint on the lift coefficient had to be relaxed temporarily. This is shown clearly in the figure: for the first 19 design cycles, C_L is allowed to deviate by up to 0.01 from the desired value. After design cycle 19 the tolerance on the lift constraint is tightened to 10^{-6}. The drag increased slightly when the lift constraint was tightened, but after the initial rise there was no further change in drag at the target lift coefficient. The net result was approximately 29 counts of drag reduction at the baseline lift. Figures 4 and 5 show comparisons of the solutions computed on the initial and final designs. The results indicate a significant reduction in the shock strength at most spanwise stations. Also shown in Figure 5 are initial and final wing sections at selected spanwise stations.

Using 16 compute processors on a 250 Mhz Origin 2000, each design cycle took approximately 115 minutes, of which approximately 100 minutes was spent in evaluating the 31 gradients, using 300 multigrid cycles. The time per design cycle can be reduced as desired by increasing the number of processors employed. Although not done in this preliminary study, it should be possible to further reduce the total optimization cost by utilizing the mesh sequencing option in CFL3D to perform most of the design variable changes on a coarser level, and only then moving up to the finest level for the final design cycles.

CONCLUDING REMARKS

A parallel, differentiated version of the CFL3D code has been demonstrated to yield accurate derivatives with respect to geometric design variables. Furthermore, these computationally intensive derivative calculations have been shown to scale well with increasing number of processors. The parallel AD code was coupled to grid deformation and optimization packages and used to reduce the inviscid, transonic drag on a wing. Future applications will consider viscous flows.

REFERENCES

1. Bischof, C., Carle, A., Khademi, P., Mauer, A., and Hovland, P.; "ADFIFOR 2.0 User's Guide," Argonne National Laboratory Mathematics and Computer Science Division Technical Memorandum No. 192, August, 1995.

2. Krist, S. L., Biedron, R. T., and Rumsey, C. L.; "CFL3D Users Manual (Version 5.0)," NASA TM-1998-208444, June 1998.

3. Faulkner, T.; "Origin 2000 Update: Studies Show CFL3D Can Obtain Reasonable Performance," NAS News, Vol. 2, No. 27, Nov.-Dec. 1997.

4. Samareh, J. A.; "Geometry Modeling and Grid Generation for Design and Optimization," ICASE/LaRC/NSF/ARO Workshop On Computational Aerosciences In The 21st Century, Hampton, VA, April 22-24, 1998.

5. Schmitt, V., and Charpin, F.; "Pressure Distributions on the ONERA-M6 Wing at Transonic Mach Numbers," Experimental Data Base for Computer Program Assessment, AGARD-AR-138, May 1979, pp. B1-1 - B1-41.

6. Vanderplaats, G. N.; "CONMIN- A FORTRAN Program For Constrained Function Minimization," NASA TM X-62282, August, 1973.

7. Walsh, J. L., Young, K. C., Pritchard, J. I., Adelman, H. M., and Mantay, W. R.; "Integrated Aerodynamic/Dynamic/Structural Optimization of Helicopter Rotor Blades Using Multilevel Decomposition,", NASA TP 3465 / ARL TR 518, January, 1995.

8. Jones, W. T., and Samareh-Abolhassani, J.; "A Grid Generation System for Multi-disciplinary Design Optimization," Proceedings from the 12th AIAA Computational Fluid Dynamics Conference, AIAA-95-1689, San Diego, CA, June 1995, pp. 657-669.

Derivative	AD (DP)	FD (DP)	% error (DP)	AD (SP)
$dC_{L/d(Tw\ 3)}$	-0.02944	-0.02944	0.0	-0.02944
$dC_{L/d(Th\ 8)}$	+0.43321	+0.43321	0.0	+0.43323
$dC_{L/d(Ca\ 8)}$	+2.8380	+2.8380	0.0	+2.8380
$dC_{D/d(Tw3)}$	-0.00246	-0.00246	0.0	-0.00246
$dC_{D/d(Th\ 8)}$	+0.07016	+0.07016	0.0	+0.07016
$dC_{D/d(Ca\ 8)}$	+0.16467	+0.16467	0.0	+0.16467

Table 1. Accuracy of lift and drag coefficient derivatives computed using automatic differentiation and central finite differences. DP denotes double precision; SP denotes single precision.

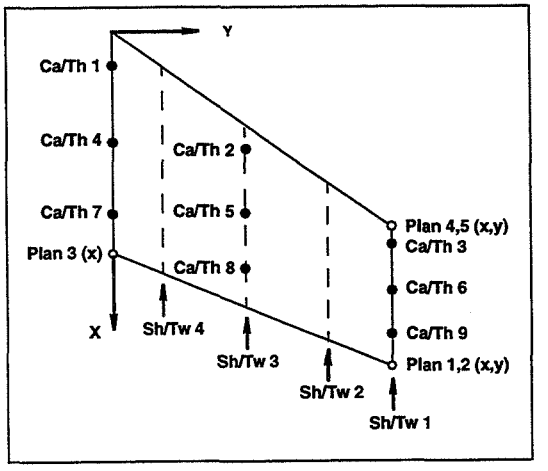


Figure 1. Design variable locations; Ca/Th denotes camber/thickness variables at points indicated by the solid circles; Sh/Tw denotes shear/twist variables, defined along the dashed lines; Plan denotes planform variables, at points indicated by the empty circles.

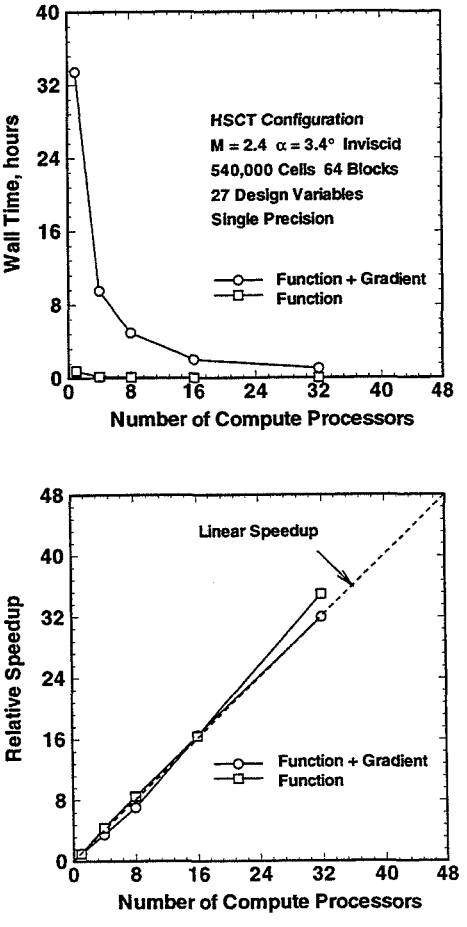


Figure 2. Origin 2000 scaling for both solution and solution plus gradient evaluation for an HSCT configuration with 27 design variables.

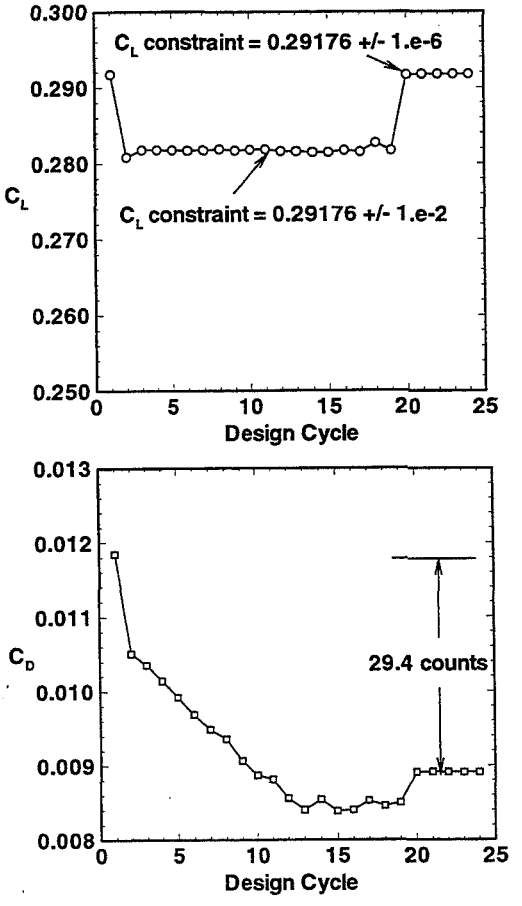


Figure 3. Design cycle history for ONERA M6 wing optimization.

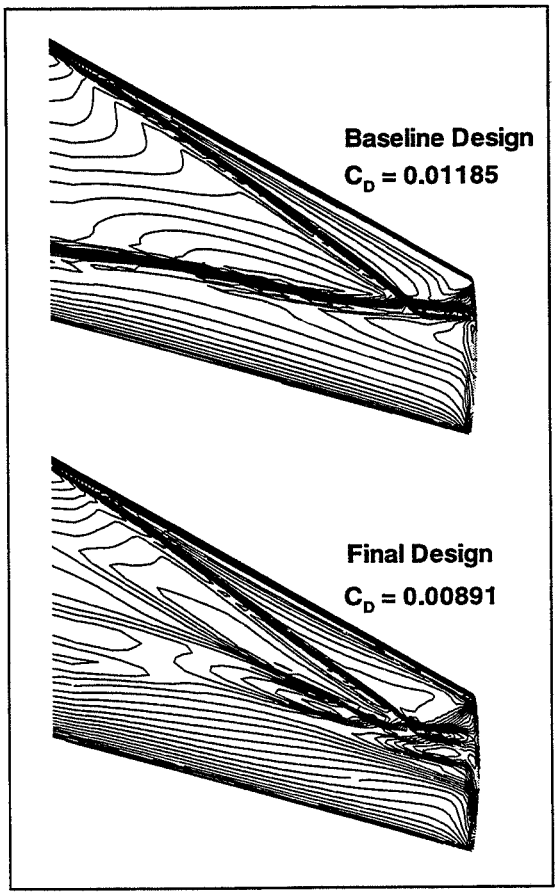


Figure 4. Comparison of surface pressures on the final wing design with the baseline M6 wing.

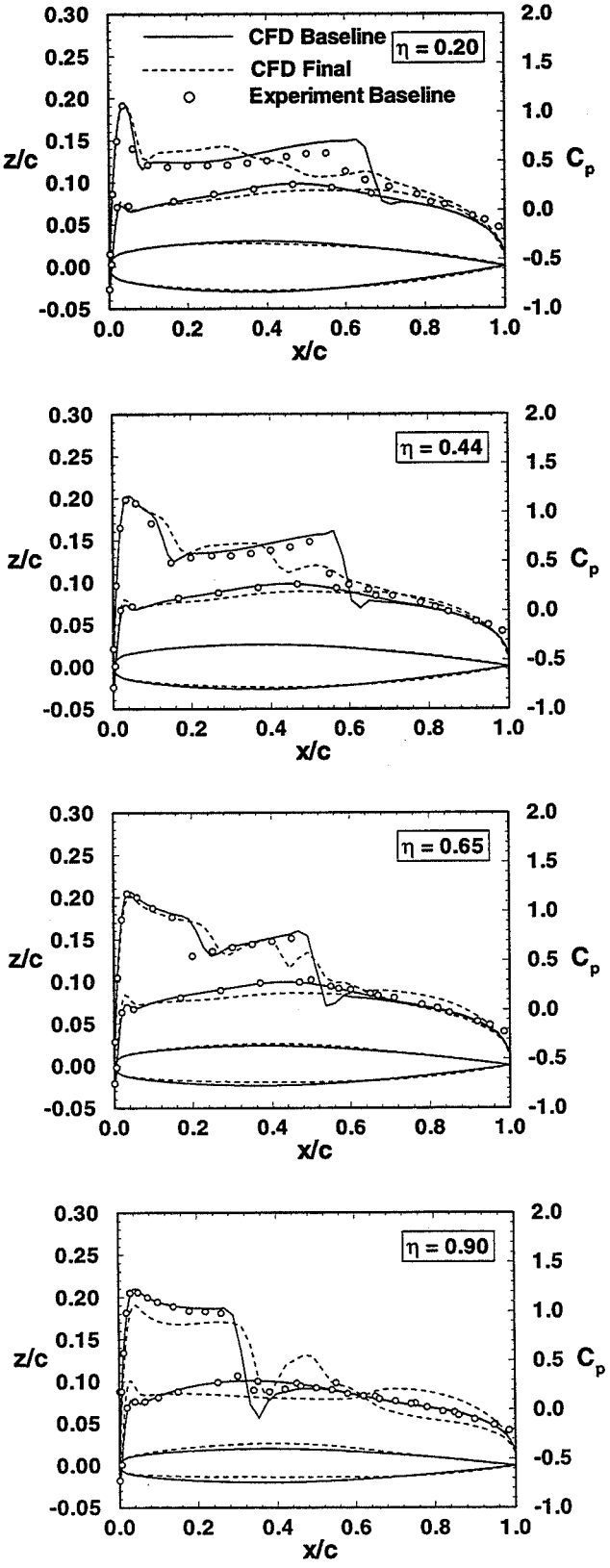


Figure 5. Comparison of initial and final Cp distribution and wing cross section at selected spanwise stations.

541-02

DEMONSTRATION OF AUTOMATICALLY-GENERATED ADJOINT CODE FOR USE IN AERODYNAMIC SHAPE OPTIMIZATION

018276

Lawrence L. Green
Mail Stop 159 NASA Langley Research Center, Hampton, VA 23681-2199
l.l.green@larc.nasa.gov 757-864-2228

366902

Alan Carle and Mike Fagan
Mail Stop 134, Rice University, 6100 Main Street, Houston, TX 77005-1892
carle@cs.rice.edu 713-285-5368
mfagan@cs.rice.edu 713-285-5178

6 p.

Introduction

Gradient-based optimization requires accurate derivatives of the objective function and constraints. These gradients may have previously been obtained by manual differentiation of analysis codes, symbolic manipulators, finite-difference approximations, or existing automatic differentiation (AD) tools such as ADIFOR (Automatic Differentiation in FORTRAN)[1]. Each of these methods has certain deficiencies, particularly when applied to complex, coupled analyses with many design variables. Recently, a new AD tool called ADJIFOR (Automatic Adjoint Generation in FORTRAN), based upon ADIFOR, was developed and demonstrated[2]. Whereas ADIFOR implements forward-mode (direct) differentiation throughout an analysis program to obtain exact derivatives via the chain rule of calculus, ADJIFOR implements the reverse-mode counterpart of the chain rule to obtain exact adjoint form derivatives from FORTRAN code.

Automatically-generated adjoint versions of the widely-used CFL3D computational fluid dynamics (CFD) code[3,4] and an algebraic wing grid generation code were obtained with just a few hours processing time using the ADJIFOR tool. The codes were verified for accuracy and were shown to compute the exact gradient of the wing lift-to-drag ratio, with respect to any number of shape parameters, in about the time required for 7 to 20 function evaluations[2]. The codes have now been executed on various computers with typical memory and disk space for problems with up to 129 x 65 x 33 grid points, and for hundreds to thousands of independent variables.

These adjoint codes are now used in a gradient-based aerodynamic shape optimization problem for a swept, tapered wing. For each design iteration, the optimization package constructs an approximate, linear optimization problem, based upon the current objective function, constraints, and gradient values. The optimizer subroutines are called within a design loop employing the approximate linear problem until an optimum shape is found, the design loop limit is reached, or no further design improvement is possible due to active design variable bounds and/or constraints. The resulting shape parameters are then used by the grid generation code to define a new wing surface and computational grid. The lift-to-drag ratio and its gradient are computed for the new design by the automatically-generated adjoint codes. Several optimization iterations may be required to find an optimum wing shape. Results from two sample cases will be discussed. The reader should note that this work primarily represents a demonstration of use of automatically-generated adjoint code within an aerodynamic shape optimization. As such, little significance is placed upon the actual optimization results, relative to the method for obtaining the results.

Problem Description

The grid generation code was created in FORTRAN specifically for use with ADIFOR and ADJIFOR automatic differentiation studies. A simple, algebraic grid generation method for a wing alone is used. The user specifies the number of grid points to be generated in each coordinate direction, the number of wing sections to be input, and eight parameters for each wing section that

describe both the wing planform and its cross-sectional shape at the specified wing stations. The wing section inputs are used to construct a wing surface composed of an expanded family of NACA four-digit airfoils, defined by real numbers rather than integers; this construction provides continuous cross-sectional shape derivatives and allows for perturbing the wing section shape by small amounts. The number of shape parameters used as design variables within the optimization problem is proportional to the number of input wing sections. The wing surface is wrapped in a single-block volume grid with C-O topology that can be split into a multiblock grid via a utility program developed by Beidron of the NASA Langley Research Center. The grid generation program provides for limited control on the spacing of the grid points in each direction, but does not include many other features commonly found in grid generation software. Grid quality and good resolution of the flow field details were not considered to be high priorities in this work.

This work is primarily a demonstration of the ADJIFOR-generated adjoint capability. As such, it is best to use as many shape design variables within the optimization process as possible. However, the ADJIFOR-generated gradients must be verified for accuracy by other means that are less practical for many design variables on large grids (ADIFOR or finite differences). Test problems were chosen as "large enough" to demonstrate adjoint efficiency, while allowing for gradient verification by other means.

The CFD code used in this work is the CFL3D program[3,4] developed by Thomas, Rumsey, and Beidron of NASA Langley Research Center. For this work, both the sequential version 5.0 and the Message-Passing Interface (MPI) parallel version 4.1 codes were used. Both codes solve the Euler or Navier-Stokes flow equations in conservation form and include numerous solver, grid interface, and turbulence modeling options. As described in reference 2, the codes were entirely differentiated by ADJIFOR, except for the turbulence models and the patched- and overset-grid options. Of the many solver and grid options available in the ADJIFOR-generated CFL3D codes, only a subset of these options have been exercised in this demonstration. One notable option that was differentiated, and not exercised in reference 2, but is used within the present work, is the use of multigrid to converge the flow derivatives in adjoint form. The use of multigrid has been found to significantly reduce the number of iterations required of the adjoint flow solver and made practical the grid runs for grids of sizes up to 129 x 65 x 33.

For this work, ADJIFOR was applied to the grid and CFD codes. The resulting codes produce exact adjoint form derivatives of the wing lift-to-drag ratio with respect to many wing shape parameters. For each design iteration, the current objective function and its gradients are used to construct an approximate, linear optimization problem. The optimizer code repeatedly interacts with this linear optimizaton problem during a design iteration to maximize the wing lift-to-drag ratio within the design variable bounds. Move limits are also imposed to prevent large design variable changes during one design iteration. The resulting shape variables are then used to construct a new wing surface and volume grid for which the lift-to-drag ratio gradient is then computed via ADJIFOR-generated code. This process, illustrated in figure 1, may be repeated until an optimum wing shape is found.

Results

The objective function (wing lift-to-drag ratio) history for a 33 x 9 x 9 grid in transonic flow ($M=0.84$, $\alpha=3.06°$) for 15 optimization iterations is shown in figure 2. In this example, the wing is described by 11 input wing sections. The gradients are computed for 88 design variables and the optimization allows both section and planform design variables to change, except at the wing root and tip, which are fixed. Each optimization cycle imposes a ten percent move limit on the design variables. The objective function increases almost linearly and has not reached a maximum, suggesting that perhaps the move limits could be increased for this case and that more optimization

iterations should be executed. The baseline and optimized wing sections and planform are shown in figures 3 and 4, respectively. The optimized wing has thinner wing sections, shorter chord lengths, and some cambering toward the wing tip. The optimized planform has increased leading edge sweep, except where the tip was constrained to prevent negative cell volumes occurring in the grid generation process.

The objective function (wing lift-to-drag ratio) history for 129 x 65 x 33 grid in transonic flow ($M=0.84$, $\alpha=3.06°$) for nine optimization iterations is illustrated in figure 5. In this example, the wing is described by 21 input wing sections (168 design variables) . The optimization allows both planform and thickness design variables to change, except at the wing root, which is fixed to prevent negative cell volumes. Move limits of 10 percent were initially used; these were subsequently reduced to only one percent to prevent negative cell volumes from occurring in the grid generation process. The objective function increases and apparently has almost reached a maximum. The baseline and optimized wing planforms are shown in figures 6 and 7, respectively; shading is proportional to the surface pressure coefficient resulting from a 33 x 9 x 9 grid analysis of the final 129 x 65 x 33 grid design. The optimized wing exhibits a different shock pattern than the baseline wing and has more highly swept and more elliptical planform than the baseline wing. Gradients for 168 design variables were obtained in about the time required for 15 function evaluations using 33 nodes on the NAS Origin 2000 parallel computer.

These optimizations illustrate the ability of ADJIFOR to compute derivatives for complex flow regimes. Both section and planform design variables were allowed to change in the optimization to provide as many design variables as possible for the adjoint formulation. However, planform optimization in transonic flow results in unusual wing shapes that are still being investigated.

Conclusions

This work is primarily a demonstration of using automatically-generated adjoint code to efficiently compute derivatives for an aerodynamic shape optimization in from a complex flow regime with a multigrid algorithm . Sequential and parallel automatically-generated adjoint versions of the CFL3D computational fluid dynamics code, coupled with an adjoint-form grid generation code, have been successfully demonstrated within an aerodynamic shape optimization in transonic flow, for grids including up to 129 x 65 x 33 points. Gradients for up to 168 shape design variables were computed in about the time required for 15 function evaluations using 33 nodes on the NAS Origin 2000 parallel computer. The resulting designs improved the wing lift-to-drag ratio over that of the baseline wing, although little significance is attached to the specific optimization results.

References

1. Bischof, C.; Carle, A; Corliss, G.; and Griewank, A.: ADIFOR Generating Derivative Codes from FORTRAN Program. *Scientific Programming*, vol. 1, 1992, pp. 11-29.

2. Carle, Alan; Fagan, Mike; and Green, Lawrence L.: Preliminary Results From the Application of Automated Adjoint Code Generation to CFL3D. AIAA 98-4807, Sept. 1998.

3. Beidron, R.; and Thomas, J.: A Generalized Patched-Grid Algorithm with Application to the F-18 Forebody and Actuated Control Strake. *Computing Systems in Engineering*, vol. 1, No. 2-4, 1990, pp. 563-576.

4. Rumsey, C.; Beidron, R.; and Thomas, J.: CFL3D: Its History and Some Recent Applications. NASA TM 112861, May 1997.

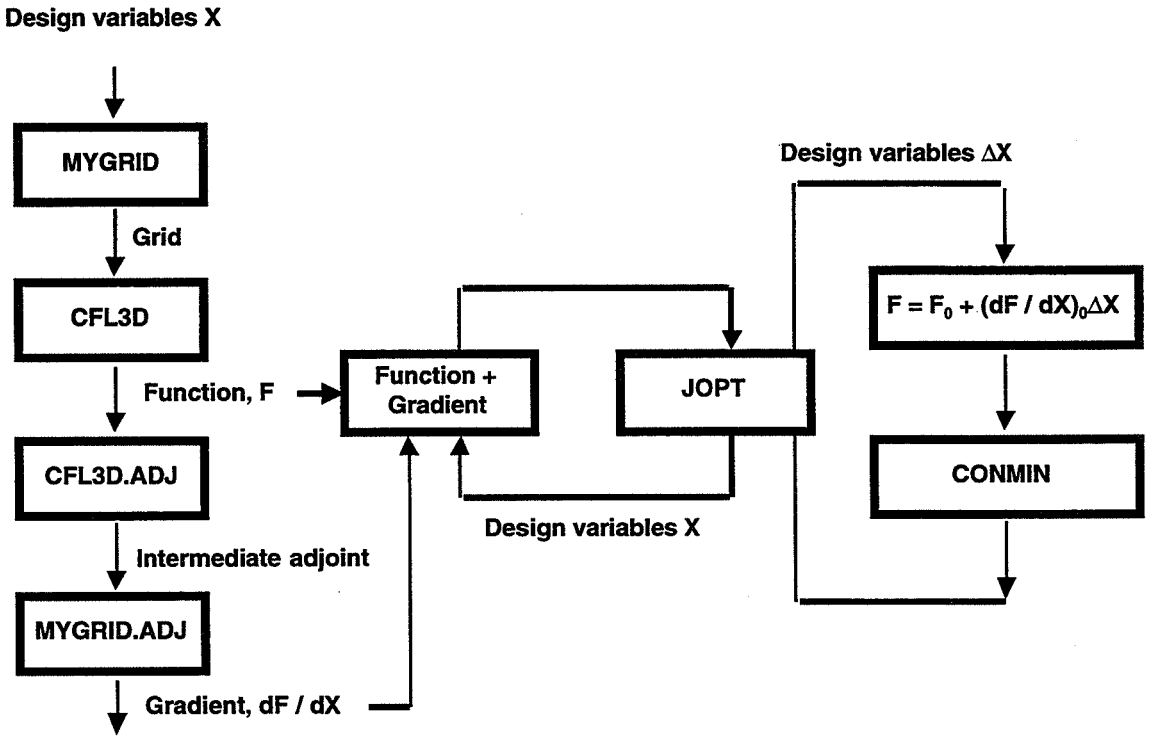


Figure 1. Optimization process flow chart.

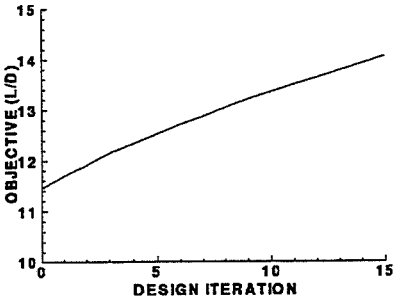


Figure 2. Optimization history (33 x 9 x 9 grid, 88 design variables, M = 0.84, α = 3.06°).

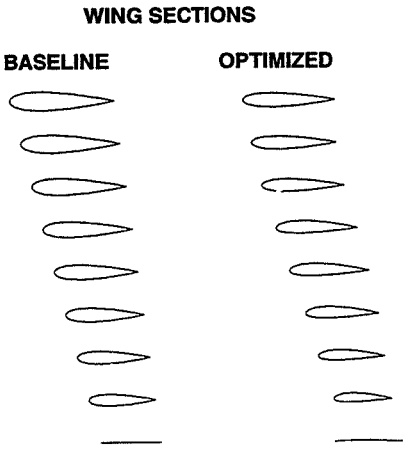


Figure 3. Baseline and optimized wing section with root and tip fixed (33 x 9 x 9 grid, 88 design variables, M = 0.84, α = 3.06°).

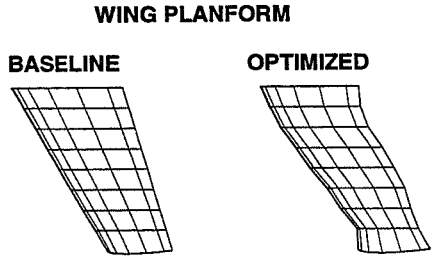


Figure 4. Baseline and optimized wing planform with root and tip fixed (33 x 9 x 9 grid, 88 design variables, M = 0.84, α = 3.06°).

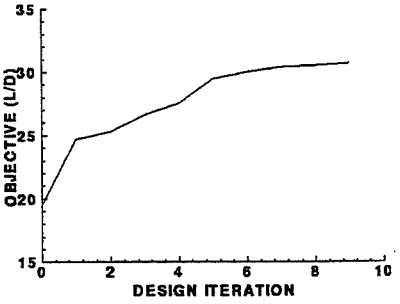


Figure 5. Optimization history (129 x 65 x 33 grid, 168 design variables, M = 0.84, α = 3.06°).

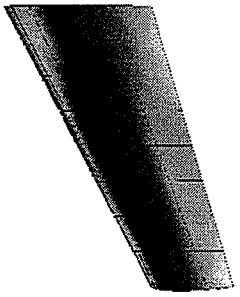


Figure 6. Baseline wing planform with root fixed (129 x 65 x 33 grid for design, 33 x 9 x 9 grid for analysis 168 design variables, M = 0.84, α = 3.06°).

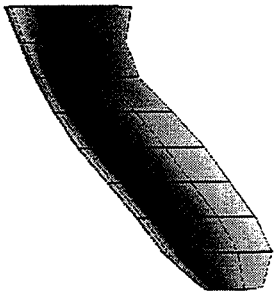


Figure 7. Optimized wing planform with root fixed (129 x 65 x 33 grid for design, 33 x 9 x 9 grid for analysis 168 design variables, M = 0.84, α = 3.06°).

APPLICATIONS OF PARALLEL PROCESSING IN AERODYNAMIC ANALYSIS AND DESIGN

P. Sundaram and James O. Hager
Phantom Works, The Boeing Company
Long Beach, CA
pichuraman.sundaram@boeing.com, james.o.hager@boeing.com

Introduction

The continuously growing size and computational complexity of CFD-based aerodynamic analysis problems demand larger and larger computational resources. In addition, quick turn-around time for design and synthesis are necessary to make high fidelity, CFD-based techniques practical. Typical full-configuration, Navier-Stokes analysis grids tend to have more than 10 million points, and the solutions to these problems require very large amounts of CPU time and memory. Also, CFD-based nonlinear shape optimization of full aircraft configurations is required within a few weeks to meet the cost and schedule challenges of today's aerospace customer. Traditional sequential computers cannot deliver these large computing resources, and no new large sequential vector supercomputers are under development. Thus, parallel processing has emerged as the most efficient and cost-effective method to achieve the large computational resources required for these advanced CFD applications.

This paper presents the recent progress made in the application of the CFL3Dhp parallel code for configuration analyses and aerodynamic shape optimization at Boeing-Phantom Works (BPW). CFL3Dhp is the coarse-grain parallel version of the CFL3D Euler/Navier-Stokes solver developed at NASA LaRC. CFL3Dhp utilizes the MPI message-passing library to exchange information with other task processors as well as with the host. CFL3Dhp runs on most available parallel platforms and distributed environments. Several utilities have been developed at BPW to provide a user-friendly parallel environment.

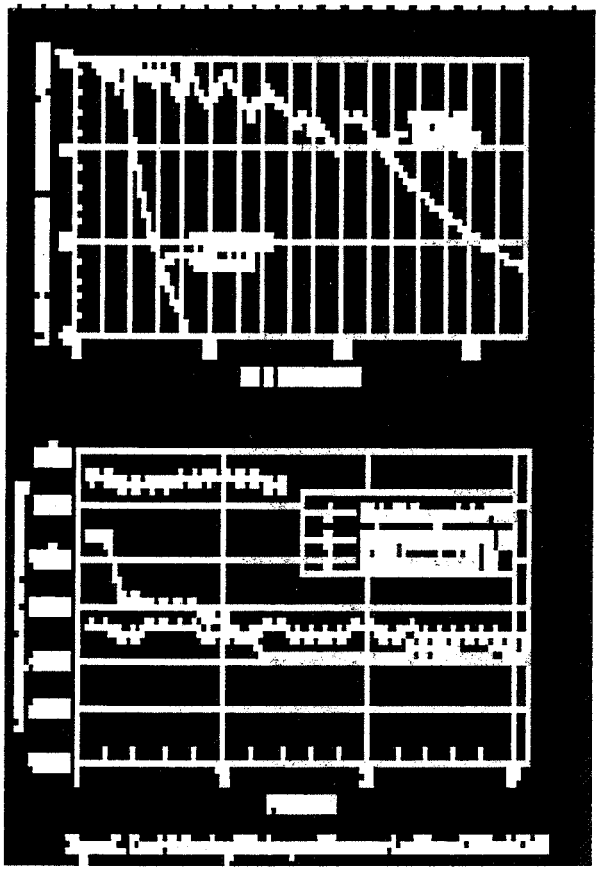
CFL3Dhp Parallel Preprocessing Tools

In the CFD analysis of a multiblock grid on a sequential shared memory supercomputer, variations in the size of the grid blocks is not an issue. For coarse-grain parallel processing on distributed memory platforms, it is necessary to split the large grid blocks that may be too large for a processor (PE) memory prior to mapping them onto the PEs. However, with CFL3Dhp, when grid blocks are split, the input files must be re-created, including the patched-interface interpolation file. The *precflinp* utility developed at BPW automatically prepares the input files for both CFL3Dhp and the patched-interface interpolation file generation program, *ronnie*, when blocks are split.

Another CFL3Dhp utility, *precfl3d,* provides the work array size parameters needed for CFL3Dhp. *precfload*, developed at BPW, is a corollary to *precfl3d* and outputs the load-balancing efficiency for all possible block-to-processor mappings. The actual number of processors to be used is determined based on both load-balancing efficiency and the maximum load. Finally, to provide fault-tolerance, in case of a processor failure during the execution in distributed computing, it is necessary to recombine the grid, solution, and restart files to exclude the failed PE. This is performed using the *Splicom* utility developed at BPW that will split and combine grids and solutions.

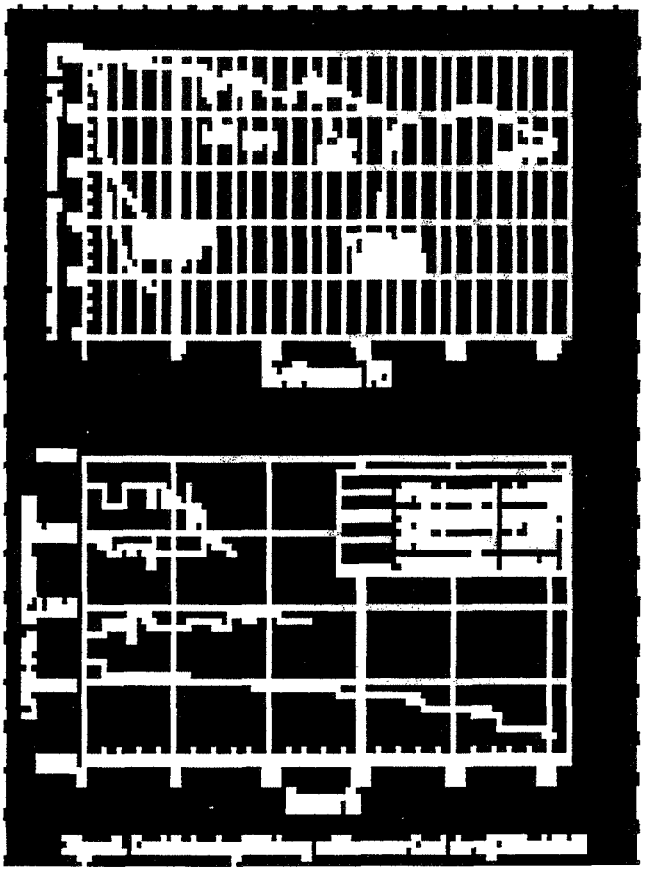
Parallel Performance Enhancement

Load-balancing and scalability are vital to achieve high parallel efficiency. Two problems are used to illustrate this. The first problem is a 42-block grid with 13.6 million points for a wing/fuselage/nacelle/diverter/empennage full-configuration in sideslip Navier-Stokes analysis. Figure 1 shows the efficiency curve and the corresponding load distribution for various block-to-processor mappings. If the 42-block grid is mapped onto 42 PEs, it would yield a poor load-balancing efficiency of only 42%. Mapping the same grid onto 31 PEs gives nearly 52% efficiency. When the grid is mapped onto 14 PEs, it gives nearly 97% load-balancing efficiency. The load distribution for this last mapping is given by the "circle" symbols. This mapping, although efficient, uses only 14 PEs and hence takes long wall-time for completing the solution. It would be desirable to use more number of PEs to reduce wall-time for the run and still maintain high load-balancing efficiency.



To increase the number of PEs, and maintain good load-balancing, it is necessary to split the blocks. Block-splitting also increases the flexibility to map the blocks to PEs compared to the original 42-block grid. Indeed, by splitting this grid into 170 blocks and mapping it onto 31 PEs, nearly 95% efficiency can been achieved (as seen in the 170-block efficiency curve). The nearly uniform processor loads corresponding to this mapping is shown by the "delta" symbols. To reduce wall-time, this split grid can be mapped onto 106 PEs and still achieve nearly 90% efficiency. A five-point, supersonic Navier-Stokes polar for this problem was obtained in one day on a 106 PEs CRAY T3E.

The second example is a 37 block 9.5 million points grid for a Navier-Stokes simulation of an installed, powered nozzle. The 37-block 9.5 million-point grid is split into 254 blocks to improve parallel efficiency. The load-balancing efficiency and the load distribution for this problem are shown in Figure 2. The mappings of this split grid onto 64, 80, and 122 PEs yield greater than 94% efficiency. These mappings provide the CFL3Dhp scalability data shown in Table 1. It can be seen that CFL3Dhp scales well



with an increasing number of PEs. Also shown in Table 1 is the timing of a single PE run on CRAY C90 for this problem which requires 400 MW of memory. The 122 PE run produces an aggregate processor speed of nearly 4.5 GigaFLOPS.

Table 1. Table highlighting scalability of CFL3Dhp with increasing PEs

PE	**CPU/PE/iter** (minutes)	**Wall time for 10000 iter** (hours)
122	31.7	43.30
80	29.4	61.25
64	29.3	78.30
1 (C90)	2.9	483.00

Parallel ADIFOR Sensitivities

Thus far, the details of the CFL3Dhp parallel environment for CFD analyses have been presented. Now, some significant progress made at BPW in the area of aerodynamic shape optimization using parallel processing is described. In any gradient-based aerodynamic optimization procedure, calculating the sensitivity of the flow to geometry changes requires the most CPU time. Here, CFL3Dhp has been used as the analysis code from which to obtain computer generated analytical flow gradients.

To compute the grid and flow sensitivities, the Automatic DIfferentiation of FORtran (ADIFOR[1]) software has been used. ADIFOR augments computer codes to compute derivatives of their outputs w.r.t. inputs by applying the chain rule. ADIFOR generates accurate analytic sensitivities and avoids the step-size issue associated with finite-difference calculations. The gradient calculations using ADIFOR require CPU times and memory proportional to the number of design variables (DVs). Specifically, the CPU time is at least (nDV+1) times the function evaluation CPU time, where nDV is the number of DVs. Because the ADIFOR-computed sensitivities are accurate even with single-precision variables, it is possible to reduce the memory required on some platforms by using single-precision instead of the default double-precision.

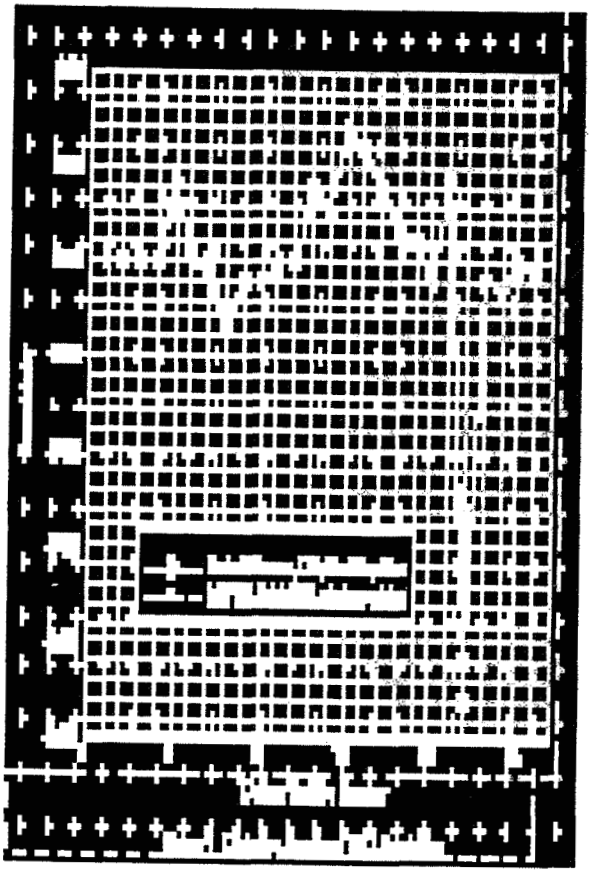
To compute grid sensitivities, the BPW grid tools were ADIFORed to obtain the corresponding sensitivity codes. The grid tools are *geom1*, which applies shape-function perturbations to a baseline geometry using a set of DVs to generate the perturbed geometry; *qgrid*, which generates the surface grid for the perturbed geometry; and *flexmesh*, which perturbs a reference volume to conform to the perturbed-geometry surface grid. The grid-generation steps are performed in parallel for each DV.

To compute flow sensitivities, the incremental iterative version of CFL3Dhp, CFL3Dhp.ADII[2], has been chosen in order to reduce the CPU time. The parallel environment allows both the wall time and memory requirements of a large design to be distributed over a number of PEs. The sensitivity calculation is broken up three ways. First, the grid is split and mapped onto a group of PEs. Second, a subset of DVs are computed on each PE group. Third, multiple PE groups are used to evaluate multiple DV subsets simultaneously. Thus, each PE solves for the flow and the gradients of *its* part of the grid for *its* group of DVs. The CPU time required to calculate the Euler flow sensitivities for 50 DVs on a grid with 0.5 million points using a 24 PE SGI Origin 2000 system is 63 times the function evaluation time. This problem is too large for sequential systems because CFL3Dhp.ADII would require approximately 10 Gigawords of memory and 400 single-PE Origin 2000 CPU hours. Using 24 PEs, the wall-time required on Origin 2000 is small. The drag-to-lift (D/L) sensitivity to a set of 26 representative DVs as computed by finite-differences and CFL3Dhp.ADII are compared in Figure 3.

Parallel ADJIFOR for Adjoint Sensitivities

Recently, the ADIFOR software has been enhanced to allow it to be applied in the reverse mode (ADJIFOR) to automatically generate adjoint code. In the reverse mode, sensitivities of the analysis code output with respect to intermediate quantities are calculated but reversing the program flow and storing the intermediate values that have a nonlinear impact on the result. After the adjoint is computed, there is a small CPU time overhead that is required to obtain the flow

sensitivities by taking the dot product of the adjoint and the grid sensitivity for each DV. The in-core memory for ADJIFORed and ADIFORed codes are comparable. However, the disk-storage required by the ADJIFORed code can become very large because of the log files created to store the intermediate values.



With the aid of parallel processing, it is possible to distribute the memory needed, as well as the I/O load, over many PEs.

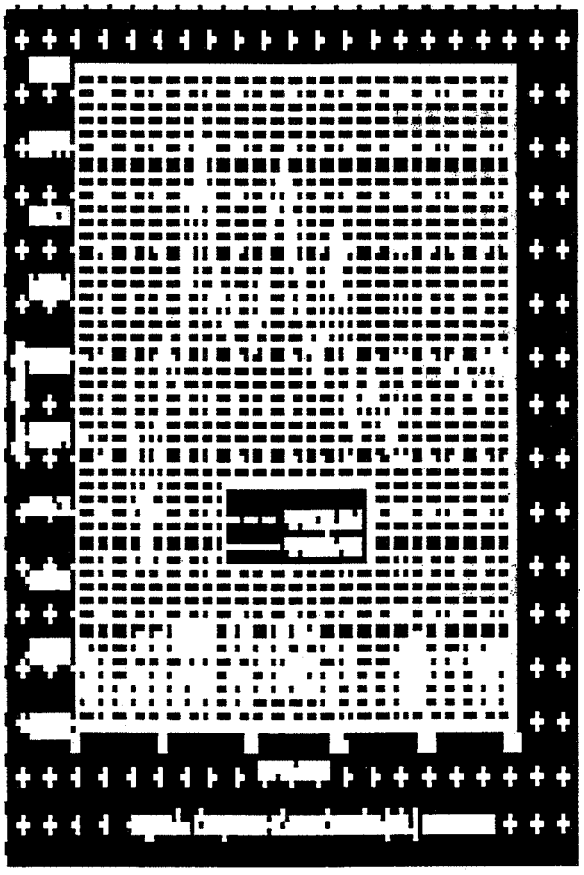
CFL3Dhp has been ADJIFORed to produce CFL3Dhp.ADJI[3]. For a grid with 0.5 million points, the CPU time to obtain the adjoint solution using CFL3Dhp.ADJI is nearly twenty times the function evaluation time on a 24-PE Origin 2000; merely 15 single-PE hours. The temporary log file for this problem is nearly 16 Gigabytes. However, since the I/O load is shared by the 24 PEs, it is not a bottleneck. Once the adjoint is calculated, the dot product of the adjoint with the grid sensitivities is computed and accumulated in 3 minutes of single PE CPU time. The CPU time required to calculate the flow sensitivities using CFL3Dhp.ADJI is 25-times less than that when using CFL3Dhp.ADII. Figure 4 compares the sensitivity calculated by these two methods.

Aerodynamic Optimization Using Analytical Sensitivities

To perform nonlinear shape optimization, the AERODynamic SHape OPtimization (AEROSHOP) system developed at BPW is used. AEROSHOP is a user-friendly, script-based, modular

optimization tool that adapts to any CFD environment, and runs on many serial and parallel platforms. The sensitivities computed using either the CFL3Dhp.ADII forward-mode or the CFL3Dhp.ADJI reverse-mode flow sensitivities are easily incorporated into AEROSHOP.

To demonstrate the advantages of performing design optimization in a parallel environment, a wing/body design was performed using more than 400 DVs and 55 constraints. The resulting, realistic design has more than a 5% increase in aerodynamic performance over the linear-theory-based configuration. In addition, this design was obtained fairly quickly using the parallel platform.



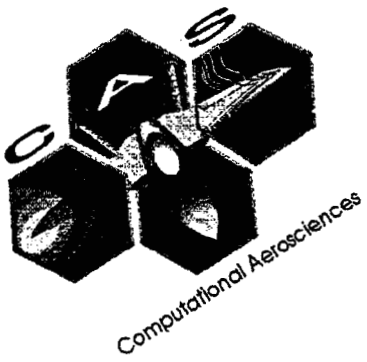
Conclusion and Future Work

The parallel computers available through HPCCP have vastly improved the CFD capabilities at Boeing-PW. First, it is possible to obtain a full-configuration, Navier-Stokes drag polar in a day. Second, the innovative application of ADIFOR-generated sensitivity codes makes this technique practical for the first time. Euler-based aerodynamic shape optimization can now be performed on complex configurations with a large number of design variables with good turn-around time. Larger analysis problems and Navier-Stokes-based designs are planned in the future.

References

1. Christial Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer, "ADIFOR 2.0: Automatic Differentiation of Fortran 77 Programs", IEEE Computer Science and Engineering, 3(3), 18-22, Fall 1996.
2. Arthur Taylor – Private Communications, 1998.
3. Alan Carle, Mike Fagan, and Lawrence Green, "Preliminary Results from the Application of Automatic Adjoint Code Generation to CFL3D", CRPC-TR98741, February 1998.

OMIT THIS PAGE



Session 10:

Parallel System Software Technology

A ROBUST AND SCALABLE SOFTWARE LIBRARY FOR PARALLEL ADAPTIVE REFINEMENT ON UNSTRUCTURED MESHES

John Z. Lou, Charles D. Norton, and Thomas A. Cwik
National Aeronautics and Space Administration
Jet Propulsion Laboratory, California Institute of Technology
MS 168-522, 4800 Oak Grove Drive, Pasadena, CA 91109-8099, U.S.A.
{John.Lou, Charles.Norton, Thomas.Cwik}@jpl.nasa.gov

Abstract

The design and implementation of **Pyramid** (http://www-hpc.jpl.nasa.gov/APPS/AMR/), a software library for performing parallel adaptive mesh refinement (PAMR) on unstructured meshes, is described. This software library can be easily used in a variety of unstructured parallel computational applications, including parallel finite element, parallel finite volume, and parallel visualization applications using triangular or tetrahedral meshes. The library contains a suite of well-designed and efficiently implemented modules that perform operations in a typical PAMR process. Among these are mesh quality control during successive parallel adaptive refinement (typically guided by a local-error estimator), parallel load-balancing, and parallel mesh partitioning using the ParMeTiS partitioner. The Pyramid library is implemented in Fortran 90 with an interface to the Message-Passing Interface (MPI) library, supporting code efficiency, modularity, and portability. An EM waveguide filter application, adaptively refined using the Pyramid library, is illustrated.

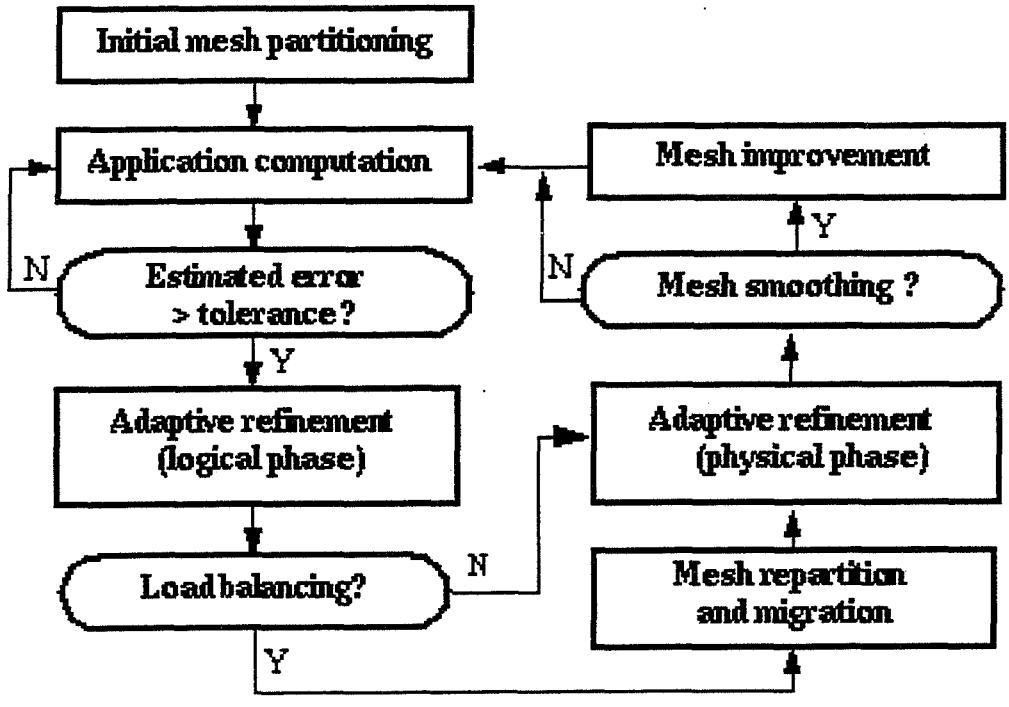
1. Introduction

Adaptive mesh refinement (AMR) represents a class of numerical techniques that has demonstrated great effectiveness for a variety of computational applications including computational physics, structural mechanics, electromagnetics, and semiconductor device modeling. When an application domain is discretized into a computational mesh, various portions of the mesh can be refined, or coarsened, in regions where varying degrees of accuracy are required. This approach saves memory and computing time over methods that use a uniform resolution over the entire application domain.

Unfortunately, the development of an efficient and robust adaptive mesh refinement component for an application, particularly for unstructured meshes on multiprocessor systems, is very complex. The motivation for our work is to provide an efficient and robust parallel AMR library that can be easily integrated into unstructured parallel applications. Therefore, our library approach separates support for parallel adaptive refinement and mesh maintainence techniques from any application-specific solution processes.

Research on parallel AMR for unstructured meshes has been previously reported [1,8]. Most efforts are based on C++, and many realize that mesh quality control during successive adaptive refinement is an active research topic. Our work features the use of Fortran 90 and MPI for parallel AMR on unstructured triangular and tetrahedral meshes, the implementation of robust schemes for parallel adaptive refinement and mesh quality control during a repeated AMR process, and a "plug-in" component for stiffness matrix construction in finite element applications. We selected

Fortran 90 for our implementation because it provides new abstraction modeling facilities beneficial for parallel unstructured AMR development. This approach also simplifies interface concerns with scientific application codes, many of which were developed in Fortran 77 for high performance.



General organization of the parallel AMR process for unstructured meshes.

2. The AMR Components

The general organization of the parallel AMR process is illustrated. Initially, the (generally random) input mesh must be repartitioned and redistributed after loading from the disk. The application computation and local error-estimation step occur, after which a logical AMR process occurs. (Load balancing can occur based on this process since the refinement scheme is completely defined, although it has not yet physically occurred.)

The load balancing process moves coarse elements from the logical refinement, based on a weighting scheme, to the proper destination processors using the migration module. At this point, the physical AMR step occurs by applying local refinement processes.

Finally, the element quality can be checked by performing an explicit mesh smoothing operation or by ensuring high quality element creation during refinement. We apply the latter approach since it prevents degradation of mesh quality after successive adaptive refinements. Every stage of our AMR process is performed using parallelism.

3. Fortran 90 and Abstraction Modeling Principles in Parallel AMR Development

Fortran 90 modernizes traditional Fortran 77 scientific programming by adding many new features. These features allow programs to be designed and written at a higher level of abstraction, while increasing software clarity and safety without sacrificing performance [7]. Fortran 90's capabilities encourage scientists to design new kinds of advanced data structures supporting complex applications, like parallel AMR. These capabilities extend beyond the well-known array syntax and dynamic memory management operations.

While Fortran 90 is not an object-oriented language (certain OO features can be emulated by software constructs) the methodology simplifies library interfaces such that the internal details are hidden from library users [5]. Fortran 90 modules and derived types allow user-defined types, like the mesh, to be defined with associated routines. Modules that capture essential features of parallel AMR can be combined with each other, adding to the flexibility and organization of the software design. These techniques, and other features, allow the library to contain clearly organized interfaces for use in parallel applications.

4. The Adaptive Refinement Process

The adaptive refinement process is based on logical and physical refinements. The logical refinement step uses an iterative procedure that traverses through elements of the coarse mesh repeatedly to "define" a consistent mesh refinement pattern on the coarse mesh. The result of the logical refinement is stored in the data structure of the coarse mesh, which completely specifies whether and how each element in the coarse mesh should be refined. Our adaptive refinement scheme is based on "edge-marking" for both triangular and tetrahedral meshes. Starting from a predetermined subset of elements, the logical refinement scheme proceeds by marking (or logically refining) element edges wherever necessary, and the refinement pattern for each element is determined by the number of marked edges in that element.

With information generated from the logical refinement step, the actual mesh refinement becomes conceptually simpler, since it is completely specified how each element should be refined. To make the physical refinement process simpler and efficient, low-level objects are refined before refining high-level objects. On a triangular mesh, it means edges are refined before refining elements.

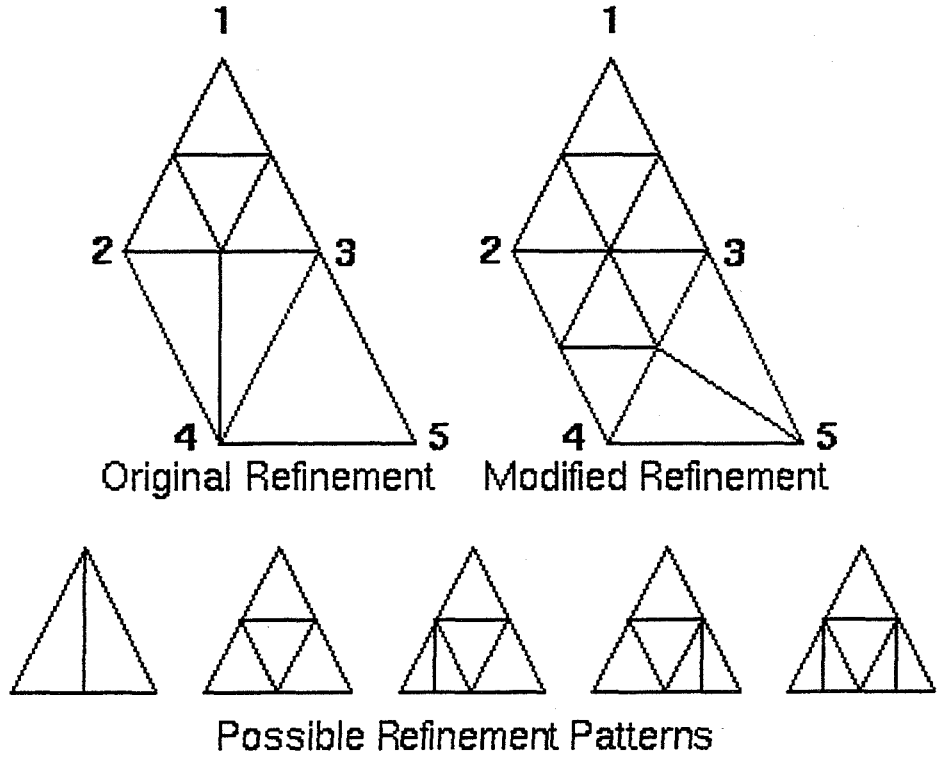
To perform a parallel logical adaptive refinement, we extend the serial scheme so that after traversing the local element set for edge-marking, each processor updates the status of edges on mesh partition boundaries by exchanging the edge status information (i.e. marked or not marked) with its neighboring processors.

5. Mesh Quality Control

A problem associated with repeated AMR operations, typically guided by a local-error estimator, is the deterioration of mesh quality. Most mesh smoothing schemes tend to change the structure of a given mesh to achieve the "smoothing effect" by rearranging nodes in the mesh. The changes made by a smoothing scheme, however, could modify the desired distribution of element density produced by the AMR procedure, and the cost of performing a global mesh smoothing could be very high. Nevertheless, applying a relatively efficient smoothing scheme over the last adaptively refined mesh is probably reasonable for mesh quality improvement. Alternatively, it is possible to

prevent, or slow down, the degradation of element quality during a repeated adaptive refinement process.

The mesh quality control scheme we have applied classifies elements based on how they were refined. This allows us to forsee the potential of creating elements with poor aspect ratios in the next refinement. After identifying those elements, we can replace them with a refinement pattern that improves upon the geometry.



The figure shows the original refinement of a coarse element (2-3-4). Successive refinements will destroy the aspect ratio of existing elements, leading to poor mesh quality. The approach we apply modifies the coarse element refinement, as shown, should either of the child elements require further refinement (due to local errors or mesh consistency constraints from neighbor element refinement). This process controls the mesh quality, at the slight expense of creating more elements.

We integrate the mesh quality control feature into our adaptive refinement scheme, for triangular meshes, by defining all possible refinement patterns for a pair of "twin" transitional elements (child elements of element 2-3-4 in the original mesh). During the logical refinement step the scheme checks all marked edges of the twin elements allowing one of the indicated refinement patterns to be selected. To simplify the physical refinement stage we ensure that the partitioner will not place the twin elements onto different processors. This guarantees that once mesh migration has occured, the physical refinement for the parent element (2-3-4) will create child elements in a local manner. Refinement patterns are also applied for tetrahedral meshes as well.

6. Interlanguage Communication and Load Balancing Issues

Our software needs to communicate with the ParMeTiS parallel mesh partitioner, which is written in the C programming languauge [6]. We have a single routine that acts as the conduit between our Fortran 90 system and the C ParMeTiS library. Interlanguage communication between Fortran 90 and C is not a problem, provided the linker knows the format of external routine names (generally, underscore_, doubleunderscore__, UPPERCASE, or lowercase). Fortran 90 derived type

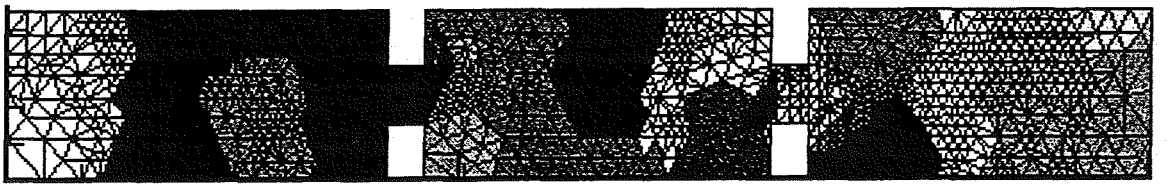
objects can be passed by reference to C structures, or simple arrays can be communicated, but we advise using the Fortran 90 SEQUENCE attribute to request the proper byte-alignment ordering. The weighted graph helps ParMeTiS attempt to minimize element movement and the number of components on partition boundaries.

This process involves converting the distributed mesh into a distributed graph by computing the dual of the mesh. When the partitioner returns, a mapping for every element is specified. The migration module redistributes the elements among the processors based on this mapping. Since the communication is irregular, and unpredictable, an efficient non-blocking irregular communication scheme has been developed for the element redistribution. In the final stage, the mesh data structure is reconstructed using efficient heap-sorting techniques. This entire process occurs in a parallel and distributed manner.

7. Application to an EM Waveguide Filter

Our AMR library has been tested in the finite-element simulation of electromagnetic wave scattering in a waveguide filter [4]. The problem is to solve Maxwell's equation for the electromagnetic (EM) fields in the filter domain. A local-error estimate procedure based on the Element Residue Method (ERM) is used in combination with the AMR technique to adaptively construct an optimal mesh for the problem solution.

The adaptive refinement and partitioning of a finite element mesh for EM scattering in the waveguide filter is illustrated on 16 processors of the NASA Goddard Cray T3E. An example of adaptive refinement for a 3D tetrahedral mesh is available.



Adaptive refinement, mesh partitioning, and migration applied to a waveguide filter.

8. Summary

A complete framework for performing parallel adaptive mesh refinement in unstructured applications on multiprocessor computers has been described. This includes a robust parallel AMR scheme, mesh quality control, load-balancing, the implementation technique using Fortran 90 and MPI, and the interlanguage communication issues. Electromagnetic scattering in a waveguide filter has been demonstrated. Parallel performance results on several multiprocessor systems will be given in our final paper. More information on Pyramid: A JPL Parallel Unstructured AMR Library is also available.

The Table 1 gives performance results of the AMR (logical and physical) step and the load balancing and migration step. The refinement randomly chooses half of the elements per processor. The number of elements increases with the partitioning slightly due to maintinain mesh consistency constrains based on this refinement scheme.

Acknowledgments

The research described in this paper was performed at Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration. The supercomputers used in this work were provided with funding from the NASA offices of Space Science, Aeronautics, and Mission to Planet Earth.

Table 1: Results for Waveguide Filter after 3 Refinements on the NASA Goddard Cray T3E

# of Processors	AMR Time	Load Balancing (Migration) Time	Number of Elements
32	57.34 sec	15.36 sec	292,612
64	13.55 sec	3.75 sec	295,405
128	2.93 sec	1.65 sec	305,221
256	0.54 sec	1.51 sec	335,527
512	0.27 sec	1.86 sec	397,145

References

1. R. Biswas, L. Oliker, and A. Sohn. "Global Load-Balancing with Parallel Mesh Adaption on Distributed-Memory Systems." Proceedings of Supercomputing '96, Pittsburgh, PA, Nov. 1996.

2. E. Boender. "Reliable Delaunay-Based Mesh Generation and Mesh Improvement." Communications in Numerical Methods in Engineering, Vol. 10, 773-783 (1994).

3. Graham F. Carey, "Computational Grid Generation, Adaptation, and Solution Strategies". Series in Computational and Physical Processes in Mechanics and Thermal Science. Taylor & Francis, 1997.

4. T. Cwik, J. Z. Lou, and D. S. Katz, "Scalable Finite Element Analysis of Electromagnetic Scattering and Radiation." to appear in Advances in Engineering Software, V. 29 (2), March, 1998.

5. V. K. Decyk, C. D. Norton, and B. K. Szymanski. "Expressing Object-Oriented Concepts in Fortran 90". ACM Fortran Forum, vol. 16, num 1, pp. 13-18, April 1997. Also as NASA Tech Briefs, Vol. 22, No. 3, pp 100-101, March 1998 (reduced version).

6. G. Karypis, K. Schloegel, and V. Kumar. "ParMeTiS: Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 1.0". Tech. Rep., Dept. of Comp. Science, U. Minnesota, 1997.

7. C. Norton, V. Decyk, and B. Szymanski. "High Performance Object-Oriented Scientific Programming in Fortran 90". Proc. 8th SIAM Conf. on Parallel Proc. for Sci. Comp., Mar. 1997.

8. M. Shephard, J. Flaherty, C. Bottasso, H. de Cougny, C. Ozturan, and M. Simone. "Parallel automatic adaptive analysis". Parallel Computing 23 (1997) pg. 1327-1347.

544-42
018279
ABS. ONLY

PARALLEL GRID MANIPULATIONS IN EARTH SCIENCE CALCULATIONS

W. Sawyer, R. Lucchesi, A. da Silva, L' L. Takacs
Data Assimilation Office, NASA GSFC, Code 910.3, Greenbelt MD, 20771
E-mail: sawyer@dao.gsfc.nasa.gov, *Phone:* 301 286 6543

366905

The National Aeronautics and Space Administration (NASA) Data Assimilation Office (DAO) at the Goddard Space Flight Center is moving its data assimilation system to massively parallel computing platforms. This parallel implementation of GEOS DAS will be used in the DAO's normal activities, which include reanalysis of data, and operational support for flight missions. Key components of GEOS DAS, including the gridpoint-based general circulation model and a data analysis system, are currently being parallelized. The parallelization of GEOS DAS is also one of the HPCC Grand Challenge Projects.

The GEOS-DAS software employs several distinct grids. Some examples are: an *observation grid*— an unstructured grid of points at which observed or measured physical quantities from instruments or satellites are associated—a highly-structured *latitude-longitude grid* of points spanning the earth at given latitude-longitude coordinates at which prognostic quantities are determined, and a *computational lat-lon grid* in which the pole has been moved to a different location to avoid computational instabilities. Each of these grids has a different structure and number of constituent points. In spite of that, there are numerous interactions between the grids, e.g., values on one grid must be interpolated to another, or, in other cases, grids need to be redistributed on the underlying parallel platform.

The DAO has designed a parallel integrated library for grid manipulations (PILGRIM) to support the needed grid interactions with maximum efficiency. It offers a flexible interface to generate new grids, define transformations between grids and apply them. Basic communication is currently MPI, however the interfaces defined here could conceivably be implemented with other message-passing libraries, e.g., Cray SHMEM, or with shared-memory constructs. The library is written in Fortran 90.

First performance results (Figure 1) indicate that even difficult problems, such as above-mentioned pole rotation—a sparse interpolation with little data locality between the physical lat-lon grid and a pole rotated computational grid—can be solved efficiently and at the GFlop/s rates needed to solve tomorrow's high resolution earth science models. In the subsequent presentation we will discuss the design and implementation of PILGRIM as well as a number the problems it is required to solve. Some conclusions will be drawn about the potential performance of the overall earth science models on the supercomputer platforms foreseen for these problems.

I/O Parallelization for the Goddard Earth Observing System Data Assimilation System (GEOS DAS)

R. Lucchesi, W. Sawyer†, L. L. Takacs, P. Lyster†, J. Zero
Data Assimilation office
NASA/GSFC, Code 910.3
Greenbelt MD, 20771
Email: lucchesi@dao.gsfc.nasa.gov
† Additional affiliation: University of Maryland, College Park, 20742

July 21, 1998

Abstract

The National Aeronautics and Space Administration (NASA) Data Assimilation Office (DAO) at the Goddard Space Flight Center (GSFC) has developed the GEOS DAS, a data assimilation system that provides production support for NASA missions and will support NASA's Earth Observing System (EOS) in the coming years. The DAO's support of the EOS project along with the requirement of producing long-term reanalysis datasets with an unvarying system levy a large I/O burden on the future system. The DAO has been involved in prototyping parallel implementations of the GEOS DAS for a number of years and is now converting the production version from shared-memory parallelism to distributed-memory parallelism using the portable Message-Passing Interface (MPI). If the MPI-based GEOS DAS is to meet these production requirements, we must make I/O from the parallel system efficient.

We have designed a scheme that allows efficient I/O processing while retaining portability, reducing the need for post-processing, and producing data formats that are required by our users, both internal and external. The first phase of the GEOS DAS Parallel I/O System (GPIOS) will expand upon the common method of gathering global data to a single PE for output. Instead of using a PE also tasked with primary computation, a number of PEs will be dedicated to I/O and its related tasks. This allows the data transformations and formatting required prior to output to take place asynchronously with respect to the GEOS DAS assimilation cycle, improving performance and generating output data sets in a format convenient for our users. I/O PEs can be added as needed to handle larger data volumes or to meet user file specifications. We will show I/O performance results from a prototype MPI GCM integrated with GPIOS. Phase two

of GPIOS development will examine ways of integrating new software technologies to further improve performance and build scalability into the system. The maturing of MPI-IO implementations and other supporting libraries such as parallel HDF should provide performance gains while retaining portability.

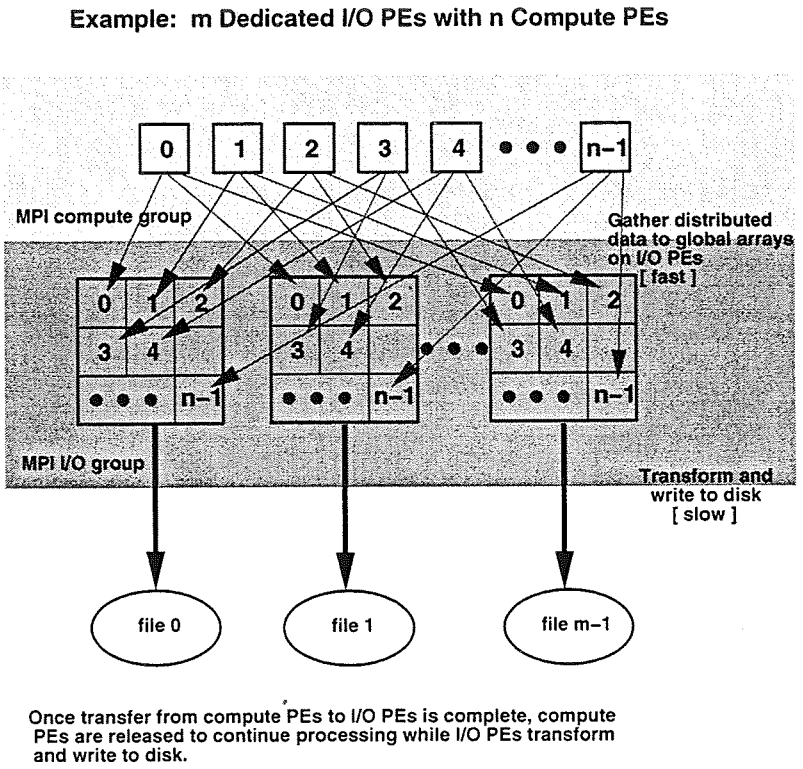


Figure 1: In the GPIOS parallel I/O concept, PEs are split into compute and I/O nodes. The top group of PEs is dedicated to the primary computation, while the middle row depicts a pool of I/O PEs. For output, each I/O PE writes to an independent file. While data input can be performed in an analogous manner, the amount of input data are small in comparison to the output data.

PORTABILITY AND CROSS-PLATFORM PERFORMANCE OF AN MPI-BASED PARALLEL POLYGON RENDERER

Thomas W. Crockett

Institute for Computer Applications in Science and Engineering
MS 403, NASA Langley Research Center, Hampton, VA 23681-2199
tom@icase.edu / (757) 864-2182

1. Introduction

Visualizing the results of computations performed on large-scale parallel computers is a challenging problem, due to the size of the datasets involved. One approach is to perform the visualization and graphics operations in place, exploiting the available parallelism to obtain the necessary rendering performance. Over the past several years, we have been developing algorithms and software to support visualization applications on NASA's parallel supercomputers. Our results have been incorporated into a parallel polygon rendering system called *PGL* [1].

PGL was initially developed on tightly-coupled distributed-memory message-passing systems, including Intel's iPSC/860 and Paragon, and IBM's SP2. Over the past year, we have ported it to a variety of additional platforms, including the HP Exemplar, SGI Origin2000, Cray T3E, and clusters of Sun workstations. In implementing PGL, we have had two primary goals: cross-platform portability and high performance. Portability is important because (1) our manpower resources are limited, making it difficult to develop and maintain multiple versions of the code, and (2) NASA's complement of parallel computing platforms is diverse and subject to frequent change. Performance is important in delivering adequate rendering rates for complex scenes and ensuring that parallel computing resources are used effectively. Unfortunately, these two goals are often at odds. In this paper we report on our experiences with portability and performance of the PGL polygon renderer across a range of parallel computing platforms.

2. The Application

Parallel rendering is a demanding application which is both computation- and communication-intensive. The heart of the rendering computation is a projection from three-dimensional object space onto a two-dimensional image plane. Extensive parallelism is available in both spaces, but exploiting it requires massive interprocessor communication [2]. In PGL, objects are modeled as collections of triangular facets. Variations in the size and distribution of these facets, along with changes in scene content and viewing parameters, lead to computations which can be highly irregular. The resulting communication patterns are data-dependent and dynamic, but can generally be characterized as many-to-many or all-to-all.

These considerations have led us to develop multiplexed asynchronous algorithms which use non-blocking communication and buffering to minimize idle time, provide latency tolerance, reduce contention, and amortize start-up costs [3,4]. Conceptually, there are three activities that proceed concurrently on each processor: a transformation phase, which projects object-space primitives

This work was supported by the National Aeronautics and Space Administration under Contract Nos. NAS1-18605, NAS1-19480, and NAS1-97046 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), M/S 403, NASA Langley Research Center, Hampton, VA 23681-2199.

into screen space; a rasterization phase, which computes individual pixel values; and a termination detection algorithm. The latter is necessary since an individual processor has no way to determine *a priori* how many screen-space primitives will be sent to it for rasterization. In practice, these three tasks are implemented as a single thread of computation with periodic polling to consume incoming messages.

To achieve portability, PGL is written in ANSI C, using the MPI message-passing standard for interprocessor communication.[1] MPI is the only mechanism currently available which will support this type of algorithm across a wide range of parallel architectures.

3. Performance Comparisons

3.1 Methodology. We have implemented both serial and parallel versions of PGL and its associated benchmark programs. The serial and parallel rendering computations are practically identical, except that the serial version of the code omits the extra control and communication logic needed in the parallel case. When porting PGL from one platform to another, we modify the code base as little as possible, making only those changes needed to assure successful compilation and correct operation. Non-portable functions (e.g., high resolution clock timers) are abstracted into platform-specific header files. For each platform, we run extensive tests with the serial code to determine the best settings for compiler optimization flags. These settings are then used for the parallel code as well. We make no effort to tune the parallel code for specific platforms, except to set runtime flags or environment variables as needed for correct operation or as recommended by the vendor.

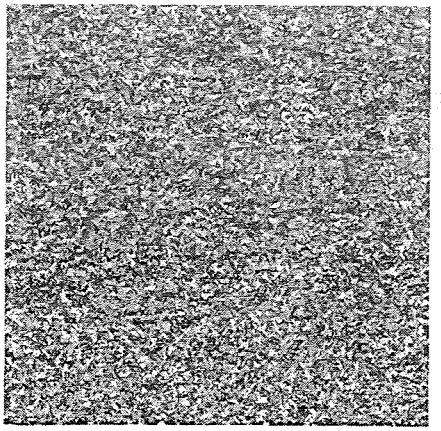
The results presented here span four generations of parallel computer architecture. During that time, PGL has been under continuous development, so that recent performance numbers reflect improvements not only in hardware, but also in algorithms and compilers. Results for the Paragon, T3E, Origin2000, and Exemplar are contemporary, reflecting essentially the same version of the code. The SP2 and iPSC/860 results were obtained using an older, less efficient termination detection algorithm; the iPSC/860 results also lack several improvements to the sequential rendering routines.

We have chosen 128 processors to be our primary configuration for benchmarking purposes. We consider this to be no more than a mid-range system by today's standards, but it is large enough to determine scalability trends. Our standard benchmark scene, chosen because of its convenient analytical properties, consists of 100,000 equal-sized triangles randomly oriented within a cubic volume (Fig. 1a).

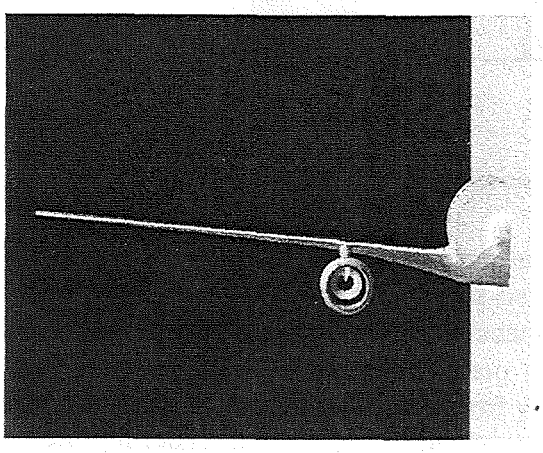
We repeat each experiment four times at each data point. The first pass ensures that code and data pages are loaded into memory and that caches are primed. We ignore the time required for this pass and report the average of the three subsequent passes. Parallel efficiencies are computed relative to the serial implementation, rather than to the parallel implementation running on a single processor.

3.2 Uniprocessor results. Figure 2 illustrates comparative performance of the serial rendering code for the three scenes shown in Figure 1. iPSC/860 results are available only for the random

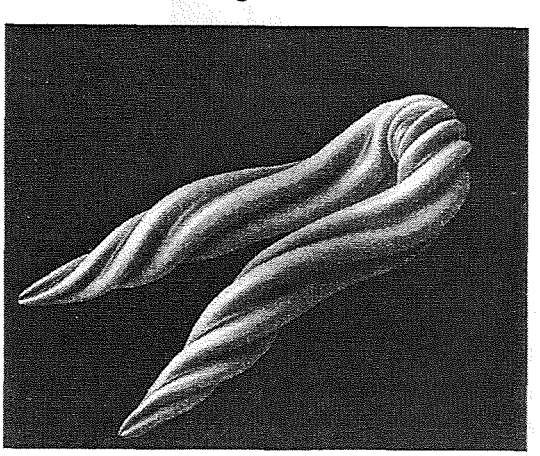
1. PGL implementations on the iPSC/860 and Paragon predate the development of MPI, and are based instead on Intel's NX message-passing library, which provides comparable functionality.



(a) *rand100k:* 100,000 random triangles, 512 x 512 image resolution.



(b) *lwt:* CFD boundary grid, 31,271 triangles, 800 x 640.



(c) *ropes:* hairpin vortex visualization, 245,616 triangles, 800 x 640.

Figure 1. Benchmark scenes.

triangle scene. As can be seen, the 250 MHz MIPS R10000 processors used in the Origin2000 offer a clear performance advantage in this application.

3.3 Parallel results. Figure 3 illustrates comparative performance as a function of message length for the random triangle scene using 128 processors.[2] Message length is determined by the number of data items which are buffered together for transmission. The maximum (54 in this case) is scene-dependent and is based on the expected number of data items which will be sent from a typical processor to each of the others. Individual data items are 24 bytes long, except on the T3E, where differences in word length and structure padding dictate 32-byte items.

Figure 4 shows parallel efficiencies from 2 to 128 processors, based on performance at the optimum message length for the number of processors in use.[3] Results for the four distributed-mem-

2. The Origin2000 line currently supports a maximum of 128 processors. To avoid interference from operating system processes, we employed only 120 processors in our tests on that platform.

3. Determining the optimum message length is an interesting problem in itself, but limitations on space prevent us from examining that here.

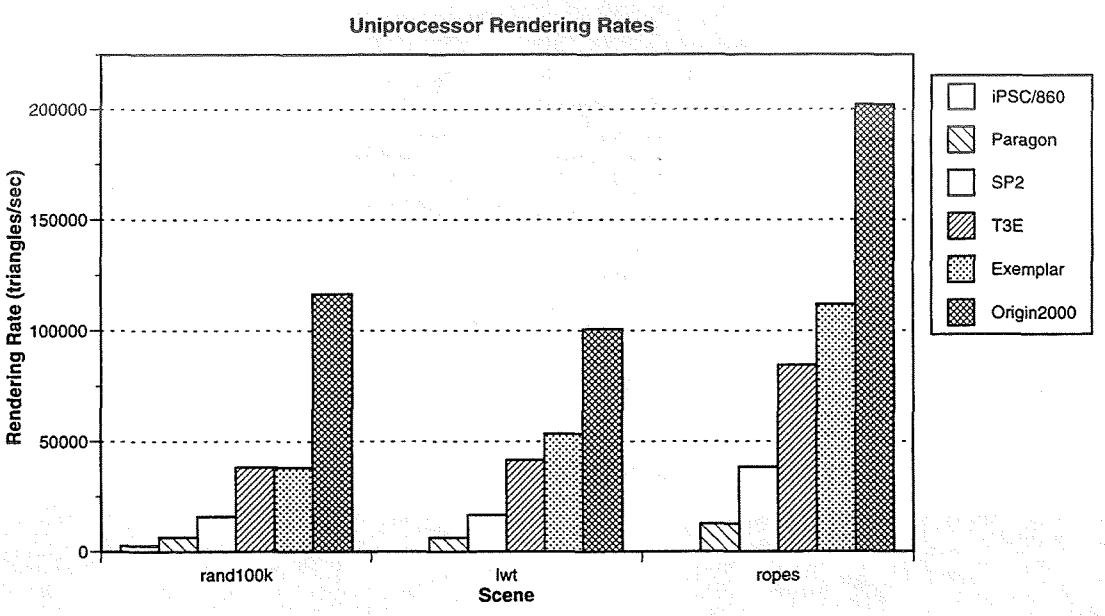


Figure 2. Comparative performance of the serial renderer.

ory systems (iPSC/860, Paragon, SP2, and T3E) are remarkably similar, with the Paragon demonstrating the best scalability.[4] The two distributed-shared-memory (DSM) systems (Exemplar and Origin2000) are also similar in their performance trends. The Exemplar shows strongly superlinear performance in small configurations, but performance degrades rapidly beyond 32 processors. The Origin2000 pays a high penalty going from the serial code to its parallel equivalent, and also scales poorly beyond 32 processors. Consequently, the highest rendering rates with 120–128 processors are obtained with the T3E (Fig. 3), even though it has to move one-third more data than any of the other systems, and its uniprocessor performance is only a third that of the Origin2000.

3.4 Discussion. Our asynchronous message-passing strategy works reasonably well on distributed-memory systems, with the primary impediment to scalability being the high software overheads associated with message transmission and reception. Although we expected that message-passing would not be the optimal programming paradigm on newer DSM architectures, we nevertheless thought that shared memory support would lead to efficient MPI implementations which would compare favorably with those on distributed-memory systems. Our results indicate that with large numbers of processors, this is not the case. Detailed internal measurements show that message-passing overheads are much higher and much more variable on the DSM architectures, with the variability introducing more idle time. Several factors seem to contribute to this poor performance, including inefficient MPI implementations, contention for shared data structures, memory management and process scheduling issues, and the absence of message co-processors. Polling operations appear to place particularly heavy demands on the system. Experiments conducted in-house at HP using our code indicate that reducing the number of *MPI_Test()* calls can

4. Scalability results for the iPSC/860 and SP2 at 128 processors would be somewhat better with the current termination detection algorithm, leaving the T3E as the least scalable of the four distributed-memory systems.

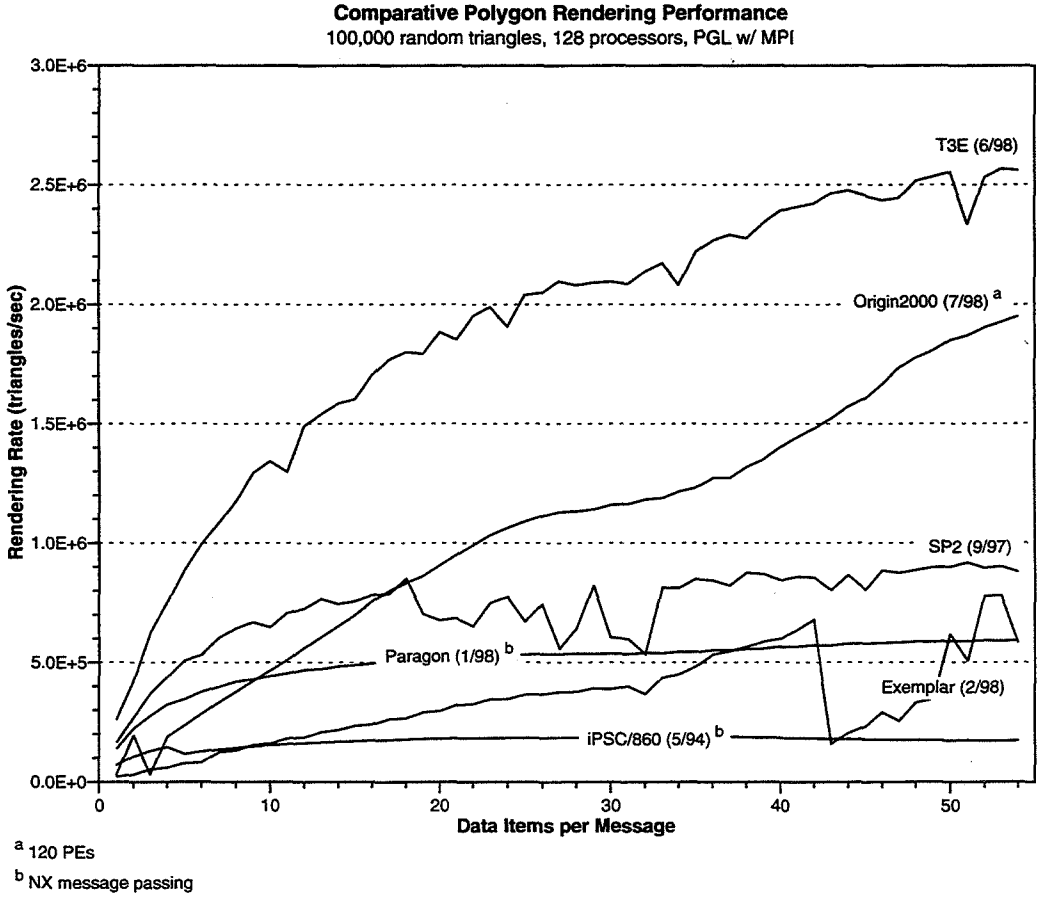


Figure 3. Comparative rendering rates for 100,000 random triangles with 128 processors.

boost performance by about a factor of two. We are currently investigating ways of reducing the polling frequency while maintaining a practical, deadlock-free algorithm.

4. Summary

We have examined the performance of a parallel polygon renderer on six different parallel architectures. The four distributed-memory systems exhibit reasonably good scalability on a fixed-size problem through 128 processors, while the two distributed-shared-memory systems scale very poorly to this level. While improvements to vendor-supplied MPI implementations and tuning of the parallel rendering algorithms may increase performance on the DSM platforms, it appears doubtful that a message-passing paradigm will be able to achieve the desired efficiencies. We therefore plan to develop an explicit shared-memory version of the code in an effort to reduce communication overheads to a minimum. The results of that experiment will help us to determine whether the current performance problems on DSM systems are due to MPI, or whether they reflect fundamental limitations of the hardware and software architectures.

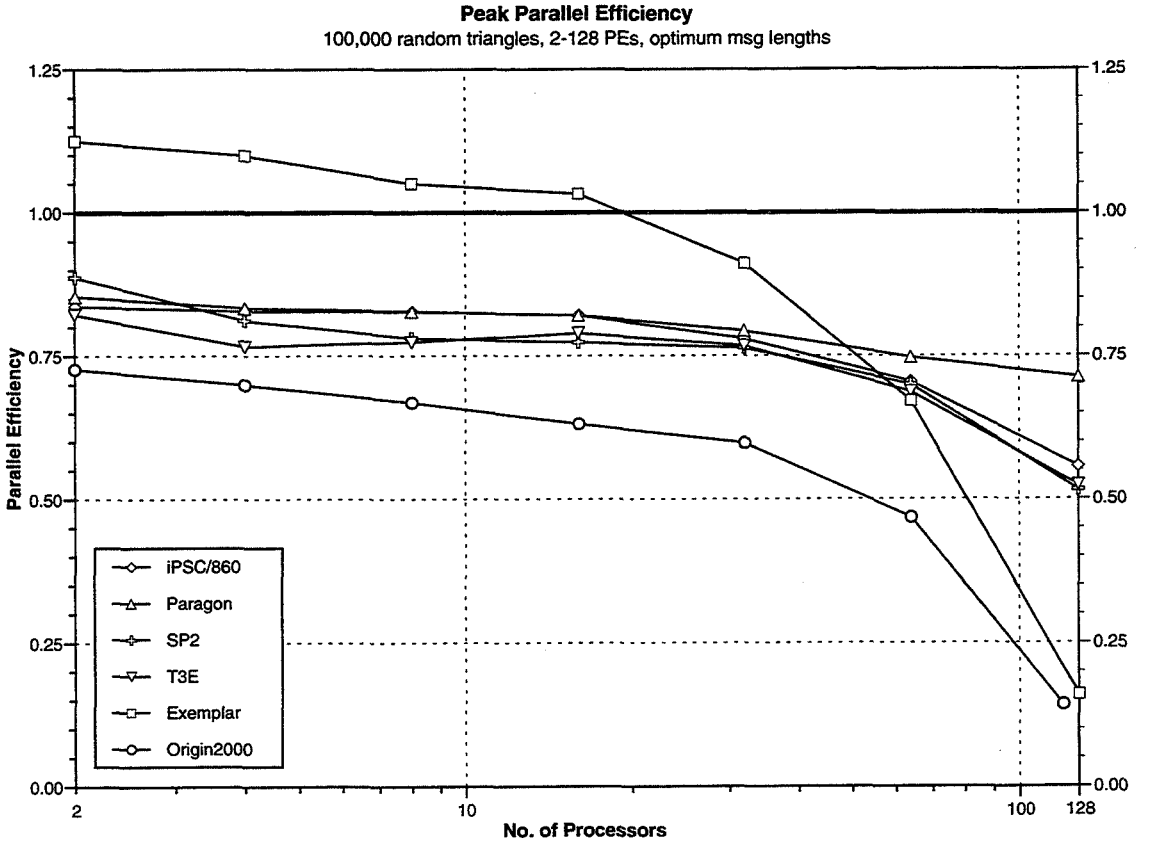


Figure 4. Parallel efficiencies from 2 to 128 processors using optimum message lengths.

Acknowledgments

The experiments described here were performed on computer systems provided by NASA's HPCCP/CAS Project and by Caltech's Center for Advanced Computing Research. We thank the system support personnel at NASA Langley, NASA Ames, NASA Goddard, and Caltech for assistance in using their facilities. Jimmy Guo at Hewlett-Packard and Bron Nelson at Silicon Graphics also provided valuable insights and assistance. Data for the benchmark scenes was contributed by Toby Orloff, Dimitri Mavriplis, and David Banks.

References

[1] T. W. CROCKETT. PGL: a parallel graphics library for distributed memory applications, ICASE Interim Report 29, http://www.icase.edu/reports/interim/29/, Feb. 1997.

[2] T. W. CROCKETT. Parallel rendering, in *Encyclopedia of Computer Science and Technology*, Vol. 34, Suppl. 19, A. Kent and J. G. Williams, eds., Marcel Dekker, 1996, pp. 335-371.

[3] T. W. CROCKETT and T. ORLOFF. Parallel polygon rendering for message-passing architectures, *IEEE Parallel and Distributed Technology*, 2(2), Summer 1994, pp. 17-28.

[4] T. W. CROCKETT. Design considerations for parallel graphics libraries, ICASE Report No. 94-49 (NASA CR 194935), ftp://ftp.icase.edu/pub/techreports/94/94-49.pdf, June 1994.

Parallel Visualization Co-Processing of Overnight CFD Propulsion Applications

David E. Edwards
Pratt & Whitney
400 Main Street, M/S 163-17
East Hartford, CT 06108
(860) 565-5591
edwardde@pweh.com

Robert Haimes
Massachusetts Institute of Technology
77 Mass Ave
Cambridge, MA 02139
(617)253-7518
haimes@orville.mit.edu

Abstract

An interactive visualization system pV3 is being developed for the investigation of advanced computational methodologies employing visualization and parallel processing for the extraction of information contained in large-scale transient engineering simulations. Visual techniques for extracting information from the data in terms of cutting planes, iso-surfaces, particle tracing and vector fields are included in this system. This paper discusses improvements to the pV3 system developed under NASA's Affordable High Performance Computing project.

Introduction

A goal of the Affordable High Performance Computing (AHPC) Project is the achievement of overnight turnaround of large three-dimensional aerospace CFD simulations. This required that improvements to the processing of information created from the numerical simulations be made. To address this problem a team from Pratt & Whitney, Massachusetts Institute of Technology (MIT), CFDRC and NASA was formed. Post-processing software that supports large scale propulsion simulation applications on clustered workstations in the design environment was to be developed. A requirement of the software was for it to to be made fast and efficient to enable rapid design evaluation. A decision was made to build the visualization system using the MIT pV3 [1,2] as the basis system.

pV3 is a visualization turnkey system for the examination of data that is either structured or unstructured. The system can process volumetric data that is represented as tetrahedral, pyramid, prism or hexahedral cells. The pV3 system also accepts structured multi-block data with PLOT3D blanking.

Specific enhancements to pV3 include:

- Motif Graphical User Interface
- Batch collector/playback viewer
- Multi-disciplinary viewing of multiple scalar and vector fields
- Solution Sub-sectioning/Replication
- Annotation
- MPEG Animation
- Collaboration Capability

The remainder of this paper will describe the new features in the visualization system.

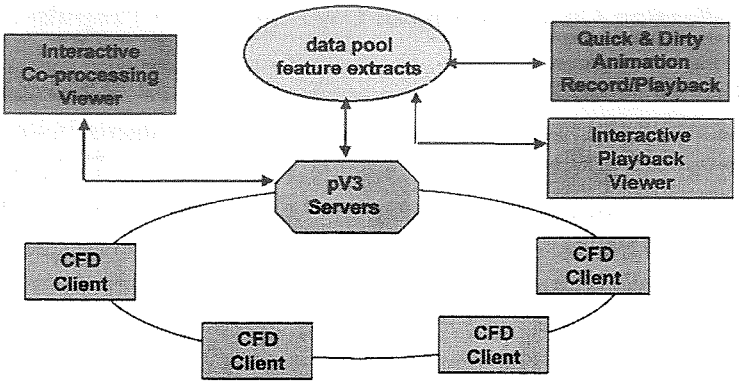


Figure 1 – AHPC Project Visualization System

AHPC Project Visualization System

The visualization system developed in this program system is shown in Fig. 1. The pV3 system consists of a collection of multiple pV3 clients which contains tools to extract point, lines and surface information from a volumetric database. The clients can either be attached directly to the solvers or used to read solution files. Requests for specific information comes from a pV3 server that can run in either an interactive mode or a batch mode. The batch mode stores requested extracts to disk that can be displayed using an interactive playback viewer. Animation can be stored as MPEG files from either the interactive viewer or the playback viewer.

Motif Graphical User Interface

At the onset of this program, a recognized deficiency in the pV3 system was its graphical user interface (GUI). The original pV3 GUI shown in Fig. 2 is divided into six different windows (3D Graphics, 2D Graphics, 1D Graphics, Key, Dial and Text). The functions invoked by mouse, button or keyboard input are dependent upon the position of the cursor. While MIT developed the system with a significant number of post processing techniques, the user was not always aware of the capability unless he referred to the documentation.

A key feature in this program was to give the user the ability to modify pV3's user interface using advanced programming calls from pV3. To demonstrate this new capability, a Motif GUI was developed by Pratt & Whitney and integrated with the pV3 visualization server (see Fig. 3). The Motif GUI is an event-driven interface. An event is defined as an occurrence caused by user input through either mouse, keyboard or input devices attached to serial ports. When an event is detected by the user interface, the system will updates information in the data structures and calls the appropriate procedure in pV3. The selected visualization procedure will then read the necessary information from the data structures and construct the visual objects that are displayed in the 1D, 2D or 3D graphics windows. Documentation on the Motif GUI was provided as Web pages that could be launched though the Motif GUI using either Netscape or Explorer Web browsers.

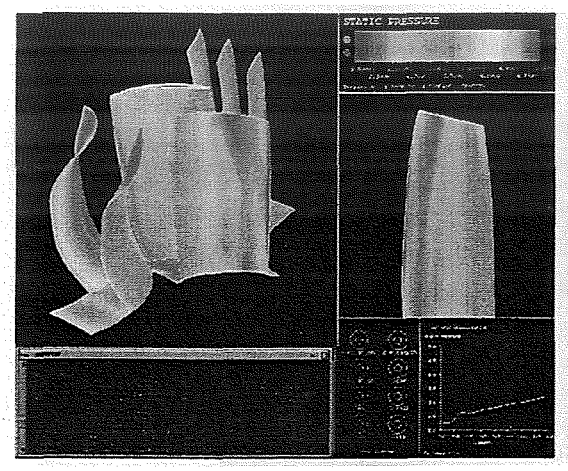


Figure 2 - Original pv3 GUI

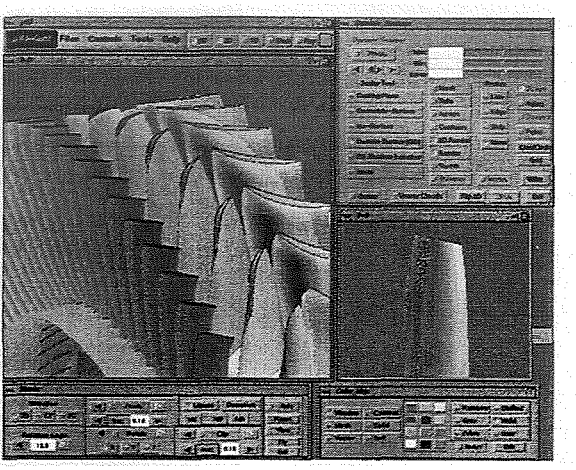


Figure 3 – pV3 with Motif GUI

Batch collector/playback viewer

Another weakness in the original pV3 system is that the user may not be around to start the interactive pV3 server and view the results. Even if someone is looking at the data, when the image on the screen is updated, the old information is not saved for later retrieval. The problem with saving away the entire volume of data for post-processing is that the time-step selected for the visualization is based on the available disk space and not the physical time scales of the integration. The pV3 system does not exhibit this problem because co-processing is fully supported. To resolve these issues another new module has been added to the system (Fig. 1), the batch server. In this case, the pV3 client side remains totally unchanged.

When a batch job starts, the batch pV3 server is also started. Data is read on where and what tools and probes are to be active and their locations. The results are collected and written to disk for later playback using a MIT Generic Extraction Data Structure (GEDS) file format. This is different from the normal post-processing in that the entire volume of data is not written to disk every iteration. The purpose of the batch server is to write the visualization data to disk

The end result is something that is not interactive in the placement of tools, but can be thought of as analogous to a wind-tunnel experiment. Some knowledge of the flow must be used in the placement of probes to extract data of interest. If important information is missed the `tunnel' will have to re-run. A simple pV3 viewer was developed to playback the results. The Motif interface was integrated with this viewer.

Figure 4 shows a snap shot of a hot streak migration in the UTRC Large Scale Rotating Rig (LSRR). The results were generated with the pV3 system in a batch mode to collect the results a hot streak migration simulation. The size of GEDS file generated from this simulation was significantly smaller than saving the entire volume (5% of the original file). An animation of these results was generated.

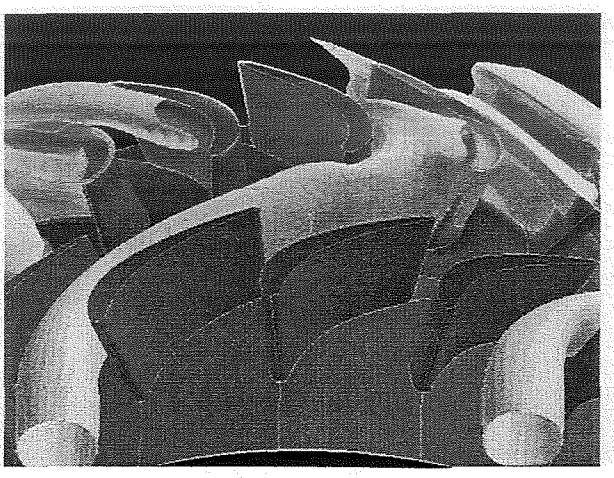


Figure 4 – MIT GEDS Extraction File Applied to Collect Transient Information Hot Streak In Large Scale Rotating Rig Simulation

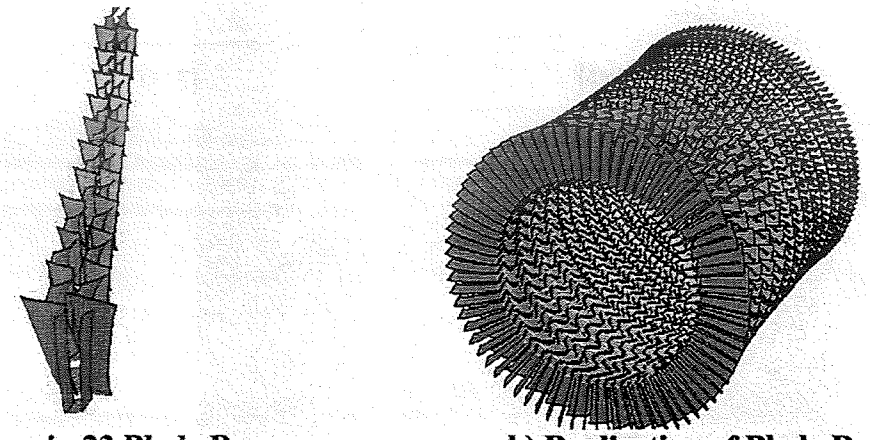
Multidisciplinary viewing of multiple scalar and vector fields

The client-side of pV3 has been altered to extract information simultaneously from a set of different disciplines. A discipline is one or more similar clients. All clients that are members of the discipline must have the same set of field variables. The classic example of multidiscipline visualization is a structures/fluids coupled simulation. In this case the volume for the fluid and the volume for the structure would abut. The nodes that make up the fluid volume usually contain variables such as density, energy and momentum. The nodes of the solid may contain variables such as deflection and stress. Other quantities are constructed from these state vectors and displayed during the visualization session. A new capability allowing the simultaneous visualization of multi-disciplines has been developed by MIT and integrated into pV3's visualization servers and clients. A discipline is one or more similar clients. The classic example of MDV is a structures/fluids coupled simulation. All pV3 clients belonging to the same discipline are assumed to have the same field variables (i.e. scalar, vector, and etc.).

The design of the server (and post-processing viewer) for multi-disciplinary cases assumes there is a current active discipline. All user interaction effects that discipline.

Sub-Sectioning/Replication of the solution

pV3 was modified to allow the visualization server to select the portion of the simulation for viewing. Either the entire simulation can be viewed or specific pV3 clients can be selected. Addition tranformations can be made to each client to allow replication (or duplication) of client surfaces which was consider an important requirement for turbomachinery applications. An example of this can be seen in Fig. 5. This figure shows the pressure on the blades of the multistage compressor. The simulation is a three-dimensional steady flow Navier-Stokes simulation of a complete high-pressure compressor with 23 airfoil rows. Approximately 2000 blades are in Fig. 5b.



a) Pressure in 23 Blade Rows **b) Replication of Blade Rows**

Figure 5 –Multistage High Pressure Compressor CFD Steady State Simulation

Annotation

Another limitation of pV3 was its lack of providing the user the ability to write text in the graphics windows. The capability to position user specified text and arbitrary drawing objects into pV3's 3D and 2D windows was added to the pV3 system.

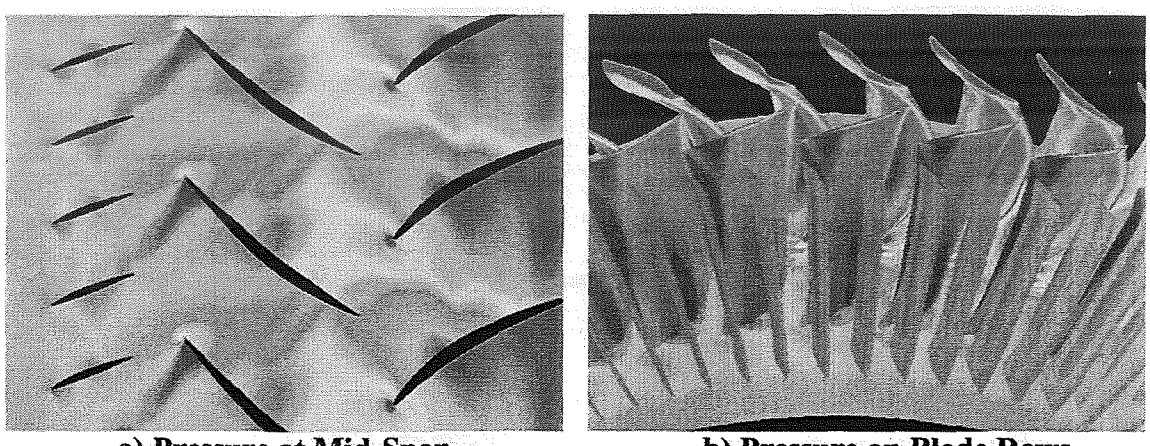
MPEG Animation

A new capability was developed to allow key frame viewing positions to be saved with animation of the viewing position accomplished by cycling through the key frames. This allowed for recording/collection of frames from either the 2D or 3D graphics window during an interactive session that can be combined into a Moving Picture Experts Group (MPEG) file using a MPEG encoder. This technique was developed to allow the engineer to create quick animation that is stored in an MPEG -1 digital format for archiving or distribution through either the WWW or electronic e-mail. An example of this can be seen in Fig. 6. This figure shows a snapshot of the pressure on the blades and midspan of a compressor. The simulation is a three-dimensional unsteady Navier-Stokes simulation of resonant stress attributable to rotor/stator interaction. The animation was created by collecting information using the pV3 batch processor coupled with the CFD solver to create a pV3 GEDS extract file. The pV3 viewer was then used to generate the frames for the MPEG file.

Collaboration techniques

The pV3 visualization system was redesigned to add the ability for collaborative visualization where multiple visualization viewers can be used in a single session or a single user to examine a simulation with multiple viewers (either for examining the results from different view points or for examining different variables simultaneously). Each viewer has complete independence -- no viewer to viewer communication.

A new feature tested allows multiple visualization servers to be coupled to the simulation as it is running thus allowing both the co-processing visualization server and the batch collector to run simultaneously. This feature would also allow remote users to link into the simulation.



a) Pressure at Mid-Span **b) Pressure on Blade Rows**

Figure 6 –Compressor Resonant Stress CFD Unsteady Simulation

Conclusion

An interactive visualization system pV3 is being developed for the investigation of advanced computational methodologies employing visualization and parallel processing for the extraction of information contained in large-scale transient engineering simulations. Visual techniques for extracting information from the data in terms of cutting planes, iso-surfaces, particle tracing and vector fields are included in this system. This paper discussed improvements to the pV3 system developed under NASA's Affordable High Performance Computing project.

Acknowledgments

This work was sponsored by the Affordable High Performance Computing project at NASA Lewis Research Center with Theresa Babrauckas as the Technical Monitor.

References

1. Robert Haimes, pV3: A Distributed System for Large-Scale Unsteady Visualization, AIAA Paper 94-0321, 1994.
2. Robert Haimes and David Edwards, Visualization in a Parallel Processing Environment AIAA Paper 97-0348, 1997.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1999	Conference Publication

4. TITLE AND SUBTITLE
HPCCP/CAS Workshop Proceedings 98

5. FUNDING NUMBERS
509-10-61

6. AUTHOR(S)
Edited by Catherine Schulbach

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER
A-990762

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/TM–1999-208757

11. SUPPLEMENTARY NOTES
Point of Contact: Catherine Schulbach, Ames Research Center, MS T27A-2, Moffett Field, CA 94035-1000
(650) 604-3180

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified — Unlimited
Subject Category 01
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

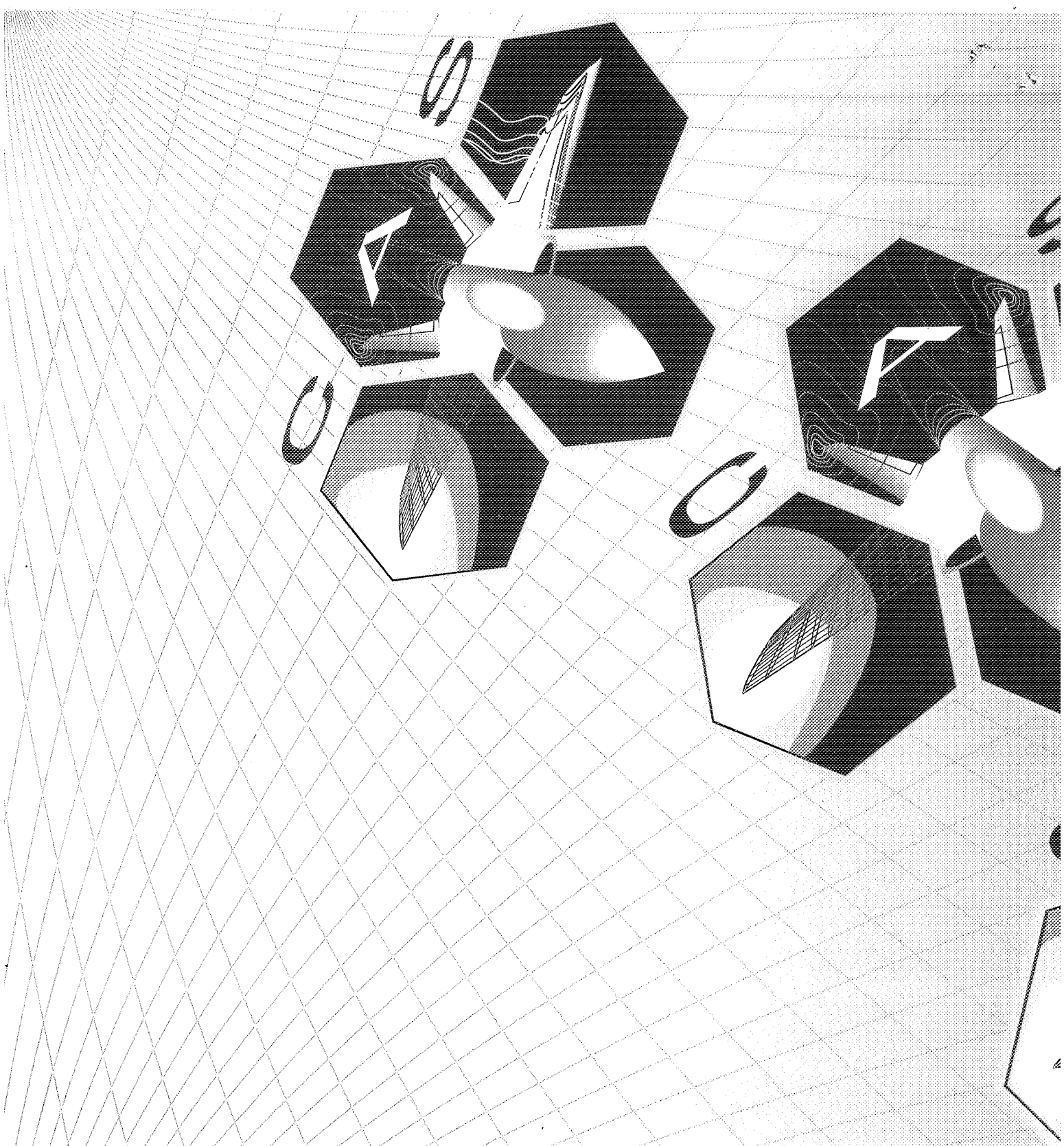
13. ABSTRACT ***(Maximum 200 words)***

This publication is a collection of extended abstracts of presentations given at the HPCCP/CAS Workshop held on August 24-26, 1998, at NASA Ames Research Center, Moffett Field, California. The objective of the Workshop was to bring together the aerospace high performance computing community, consisting of airframe and propulsion companies, independent software vendors, university researchers, and government scientists and engineers. The Workshop was sponsored by the High Performance Computing and Communications Program Office at NASA Ames Research Center.

The Workshop consisted of over 40 presentations, including an overview of NASA's High Performance Computing and Communications Program and the Computational Aerosciences Project; ten sessions of papers representative of the high performance computing research conducted within the Program by the aerospace industry, academia, NASA, and other government laboratories; two panel sessions; and a special presentation by Mr. James Bailey.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Computational aerosciences, High performance computing, Grand challenges	275
	16. PRICE CODE: A14

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		



NASA Ames Research Center