JOURNAL OF AIRCRAFT
Vol. 33, No. 3, May–June 1996

Parallel Computational Fluid Dynamics Design on Network-Based Computer

Samson Cheung*
NASA Ames Research Center, Moffett Field, California 84035

Combining multiple engineering workstations into a network-based heterogeneous parallel computer allows the application of aerodynamic optimization with advanced computational fluid dynamics codes, which can be computationally expensive on mainframe supercomputers. This article introduces a nonlinear quasi-Newton optimizer designed for this network-based heterogeneous parallel computing environment utilizing a software called Parallel Virtual Machine. This article will introduce the methodology behind coupling a parabolized Navier–Stokes flow solver to the nonlinear optimizer. This parallel optimization package is applied to reduce the wave drag of a body of revolution and a wing/body configuration with results of 5–6% drag reduction.

Introduction

As the microelectronic industry improves the performance of personal computers and workstations, the idea of using these low-cost computers to perform numerical optimization naturally presents itself. Unlike large mainframe computers, such as the Cray C90, which are typically saturated with jobs waiting to be executed, most personal engineering workstations remain idle during off hours. A workstation is normally used for only 40 h a week, thus leaving 128 h of free time. This idle computer represents a significant computational resource equating about 3 CPU hours of Cray C90 time per week. Furthermore, the annual maintenance fee for a typical supercomputer is about $1.2M (hardware alone, not including the cooling system), which is approximately equivalent to the cost of purchasing 30 high-performance workstations. Recently, computational fluid dynamics (CFD) flow solvers have been implemented on clusters of workstations as a network-based parallel computer.[1,2] The objective of this article is to explore the possibility of using a numerical optimizer on a network-based parallel computer system to perform aerodynamic design with advanced CFD flow solvers.

A common direct optimization technique is called the brute force method.[3,4] In this method, an analysis code is coupled with a numerical optimizer to find a shape that optimizes the objective function. This method may be computationally expensive because of the large number of gradient evaluations, each requiring at least one CFD calculation. Recently, intelligent methods, such as the one-shot method[5,6] and the control theory based method,[7] have yielded significant improvements in computational efficiency. Since these two methods make use of the governing equations to evaluate sensitivity derivatives, they require considerable programming effort for each new objective function, boundary condition, and flow solver. Because of this complexity in implementation, they take a long time to mature. Therefore, these methods cannot take advantage of existing validated CFD flow solvers without a serious programming effort.

A shape perturbation brute force method is chosen for this study. A parabolized Navier–Stokes CFD flow solver coupled with a numerical optimizer designed for parallel computation serves as a new aerodynamic design tool. A similar technique has been successfully applied to minimize the drag of a theoretical minimum-drag body, and to study the computational grid effect on aerodynamic optimization on a single Cray Y-MP processor, by Cheung.[8] The present numerical optimization code, which is based on the quasi-Newton method, is suitable for aerodynamic optimization on parallel computers. This optimization package is applied to minimize the wave drag of two simple geometries that will be described in later sections.

Received Feb. 11, 1995; presented as Paper 95-0984 at the AIAA Computing in Aerospace 10, San Antonio, TX, March 28–30, 1995; revision received Oct. 30, 1995; accepted for publication Oct. 30, 1995.

*Research Scientist; currently at McDonnell Douglas Aerospace, Long Beach, CA 90807-4418. Member AIAA.

Parallel Virtual Machine

The network communication software package used in this study, Parallel Virtual Machine (PVM), was developed jointly at Emory University and the Oak Ridge National Laboratory.[9] This software package allows a heterogeneous network of parallel and series computers to appear as a single concurrent computational resource. In other words, PVM links up the computational systems preferred by users to work as a single distributed-memory (parallel) machine. This machine is called a virtual machine. The PVM software is composed of two parts. The first part is the daemon (pvmd). This is the control center of the virtual machine. It is responsible for starting processes, establishing links between processes, passing messages, and many other activities in PVM. The second part of the system is a library of PVM interface routines. This library contains user callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

Optimization Algorithms

The merit of the method of steepest descent is its guarantee of convergence. However, it takes a large number of small steps before it converges to the optimum. Unlike the method of steepest descent, Newton's method has fast convergence; but the evaluation of the Hessian matrix is impractical or costly. Moreover, the initial guess has to be relatively close to the optimum for the method to converge. The presented optimizer is an unconstrained numerical optimization code based on the quasi-Newton method. The idea of the quasi-Newton method is to start as a steepest descent, and it ends as Newton's method. In the following sections these three traditional approaches will be briefly reviewed.

Steepest Descent

Let $f(x)$ be the function of several variables being optimized and whose first partial derivatives are continuous. The descent

direction at point $x = a$ is a column vector, $\nabla f(a)^T$. For convenience, denote $\nabla f(a)^T$ by g. The method of steepest descent is defined by the iterative algorithm

$$a_{n+1} = a_n - \alpha_n g_n \tag{1}$$

where αx is a nonnegative scalar minimizing $F(\alpha) \equiv f(a_n - \alpha g_n)$. This method is guaranteed to converge, but with many very small steps.

Newton's Method

Newton's method extends the concept of steepest descent such that the function f, being minimized, is approximated locally by a quadratic function. Thus, near a_n, f can be approximated by the truncated Taylor series

$$f(a) \approx f(a_n) + \nabla f(a_n)(a - a_n) + \tfrac{1}{2}(a - a_n)^T F(a_n)(a - a_n)$$

where ∇f and F are the gradient and the Hessian matrix of f, respectively. The right-hand side is minimized at

$$a_{n+1} = a_n - [F(a_n)]^{-1} g_n \tag{2}$$

and this equation is the pure form of Newton's method. Once a_{n+1} is calculated, the gradient and the Hessian are updated at the new point, $n + 1$, and the process is repeated.

Quasi-Newton Method

Equations (1) and (2) are the algorithms of steepest descent and Newton's method, respectively. A generalization of these two equations is

$$a_{n+1} = a_n - \alpha_n H_n g_n \tag{3}$$

Note that Eq. (3) is the steepest descent if H_n is the identity matrix; and it is Newton's method if H_n is the inverse of the Hessian and α_n is unity. It is clear that the matrix H_n approximates the inverse of the Hessian. It is possible to construct a method that does not entail the direct evaluation of F; methods of this type are referred to as quasi-Newton methods and have the following properties:

1) As n tends to infinity, H_n tends to the inverse of Hessian.

2) H_n is symmetric and positive definite because Hessian of f is.

The earliest, and certainly one of the most clever schemes for constructing the H matrix, was originally proposed by Davidon,[10] and later developed by Fletcher and Powell.[11] This method is called the Davidon–Fletcher–Powell (DFP) method. The algorithm of the quasi-Newton method using the DFP approach is given as follows.

For $n = 0$, starting with any symmetric positive definite matrix H_0 (normally is the identity matrix) and any initial guess a_0:

Step 1: Compute the gradient g_n of f at a_n.

Step 2: Minimize $F(\alpha) = f(a_n - \alpha H_n g_n)$ with respect to $0 \le \alpha \le 1$ to obtain α_{n+1}.

Step 3: Compute the gradient g_{n+1} and $p_n = -\alpha_n H_n g_n$.

Step 4: Set $q_n = g_{n+1} - g_n$ and update H_n by

$$H_{n+1}^{\text{DFP}} = H_n + \frac{p_n p_n^T}{p_n^T q_n} - \frac{H_n q_n q_n^T H_n}{q_n^T H_n q_n} \tag{4}$$

Return to step 2 and repeat until convergence is achieved.

The DFP updating formula [Eq. (4)] is for approximation of the inverse Hessian of f. It is also possible to update approximations of the Hessian itself. An updating formula for the Hessian, called BFGS, which has the same form as Eq. (4) with p and q interchanged, has been introduced by Broyden et al.[12] By taking the inverse of this BFGS formula, a corresponding update for H is produced:

$$H_{n+1}^{\text{BFGS}} = H_{n+1} + \left(\frac{1 + q_n^T H_n q_n}{q_n^T p_n}\right)\frac{p_n p_n^T}{p_n^T q_n} - \frac{p_n q_n^T H_n + H_n q_n p_n^T}{q_n^T p_n} \tag{5}$$

Weighted combinations of Eqs. (4) and (5) give a collection of updates known as the Broyden family:

$$H^{\phi} = (1 - \phi)H^{\text{DFP}} + \phi H^{\text{BFGS}} \tag{6}$$

After algebraic manipulation, Eq. (6) can be written as

$$H_{n+1}^{\phi} = H_n + \frac{p_n p_n^T}{p_n^T q_n} - \frac{H_n q_n q_n^T H_n}{q_n^T H_n q_n} + \phi_n (v_n v_n^T) \tag{7}$$

where

$$v_n = (q_n^T H_n q_n)^{1/2}\left(\frac{p_n}{p_n^T q_n} - \frac{H_n q_n}{q_n^T H_n q_n}\right)$$

Originally there was considerable effort devoted to searching for the best choice of ϕ in a Broyden method, but Dixon[13] showed that all members are identical in the case of an exact line search. For an excellent review of the subject, see Dennis and More.[14]

Scaling

In step 2 of the algorithm, a line search for $f(a_n - \alpha H_n g_n)$ is required. Theoretically, any textbook line search technique will make the algorithm complete. Nevertheless, an exact line search is impractical because of the expensive CFD computations, round-off errors, and nonquadratic terms in the objective function f. Therefore, it is desirable to know if the updating methods described previously are sensitive to the accuracy of the line search.

Example

The following example shows that the BFGS method is indeed sensitive to the accuracy of the line search algorithm. Consider the minimization problem of this analytic quadratic function $f(x) = 0.5x^T Q x$, where

$$Q = \begin{bmatrix} 40 & 0 & 0 & 0 & 0 & 0 \\ 0 & 38 & 0 & 0 & 0 & 0 \\ 0 & 0 & 36 & 0 & 0 & 0 \\ 0 & 0 & 0 & 34 & 0 & 0 \\ 0 & 0 & 0 & 0 & 32 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

The solution is obviously $x = (0, 0, 0, 0, 0, 0)^T$. Let the initial guess be $a_0 = (10, 10, 10, 10, 10, 10)^T$. For this simple quadratic problem, the exact value of α is known by the formula of conjugate gradient method[15]:

$$\alpha_n = \frac{g_n^T H_n g_n}{(H_n g_n)^T Q (H_n g_n)}$$

Table 1 shows the convergence history of the BFGS method with different errors made in α on the line search.

Table 1 Values of f with different error in α

Iterations	No error	1% error	10% error
1	96.30	97.34	200.33
2	6.90E-01	1.68	99.71
3	3.99E-03	9.37E-01	83.39
4	1.68E-05	7.340E-01	5.11
5	3.88E-08	1.85E-02	9.01E-01
6	7.10E-19	1.26E-04	8.01E-01
7	2.07E-39	7.81E-05	5.42E-02
8	——	7.83E-09	7.85E-03
9	——	7.83E-13	7.85E-05
10	——	7.83E-17	7.85E-07

Table 2 Values of f with and without scaling

Iterations	No error in α	With scaling 10% error in α	No scaling 10% error in α
1	96.30	200.33	200.33
2	6.90E-01	2.77	99.71
3	3.99E-03	3.49E-02	83.39
4	1.68E-05	4.12E-04	5.11
5	3.88E-08	4.64E-06	9.01E-01
6	7.10E-19	5.05E-08	8.01E-01
7	2.07E-39	5.38E-10	5.42E-02
8	——	5.62E-12	7.85E-03
9	——	5.77E-14	7.85E-05
10	——	——	7.85E-07

This table shows that BFGS method works properly only if the error in the line search procedure is small. Since an exact line search is impractical in the CFD optimization process, an error in α because of approximation and round-off errors would be too big to slow down the convergence of the optimization process.

Scaling Factor

The error in the line search affects the accuracy of matrix $\boldsymbol{H}_n$, and the rate of convergence of the method suffers. To have an efficient quasi-Newton algorithm, the formula in Eqs. (4), (5), or (7) needs to be modified. Since $\boldsymbol{H}_n$ is the approximation of the inverse of the Hessian matrix $\boldsymbol{F}$; the eigenvalues of $\boldsymbol{H}_n\boldsymbol{F}$ must approach unity as the method converges. It is logical to think that multiplying a scaling factor σ_n to $\boldsymbol{H}_n$ such that the eigenvalues of $\boldsymbol{H}_n\boldsymbol{F}$ are close to unity would do some good in the process of convergence. Multiplying σ_n to the $\boldsymbol{H}_n$ in Eq. (7) gives

$$\boldsymbol{H}_{n+1}^{\phi} = \frac{\boldsymbol{p}_n\boldsymbol{p}_n^T}{\boldsymbol{p}_n^T\boldsymbol{q}_n} + \left[\boldsymbol{H}_n - \frac{\boldsymbol{H}_n\boldsymbol{q}_n\boldsymbol{q}_n^T\boldsymbol{H}_n}{\boldsymbol{q}_n^T\boldsymbol{H}_n\boldsymbol{q}_n} + \phi_n(v_n v_n^T)\right]\sigma_n \quad (8)$$

Luenberger[15] suggested that

$$\sigma_n = \boldsymbol{p}_n^T\boldsymbol{q}_n/\boldsymbol{q}_n^T\boldsymbol{H}_n\boldsymbol{q}_n \quad (9)$$

Table 2 shows the convergence of the previous example with and without the scaling factor on $\boldsymbol{H}_n$. Error in α for the last two columns in Table 2 is 10%. The entries at the first and the last columns of this table are the same as Table 1. The second column shows the convergence of the scaled method [Eqs. (8) and (9)]. It is clearly shown that the scaling factor helps tremendously on the convergence vs the nonscaled one.

Parallel Optimizer

One primary benefit of parallelism in optimization is to reduce turnaround time. In aerodynamic optimization, one may include the following parallelism in an optimization algorithm:

1) Parallelize the evaluation of the derivatives of the objective function and the line search procedure.

2) Parallelize the linear algebra involved in each iteration (i.e., quasi-Newton method in this article).

3) Parallelize each evaluation of the objective function (i.e., the CFD flow solver is parallelized).

4) Parallelize the optimization procedure to perform multidisciplinary design optimization (MDO) concurrently.

The present optimization code on the PVM is called *I*IOWA (parallel optimization with aerodynamics). At the present stage, the code is written only with 1 and 2 taken into account.

Master and Slave

To parallelize the evaluation of the derivatives of the objective function, the present optimization algorithm uses the master/slave concept in parallel programming. Figure 1 shows graphically how the concept works. Steps 1 and 3 of the quasi-Newton algorithm require the evaluation of the gradient $\boldsymbol{g}$, which is typically obtained by a central- or forward-differencing method. Let $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_M)$ be the difference intervals in the differencing scheme. The gradient is given by

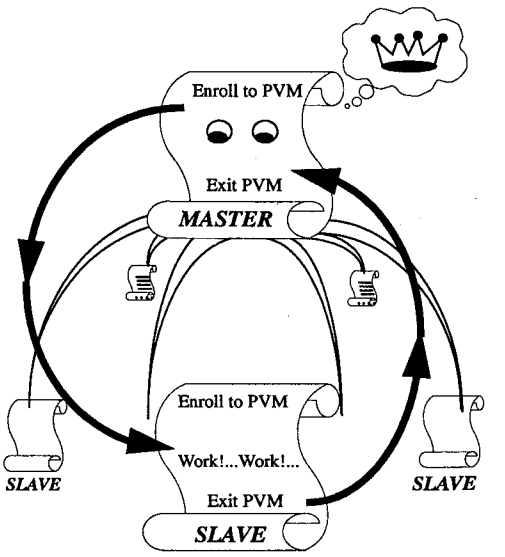

Fig. 1 Master/slave concept in parallel communication.

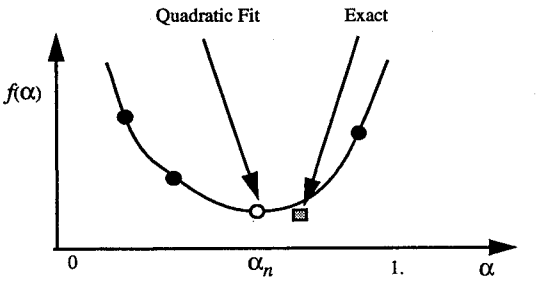

Fig. 2 Quadratic curve fitting to approximate minimum at line search routine.

$$\boldsymbol{g}^T \equiv \nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_M}\right)\bigg|_{x=a} \quad (10)$$

where the ith component is approximated by (forward differencing)

$$\frac{\partial f}{\partial x_i}\bigg|_{x=a} \approx [f(a_1, \ldots, a_i + \varepsilon_i, \ldots, a_M) - f(a_1, \ldots, a_i, \ldots, a_M)]/\varepsilon_i \quad (11)$$

In CFD aerodynamic design, usually the most time-consuming procedure is the CFD calculations. With M processors of the same speed computing simultaneously, it should speed up the design process by a factor of M.

Line Search

In the line search procedure for $F(\alpha) = f(\boldsymbol{a}_n - \alpha\boldsymbol{H}_n\boldsymbol{g}_n)$, only an approximation of α is obtained. Three or more processors are called to obtain the values of F over a range of α, simultaneously. A quadratic curve fitting is used to predict the minimum of the line search (see Fig. 2). The range of α is set to be between 0 and 1, because as the method converges to Newton's method, α approaches unity for the solution to the quadratic problem.

The author's experience shows that α should be chosen as 1, 1/4, $1/4^2$, $1/4^3$, ..., 0, initially. If the minimum is at $\alpha = 0$, a search between (0, 1/4) is conducted until the quadratic curve

fitting can be applied or α is small enough to conclude that the local minimum is found.

Parallelizing Linear Algebra

To parallelize the linear algebraic calculations of the quasi-Newton method, there are two common methods to perform the BFGS update in step 4 of quasi-Newton algorithm:

1) Updating a Cholesky factorization of the Hessian matrix $\boldsymbol{F}_n$ to a Cholesky factorization of $\boldsymbol{F}_{n+1}$, and then reducing this updated matrix back to lower triangular form by a sequence of operations; each calculation of the gradient $\boldsymbol{g}_n$ requires two triangular solves.

2) Updating the inverse of the Hessian $\boldsymbol{H}_n$ to $\boldsymbol{H}_{n+1}$ by adding a rank-two matrix to $\boldsymbol{H}_n$; each calculation of the gradient $\boldsymbol{g}_n$ requires a matrix–vector multiplication.

Details described for these algebraic methods can be found in Ref. 16. Both methods require $\mathcal{O}(n^2)$ operations. Method 2 is cheaper, but method 1 has been used in most production codes (such as NPSOL, DOT, and MATLAB),[17] because it implicitly retains positive definiteness of the Hessian approximation guaranteeing the search directions are descent directions. Nonetheless, method 2 requires only matrix–vector multiplication, which can be parallelized very nicely, whereas the reducing to triangular form and triangular solves in method 1 require sequences of vector–vector operations on vectors ranging from length 2 to n, which parallelize very poorly.

Note that a recent study found negligible differences in the numerics over broad sets of problems.[18] Therefore, the present optimizer is written with method 2. This is desirable, especially if the number of design variables is reasonably large.

Flow Solver

The implemented CFD flow solver is a three-dimensional parabolized Navier–Stokes code, UPS3D (Ref. 19). This is a space-marching code that can calculate steady-state viscous or inviscid solutions under supersonic flow conditions. A conic approximation is made for the initial marching plane. This code is further supported by a hyperbolic grid generation scheme[20] that is sufficiently fast and robust to operate within an automated optimization environment. In this study, only inviscid supersonic calculations are employed.

In this article, the UPS3D code uses a step size of 0.06 to 0.1% of the body length L on the grids of two geometries: 1) a body of revolution and 2) a wing/body configuration. The grid points are clustered near the body surface to obtain accurate pressure drag. Figure 3 illustrates a typical grid on a body of revolution used by UPS3D.

This CFD flow solver is highly vectorized and it is very efficient on a mainframe supercomputer like the Cray C90. However, vectorized codes may not have optimal performance on personal workstations because of the hierarchical memory in workstations. For a typical inviscid computation on a grid of 21 × 41 (step size 0.006), UPS3D code performs at 4.2 Mflops (million floating point operations per second) rate on the SGI Indigo-2 workstation.

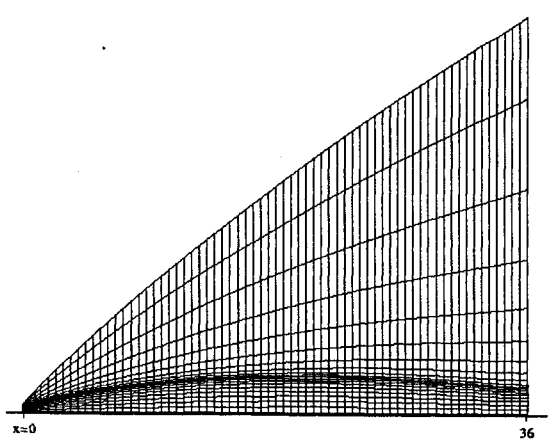

Fig. 3 Schematic marching grid for UPS3D code.

Test Cases

Two simple geometries are considered in this article. The first geometry is a body of revolution called the Haack–Adams (H–A) body,[8,21,22] and the second geometry is the Boeing arrow wing/body configuration.[23] Both geometries have wind-tunnel data for code validation. In this study, the wave-drag coefficient C_{Dw} is the objective function being minimized. The C_{Dw} is calculated by integrating pressure over the surface of the geometry. The CFD calculations and optimization process are performed on four SGI Indigo-2 workstations.

H–A Body

The H–A body is a classic aerodynamic shape of minimum drag derived from supersonic slender body theory. This body is chosen as an optimization test case because of the large database of experimental data[24] that can be used to verify the CFD code. The finite area at the base A_{base} of the H–A body facilitates correlation with experimental models, which have an attached sting and simplifies modeling with space-marching codes. Let x be the streamwise direction with θ defined such that $x = L(1 + \cos\theta)/2$. Also, let $r(x)$ and $A(x)$ be the radius and area distributions of the body along x. Define $x_{\max}$ to be the location at which the body has maximum radius $r_{\max}$. The radius distribution of the H–A body is given by

$$r^2/r_{\max}^2 = (A_{\text{base}}/\pi A_{\max})(\pi - \theta + \tfrac{1}{2}\sin 2\theta + \gamma_1 \tfrac{4}{3}\sin^3\theta) \quad (12)$$

where $\gamma_1 = 1/[2\cos(\theta_{\max})]$. As a validation test case, the UPS3D code was run in inviscid mode over a range of supersonic Mach numbers. A conic approximation at 1% body length downstream from the nose was made for the initial marching plane of the code. With this initial plane choice an acceptable level of error in wave drag was demonstrated.[8] Furthermore, this downstream distance is far enough to allow the use of a relatively large marching step size (0.1% of body length). Figure 4 shows that the CFD results compare well to those from theory of characteristics and experimental data. Note the variation of wave drag with Mach number is predicted by both the theory of characteristics and inviscid CFD solutions. However, slender body theory predicts no variation of wave drag with Mach number.

Previous explorations[8] of wave drag minimization have been performed on a Cray Y-MP with the serial numerical optimizer NPSOL.[25] The optimized body shape has the form:

$$\frac{r^2}{r_{\max}^2} = \frac{A_{\text{base}}}{\pi A_{\max}}\left\{\pi - \theta + \frac{1}{2}\sin 2\theta + \gamma_1 \frac{4}{3}\sin^3\theta + \sum_{m=2}^{5}\gamma_m\left[\frac{\sin m\theta}{m} - \frac{\sin(m+2)\theta}{m+2}\right]\right\} \quad (13)$$

According to linear theory, the γ_m, $m \leq 2$, are set to zero [compare Eqs. (12) and (13)]. However, since nonlinear effects are included in the CFD analysis, four of these coefficients ($m = 2, 3, \ldots, 5$) are considered to be design variables. In this study, the parallel optimization package on the workstations took 5 h and 43 min wall-clock time to obtain a new configuration whose drag coefficient is 0.075096. Compared with the original drag coefficient of the H–A body of 0.079598, the drag on the new body has been reduced by 5.65%. Figure 5 shows the radius distribution of the original H–A body and the optimized bodies from *I*IOWA (workstations) and NPSOL (Cray Y-MP). The tiny discrepancy between the two optimized results could be from the difference in machine precisions.

Arrow Wing/Body

The second geometry chosen for this study is an arrow wing/body configuration tested by Boeing in 1976 (Ref. 23). The

fuselage of the original configuration is a body of revolution with constant radius as shown in Fig. 6. A typical chordwise pressure coefficient comparison at 20% of semispan is shown in Fig. 7. The two results are in good agreement.

This exercise is to reduce the drag on the aircraft by changing the fuselage cross-sectional area in the region where the wing intersects. The fuselage radius is perturbed by a Fourier sine series. The design variables are the Fourier coefficients. The idea of picking Fourier coefficients as design variables is the same as the previous test case. The only constraint to this optimization problem is to make sure the radius of the fuselage is positive.

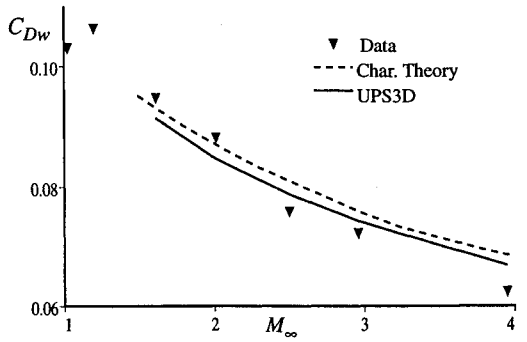

Fig. 4 Wave drag comparison over a range of Mach numbers for H–A body.

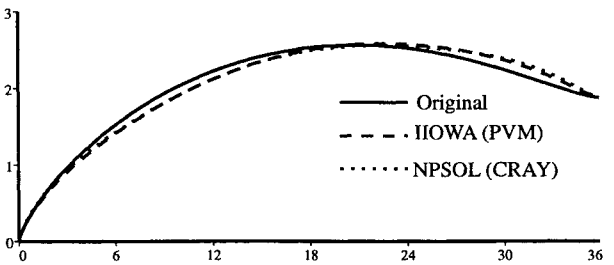

Fig. 5 Radius distribution of the optimized H–A body on NPSOL and IIOWA.

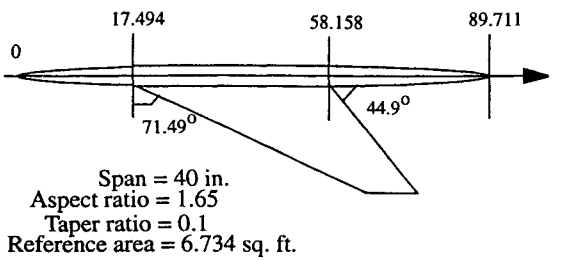

Fig. 6 Boeing arrow wing/body geometry.

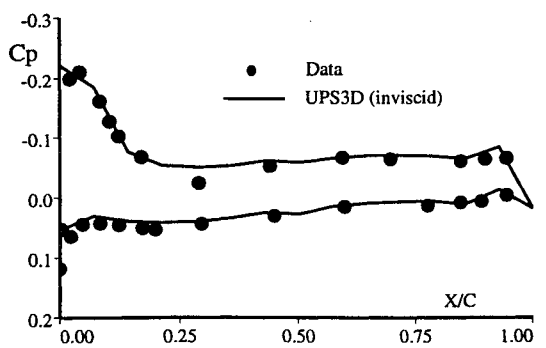

Fig. 7 Pressure coefficient at 20% semispan. Freestream Mach number 2.1, angle of attack 4 deg.

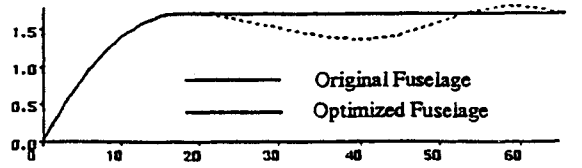

Fig. 8 Optimized fuselage has a coke-bottle shape as expected in supersonic linear theory.

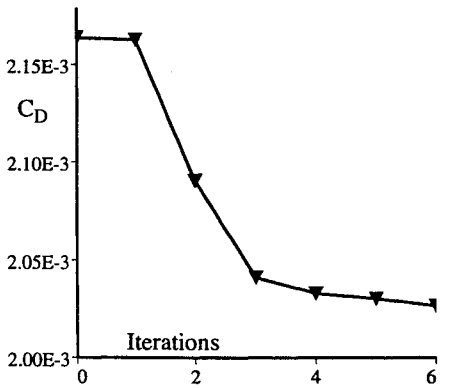

Fig. 9 Convergence history of the optimization process in the case of the arrow wing/body configuration.

Figure 8 summarizes the result of the optimization after 24 wall-clock hours on the four SGI workstations. The optimized fuselage has a coke-bottle shape, which is expected from supersonic linear theory. The drag reduction is about 6.5%. One thing that can be learned from the U-velocity (the streamwise velocity component) contours is that changing the fuselage area can disturb the flow at the trailing edge where the nacelles are usually mounted. Therefore, in the design of propulsion integration, wing/body interference can be taken into account. Figure 9 shows the convergence history of this optimization process. The first step was steepest descent and the speed of convergence was low; but as iterations increased, the convergence was sped up quadratically as promised in the quasi-Newton method.

Conclusions

Combining multiple workstations into a network-based heterogeneous parallel computer opens the door of using these low-cost, high-performance workstations to perform computationally intensive applications, which could only be achieved on supercomputers less than a decade ago. This article introduces a quasi-Newton optimization method (*II*OWA) designed for the parallel computing environment by utilizing the PVM software. The error in the line search procedure of the quasi-Newton method retards the convergence rate and accuracy suffers. This error is difficult to avoid, especially for CFD-based optimizations in which grid density, complex geometry, and turbulence model contribute to the complexity of the problems.

However, a simple scaling factor to the Hessian approximation matrix, which can stabilize the optimization process, is implemented in the optimizer. An example of *II*OWA's application is given in which the wave drag of a body of revolution and a wing/body configuration are reduced by 5–6%.

References

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

[2]Smith, M. H., and Van der Wijngaart, B. F., "Granularity and the Parallel Efficiency of Flow Solution on Distributed Computer Systems," AIAA Paper 94-2260, 1994.

[3]Cosentino, G. B., and Holst, T. L., "Numerical Optimization Design of Advanced Transonic Wing Configurations," NASA TM

85950, May 1984.

[4]Vanderplaats, G. N., and Hicks, R. M., "Numerical Airfoil Optimization Using a Reduced Number of Design Coordinates," NASA TM X-73,151, July 1976.

[5]Ta'asan, S., "One Shot Methods for Optimal Control of Distributed Parameters System I: Finite Dimensional Control," Inst. for Computer Applications in Science and Engineering Rept. 91-2, Jan. 1991.

[6]Ta'asan, S., Kuruvila, G., and Salas, M., "Aerodynamic Design and Optimization in One Shot," AIAA Paper 92-0025, Jan. 1992.

[7]Reuther, J., and Jameson, A., "Control Theory Based Airfoil Design for Potential Flow and a Finite Volume Discretization," AIAA Paper 94-0499, Jan. 1994.

[8]Cheung, S., Aaronson, P., and Edwards, T., "CFD Optimization of a Theoretical Minimum-Drag Body," AIAA Paper 93-3421, Aug. 1993.

[9]*PVM Manual*, Oak Ridge National Lab., Oak Ridge, TN, May 1994.

[10]Davidon, W., "Variable Metric Method for Minimization," U.S. Atomic Energy Commission, Argonne National Labs., R&D Rept. ANL-5990 (Ref.), 1959.

[11]Fletcher, R., and Powell, M. J. D., "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, Vol. 6, 1963, pp. 163–168.

[12]Broyden, C., "The Convergence of a Class of Double Rank Minimization Algorithms: Part I and II," *Journal of the Institute of Mathematics and It's Applications*, Vol. 6, 1970, pp. 76–90, 222–231.

[13]Dixon, L., "Quasi-Newton Algorithms Generate Identical Points," *Mathematical Programming*, Vol. 2, 1972, pp. 383–387.

[14]Dennis, J., Jr., and More, J., "Quasi-Newton Methods, Motivation and Theory," *SIAM Review*, Vol. 19, 1977, pp. 46–89.

[15]Luenberger, D. G., *Linear and Nonlinear Programming*, Addison–Wesley, Reading, MA, 1984.

[16]Dennis, J. E., and Schnabel, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[17]More, J. J., and Wright, S. J., *Optimization Software Guide*, Society for Industrial and Applied Mathematics, 1993.

[18]Byrd, R. H., Schnabel, R. B., and Shultz, G. A., "Parallel Quasi-Newton Methods for Unconstrained Optimization," *Mathematical Programming*, Vol. 42, 1988, pp. 273–306.

[19]Lawrence, S., Chaussee, D., and Tannehill, J., "Application of an Upwind Algorithm to the 3-D Parabolized Navier-Stokes Equations," AIAA Paper, 87-1112, June 1987.

[20]Chan, W., and Steger, J., "A Generalized Scheme for Three-Dimensional Hyperbolic Grid Generation," AIAA Paper 91-1588, June 1991.

[21]Adams, A. C., "Determination of Shapes of Boattail Bodies of Revolution for Minimum Wave Drag," NACA TN-2550, 1951.

[22]Haack, W., "Projectile Shapes for Smallest Wave Drag," U.S. Air Force Material Command, Brown Univ., Translation A9-T-3, Contract W33-038-ac-15004 (16351), ATI 27736, Providence, RI, 1948; also Sears, W. R., "On Projectiles with Minimum Wave Drag," *Quarterly Applied Mathematics*, Vol. 4, No. 4, 1974, pp. 361–366.

[23]Manro, M. E., NASA CR-145046, 1976.

[24]Harris, R., and Landrum, E., "Drag Characteristic of a Series of Low-Drag Bodies of Revolution at Mach Numbers from 0.6 to 4.0," NASA TN D-3163, Dec. 1965.

[25]Gill, P., Murray, W., Saunders, M., and Wright, M., "User's Guide for NPSOL: A Fortran Package for Nonlinear Programming," Stanford Univ., Dept. of Operations Research, TR SOL 86-2, Stanford, CA, 1986.