AIAA 96–0409

Automatic Aerodynamic Optimization on Distributed Memory Architectures

Antony Jameson and Juan J. Alonso
Princeton University, Princeton, NJ 08544

34th AIAA Aerospace Sciences Meeting and Exhibit
January 8–12, 1996/Reno, NV

AIAA 96–0409

Automatic Aerodynamic Optimization on Distributed Memory Architectures

Antony Jameson* and Juan J. Alonso†
Princeton University, Princeton, NJ 08544

This paper presents a parallel implementation of an automatic Euler design method based on the control theory of systems governed by partial differential equations. The Euler equations and the resulting adjoint equations necessary to calculate the Frechet derivatives for the gradient of the cost function are solved using a domain decomposition approach with communication handled by the MPI (Message Passing Interface) Standard. Parallel performance is evaluated on a distributed memory parallel computer and sample calculations are presented. A complete optimization procedure on a $192 \times 32 \times 48$ mesh can be completed in 7 minutes using 16 processors of an IBM SP2 system. This clearly shows that parallel processing is a key enabling technology for CFD to become an efficient tool in a realistic design environment. The parallel implementation of a multiblock version of the program which allows for a higher degree of geometric complexity in the design has recently been completed. Parallel performance trends of the multiblock code are consistent with the ones observed in the single block implementation.

Introduction

UP to now, several factors have limited the impact of Computational Fluid Dynamics (CFD) in the industrial design environment. Among these factors, two seem especially limiting. Firstly, traditional CFD methods have mostly focused on the analysis of existing configurations with the purpose of identifying possible problems and shortcomings of the existing design: little or no guidance is offered to the designer on how to improve the existing design or how to circumvent the configuration shortcomings. Secondly, the complexity of the required calculations has lengthened the turn-around time to a point where CFD cannot play an effective role in the continuously shrinking design cycles. For numerical techniques to be employed successfully in a design environment, these two key issues need to be dealt with effectively.

In the last four years, considerable efforts have been directed towards the solution of these two problems using two complementary approaches. First, the field of parallel computing has reached a level of maturity in both hardware configurations, and message passing standards (PVM, MPI, Linda, Express, etc.[6,14]). Distributed memory computing platforms are starting to be considered more than just a research tool, and computer programs for numerical simulations can now be both efficient and portable. Second, the relevance of tackling the design problem has gained the recognition of the aerospace community, and several efforts are under way in order to decrease the computational and algorithmic costs associated with complex geometry flow sensitivity calculations.

For the purpose of optimization, it is typical to describe the shape to be optimized with a series of base functions multiplied by a given design variable. Once a suitable cost function to be minimized has been defined (drag coefficient, lift/drag ratio, difference from a specified pressure distribution, etc.), it is customary to take small steps in each and every one of the design variables independently, in order to find the sensitivity of the cost function with respect to that design variable. Each of these steps requires a complete flow solution, and one can clearly see how this approach can become unsuitable when large numbers of design variables are introduced.

In particular, Jameson has advocated the use of control theory for systems governed by partial differential equations in the design optimization problem. With this approach, the gradient of the cost function can be inexpensively found for an arbitrary number of design variables. The computational cost is essentially that of solving the adjoint equation, whose computational complexity is slightly lower than that of the flow solution. This approach has been proven to yield optimized solutions of wing and wing-body configurations at a very reasonable cost.[11,16]

In spite of the large improvements in algorithmic efficiency, methods based on control theory still require the repeated calculation of flow solutions, which results in demanding computational requirements. Therefore the reduction of computational cost is crucial to the further development of effective design methods. Recently, a viable option has emerged

*James S. McDonnell Distinguished University Professor of Aerospace Engineering, AIAA Fellow

†Graduate Student, AIAA Member

that is proving to be quite attractive: parallel systems with a moderate number (on the order of 10 to 100) of relatively high performance RISC based CPUs. These systems usually have distributed memory, and the different processors that make up the machine are connected through some sort of high bandwidth, low latency network. This new class of machines made up of off-the-shelf processors provides a level of cost/performance that very well suits the optimization problem. There also exists the possibility of using existing high performance workstations connected with a high bandwidth, low latency network such as HIPPI or FDDI.

The parallelization of CFD problems is usually tackled using domain decomposition methods whereby all the processors in a calculation perform the same operations on a different subset of the global data. Since these subsets of data physically reside in different processors, the need for an explicit communication of the data arises. Several message passing "pseudo-standards" have appeared in the last few years. Unfortunately, these have not been unanimously recognized by the user community, and impose a considerable risk for applications that are ported to these interfaces, since future versions may not be compatible with previous ones, and the communication software might not even be available at a later time.

More recently, a new standard has emerged out of the cooperation of several members of industry, government laboratories, and academe. The MPI (Message Passing Interface) Standard is currently supported by the leading computer manufacturers (IBM, Silicon Graphics, Cray, Intel, Convex, etc.) and it has been gaining acceptance among the research and engineering communities. Extensive experience[1,4] has proven that porting of existing codes can be achieved in a relatively straightforward fashion, and parallel efficiencies of 70% and above can be obtained for engineering size problems using up to 32 processors. Portable implementations of the MPI Standard have been written[13] and allow a single code to run on both multiprocessor machines and distributed networks of workstations without additional programming work.

This paper presents the application of parallel computing to the aerodynamic design optimization problem using control theory. In the following sections, the design method is outlined, the parallelization strategy is discussed, and some calculations, as well as performance results are presented. A complete design procedure for an isolated wing can be completed in about 7 minutes on 16 processors of an IBM SP2 computer. For a complete configuration, the wing of a business jet in the presence of other aerodynamic components (fuselage, nacelles, vertical and horizontal tails) can be designed in 170 minutes using 16 processors and a multiblock implementation of our design method; in the near future, this rapid turn-around will clearly allow automatic design calculations to become routine for the aircraft designer.

The Design Problem as a Control Problem

Aerodynamic design has traditionally been carried out on a cut and try basis, with the aerodynamic expertise of the designer guiding the selection of each shape modification. Although considerable gains in aerodynamic performance have been achieved by this approach, continued improvement will most probably be much more difficult to attain. The subtlety and complexity of fluid flow is such that it is unlikely that repeated trials in an interactive analysis and design procedure can lead to a truly optimum design. Automatic design techniques are therefore needed in order to fully realize the potential improvements in aerodynamic efficiency.

The simplest approach to optimization is to define the geometry through a set of design parameters, which may, for example, be the weights α_i applied to a set of shape functions $b_i(x)$ so that the shape is represented as

$$f(x) = \sum \alpha_i b_i(x).$$

Then, a cost function I is selected which might, for example, be the drag coefficient or the lift to drag ratio, and I is regarded as a function of the parameters α_i. The sensitivities $\frac{\partial I}{\partial \alpha_i}$ may now be estimated by making a small variation $\delta\alpha_i$ in each design parameter in turn and recalculating the flow to obtain the change in I. Then

$$\frac{\partial I}{\partial \alpha_i} \approx \frac{I(\alpha_i + \delta\alpha_i) - I(\alpha_i)}{\delta\alpha_i}.$$

The gradient vector $\frac{\partial I}{\partial \alpha}$ may now be used to determine a direction of improvement. The simplest procedure is to make a step in the negative gradient direction by setting

$$\alpha^{n+1} = \alpha^n - \lambda \frac{\partial I}{\partial \alpha},$$

so that to first order

$$I + \delta I = I + \frac{\partial I^T}{\partial \alpha}\delta\alpha = I - \lambda \frac{\partial I^T}{\partial \alpha}\frac{\partial I}{\partial \alpha}.$$

For some time Jameson has advocated the advantages of formulating both the inverse problem and more general aerodynamic problems within the framework of the mathematical theory for the control of systems governed by partial differential equations.[12] A wing, for example, is a device to produce lift by controlling the flow, and its design can be regarded as a problem in the optimal control of the flow equations by variation of the shape of the boundary. If the boundary shape is regarded as arbitrary within some requirements of smoothness, then the full generality of shapes cannot be defined with a finite number of parameters, and one must use the concept of the Frechet derivative of the

cost with respect to a function. Clearly, such a derivative cannot be determined directly by finite differences of the design parameters because there are now an infinite number of these. Using techniques of control theory, however, the gradient can be determined indirectly by solving an adjoint equation which has coefficients defined by the solution of the flow equations. The cost of solving the adjoint equation is comparable to that of solving the flow equations. Thus the gradient can be determined with roughly the computational cost of two flow solutions, independently of the number of design variables, which may be infinite if the boundary is regarded as a free surface.

For flow about an airfoil or wing, the aerodynamic properties which define the cost function are functions of the flow-field variables, w, and the physical location of the boundary, which may be represented by the function $\mathcal{F}$, say. Then

$$I = I\left(w, \mathcal{F}\right),$$

and a change in $\mathcal{F}$ results in a change

$$\delta I = \frac{\partial I^T}{\partial w}\delta w + \frac{\partial I^T}{\partial \mathcal{F}}\delta \mathcal{F}, \tag{1}$$

in the cost function. Using control theory, the governing equations of the flowfield are introduced as a constraint in such a way that the final expression for the gradient does not require reevaluation of the flowfield. In order to achieve this δw must be eliminated from (1). Suppose that the governing equation R which expresses the dependence of w and $\mathcal{F}$ within the flowfield domain D can be written as

$$R\left(w, \mathcal{F}\right) = 0. \tag{2}$$

Then δw is determined from the equation

$$\delta R = \left[\frac{\partial R}{\partial w}\right]\delta w + \left[\frac{\partial R}{\partial \mathcal{F}}\right]\delta \mathcal{F} = 0. \tag{3}$$

Next, introducing a Lagrange Multiplier ψ, we have

$$\begin{aligned}\delta I &= \frac{\partial I^T}{\partial w}\delta w + \frac{\partial I^T}{\partial \mathcal{F}}\delta \mathcal{F} - \psi^T\left(\left[\frac{\partial R}{\partial w}\right]\delta w + \left[\frac{\partial R}{\partial \mathcal{F}}\right]\delta \mathcal{F}\right)\\ &= \left\{\frac{\partial I^T}{\partial w} - \psi^T\left[\frac{\partial R}{\partial w}\right]\right\}\delta w + \left\{\frac{\partial I^T}{\partial \mathcal{F}} - \psi^T\left[\frac{\partial R}{\partial \mathcal{F}}\right]\right\}\delta \mathcal{F}.\end{aligned}$$

Choosing ψ to satisfy the adjoint equation

$$\left[\frac{\partial R}{\partial w}\right]^T \psi = \frac{\partial I}{\partial w} \tag{4}$$

the first term is eliminated, and we find that

$$\delta I = \mathcal{G}\delta \mathcal{F}, \tag{5}$$

where

$$\mathcal{G} = \frac{\partial I^T}{\partial \mathcal{F}} - \psi^T\left[\frac{\partial R}{\partial \mathcal{F}}\right].$$

The advantage is that (5) is independent of δw, with the result that the gradient of I with respect to an arbitrary number of design variables can be determined without the need for additional flow-field evaluations. In the case that (2) is a partial differential equation, the adjoint equation (4) is also a partial differential equation and appropriate boundary conditions must be determined.

After making a step in the negative gradient direction, the gradient can be recalculated and the process repeated to follow a path of steepest descent until a minimum is reached. In order to avoid violating constraints, such as a minimum acceptable wing thickness, the gradient may be projected into the allowable subspace within which the constraints are satisfied. In this way one can devise procedures which must necessarily converge at least to a local minimum, and which can be accelerated by the use of more sophisticated descent methods such as conjugate gradient or quasi-Newton algorithms. There is the possibility of more than one local minimum, but in any case the method will lead to an improvement over the original design. Furthermore, unlike the traditional inverse algorithms, any measure of performance can be used as the cost function.

This approach to optimal aerodynamics design has proved effective in a variety of applications.[9, 10, 16] The adjoint equations have also been used by Ta'asam, Kuruvila and Salas,[18] who have implemented a one shot approach in which the constraint represented by the flow equations is only required to be satisfied by the final converged solution, and computational cost is also reduced by applying multigrid techniques to the geometry modifications as well as the solution of the flow and adjoint equations. Pironneau has studied the use of control theory for optimal shape design of systems governed by elliptic equations,[15] and more recently the Navier-Stokes equations, and also wave reflection problems. Adjoint methods have also been used by Baysal and Eleshaky.[3]

Design using the Euler Equations

The application of control theory to aerodynamic design problems is illustrated in this section for the case of three-dimensional wing design using the compressible Euler equations as the mathematical model. It proves convenient to denote the Cartesian coordinates and velocity components by x_1, x_2, x_3 and u_1, u_2, u_3, and to use the convention that summation over $i = 1$ to 3 is implied by a repeated index i. Then, the three-dimensional Euler equations may be written as

$$\frac{\partial w}{\partial t} + \frac{\partial f_i}{\partial x_i} = 0 \quad \text{in } D, \tag{6}$$

where

$$w = \begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho E \end{Bmatrix}, \quad f_i = \begin{Bmatrix} \rho u_i \\ \rho u_i u_1 + p\delta_{i1} \\ \rho u_i u_2 + p\delta_{i2} \\ \rho u_i u_3 + p\delta_{i3} \\ \rho u_i H \end{Bmatrix} \tag{7}$$

and δ_{ij} is the Kronecker delta function. Also,

$$p = (\gamma - 1)\,\rho \left\{ E - \frac{1}{2}\left(u_i^2\right) \right\}, \tag{8}$$

and

$$\rho H = \rho E + p \tag{9}$$

where γ is the ratio of the specific heats.

Consider a transformation to coordinates ξ_1, ξ_2, ξ_3 where

$$K_{ij} = \left[\frac{\partial x_i}{\partial \xi_j} \right], \quad J = \det(K), \quad K_{ij}^{-1} = \left[\frac{\partial \xi_i}{\partial x_j} \right],$$

and

$$Q = JK^{-1}.$$

The elements of Q are the coefficients of K, and in a finite volume discretization they are just the face areas of the computational cells projected in the x_1, x_2, and x_3 directions. Also introduce scaled contravariant velocity components as

$$U_i = Q_{ij} u_j.$$

The Euler equations can now be written as

$$\frac{\partial W}{\partial t} + \frac{\partial F_i}{\partial \xi_i} = 0 \quad \text{in } D, \tag{10}$$

where

$$W = Jw,$$

and

$$F_i = Q_{ij} f_j = \begin{bmatrix} \rho U_i \\ \rho U_i u_1 + Q_{i1} p \\ \rho U_i u_2 + Q_{i2} p \\ \rho U_i u_3 + Q_{i3} p \\ \rho U_i H \end{bmatrix}.$$

Assume now that the new computational coordinate system conforms to the wing in such a way that the wing surface B_W is represented by $\xi_2 = 0$. Then the flow is determined as the steady state solution of equation (10) subject to the flow tangency condition

$$U_2 = 0 \quad \text{on } B_W. \tag{11}$$

At the far field boundary B_F, conditions are specified for incoming waves, as in the two-dimensional case, while outgoing waves are determined by the solution.

The weak form of the Euler equations for steady flow can be written as

$$\int_{\mathcal{D}} \frac{\partial \phi^T}{\partial \xi_i} F_i d\mathcal{D} = \int_{\mathcal{B}} n_i \phi^T F_i d\mathcal{B}, \tag{12}$$

where the test vector ϕ is an arbitrary differentiable function and n_i is the outward normal at the boundary. If a differentiable solution w is obtained to this equation, it can be integrated by parts to give

$$\int_{\mathcal{D}} \phi^T \frac{\partial F_i}{\partial \xi_i} d\mathcal{D} = 0$$

and since this is true for any ϕ, the differential form can be recovered. If the solution is discontinuous (12) may be integrated by parts separately on either side of the discontinuity to recover the shock jump conditions.

Suppose now that it is desired to control the surface pressure by varying the wing shape. For this purpose, it is convenient to retain a fixed computational domain and then, variations in the shape result in corresponding variations in the mapping derivatives defined by K. Introduce the cost function

$$I = \frac{1}{2} \iint_{B_W} (p - p_d)^2 \, d\xi_1 d\xi_3,$$

where p_d is the desired pressure. The design problem is now treated as a control problem where the control function is the wing shape, which is to be chosen to minimize I subject to the constraints defined by the flow equations (10–). A variation in the shape will cause a variation δp in the pressure and consequently a variation in the cost function

$$\delta I = \iint_{B_W} (p - p_d)\, \delta p \;\; d\xi_1 d\xi_3. \tag{13}$$

Since p depends on w through the equation of state (8–9), the variation δp can be determined from the variation δw. Define the Jacobian matrices

$$A_i = \frac{\partial f_i}{\partial w}, \quad C_i = Q_{ij} A_j. \tag{14}$$

The weak form of the equation for δw in the steady state becomes

$$\int_{\mathcal{D}} \frac{\partial \phi^T}{\partial \xi_i} \delta F_i d\mathcal{D} = \int_{\mathcal{B}} (n_i \phi^T \delta F_i) d\mathcal{B},$$

where

$$\delta F_i = C_i \delta w + \delta Q_{ij} f_j,$$

which should hold for any differential test function ϕ. This equation may be added to the variation in the cost function, which may now be written as

$$\begin{aligned} \delta I = & \iint_{B_W} (p - p_d)\, \delta p \;\; d\xi_1 d\xi_3 \\ & - \int_{\mathcal{D}} \left(\frac{\partial \psi^T}{\partial \xi_i} \delta F_i \right) d\mathcal{D} \\ & + \int_{\mathcal{B}} \left(n_i \psi^T \delta F_i \right) d\mathcal{B}. \end{aligned} \tag{15}$$

On the wing surface B_W, $n_1 = n_3 = 0$. Thus, it follows from equation (11) that

$$\delta F_2 = \begin{bmatrix} 0 \\ Q_{21}\delta p \\ Q_{22}\delta p \\ Q_{23}\delta p \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \delta Q_{21} p \\ \delta Q_{22} p \\ \delta Q_{23} p \\ 0 \end{bmatrix}. \qquad (16)$$

Since the weak equation for δw should hold for an arbitrary choice of the test vector ϕ, we are free to choose ϕ to simplify the resulting expressions. Therefore we set $\phi = \psi$, where the costate vector ψ is the solution of the adjoint equation

$$\frac{\partial \psi}{\partial t} - C_i^T \frac{\partial \psi}{\partial \xi_i} = 0 \quad \text{in } D. \qquad (17)$$

At the outer boundary incoming characteristics for ψ correspond to outgoing characteristics for δw. Consequently one can choose boundary conditions for ψ such that

$$n_i \psi^T C_i \delta w = 0.$$

Then, if the coordinate transformation is such that δQ is negligible in the far field, the only remaining boundary term is

$$-\int\int_{B_W} \psi^T \delta F_2 \; d\xi_1 d\xi_3.$$

Thus, by letting ψ satisfy the boundary condition,

$$Q_{21}\psi_2 + Q_{22}\psi_3 + Q_{23}\psi_4 = (p - p_d) \quad \text{on } B_W, \qquad (18)$$

we find finally that

$$\delta I = -\int_{\mathcal{D}} \frac{\partial \psi^T}{\partial \xi_i} \delta Q_{ij} f_j d\mathcal{D} - \int\int_{B_W} (\delta Q_{21}\psi_2 + \delta Q_{22}\psi_3 + Q_{23}\psi_4)\, p \, d\xi_1 d\xi_3. \qquad (19)$$

A convenient way to treat a wing is to introduce sheared parabolic coordinates through a conformal mapping transformation. The mapping formulas can be introduced into equation (19) and a final form for the variation in the cost function can be obtained, which solely depends on the solution of the adjoint problem, ψ, and the location of the surface of the wing, but not on the flow variables. Thus, the gradient can be inexpensively calculated. The reader is referred to[11] for more details on the final form of these formulas.

Parallelization Strategy

As one can see in the previous sections, the parallelization of the design method entails three separate parts: the solution of the flow equations, the solution of the adjoint equations, and the calculation of the gradient integral formulas.

The similarities between the flow and adjoint equations allow them to be solved using exactly the same efficient numerical techniques, with the exception of the boundary conditions and the equation coefficients: the same parallelization techniques used for the flow equations apply to the solution of the adjoint equations. Therefore, all details of the parallel implementation corresponding to these first two parts of the program will be explained with reference only to the flow equations.

Both the flow and adjoint solutions are obtained using a finite volume discretization of the governing equations with the flow and costate variables stored at the cell centers. Details of the spatial discretization, the time-stepping formulas, and the convergence acceleration techniques (multigrid, implicit residual averaging, enthalpy damping, and local time-stepping) used in this work can be found in.[8]

The parallelization strategy has so far been developed and extensively tested with a single block, wing-body Euler solver named FLO87. The automatic design version of this code is called SYN87. This program allows the automatic design of isolated wings with at most the presence of a fuselage in the flow. Its parallel implementation will be called SYN87P. Due to topological constraints, more complicated geometries cannot be treated with a single block structured mesh. For that purpose, a multiblock version of FLO87 called FLO87-MB has been used for the design method, and it allows, for example, the aerodynamic design of a wing, with additional aerodynamic components (pylons, nacelles, winglets, etc.) present. The parallel implementation of the multiblock design method will be referred to as SYN87P-MB.

SYN87P. Single Block Parallel Implementation

The three-dimensional meshes for wing (or wing-body) calculations are generated using the same conformal mapping transformation to sheared parabolic coordinates that is used for the calculation of the gradient formulas.

SYN87P is parallelized using a domain decomposition model, a SPMD (Single Program Multiple Data) strategy, and the MPI (Message Passing Interface) Library for message passing. The choice of message passing library was determined by the requirement that the resulting code be portable to different parallel computing platforms as well as to homogeneous and heterogeneous networks of workstations.

Flows were computed on a C-H mesh of size $n_i \times n_j \times n_k = 192 \times 32 \times 48$, totalling 294912 cells in the complete mesh. This domain was decomposed into subdomains containing $\frac{n_i}{N_{p_i}} \times \frac{n_j}{N_{p_j}} \times \frac{n_k}{N_{p_k}}$ points, where N_{p_i}, N_{p_j}, and N_{p_k} are the number of subdomains in the i, j, and k coordinate directions, for a

total of $N_{p_i} \times N_{p_j} \times N_{p_k}$ processors in the complete calculation. The number of subdomains in each coordinate direction is an input to the program and can be changed depending on the number of available processors. However, there is the requirement that the number of processors requested be the product of three integers ($N_{p_i} \times N_{p_j} \times N_{p_k}$). It must also be mentioned that the number of subdomains in each coordinate direction limits the maximum number of multigrid levels that can be used in a calculation in the same manner as with the serial implementation of the program.

Communication between subdomains is performed through halo cells surrounding each subdomain boundary. Since both the convective and the dissipative fluxes are calculated at the cell faces (boundaries of the control volumes), all six neighboring cells are necessary, thus requiring the existence of a single level halo for each processor in the parallel calculation. The dissipative fluxes are composed of a blend of first and third order differences corresponding to terms that mimic second and fourth derivatives of the flow quantities. For the third order differences at the boundary faces of each cell in the domain, the presence of the twelve neighboring cells (two adjacent to each face) is required. For each cell within a processor, Figure 1 shows which neighboring cells are required for the calculation of convective and dissipative fluxes. For each processor, some of these cells will lie directly next to an interprocessor boundary, in which case, the values of the flow variables residing in a different processor will be necessary to calculate the convective and dissipative fluxes. In the finest mesh of the multigrid

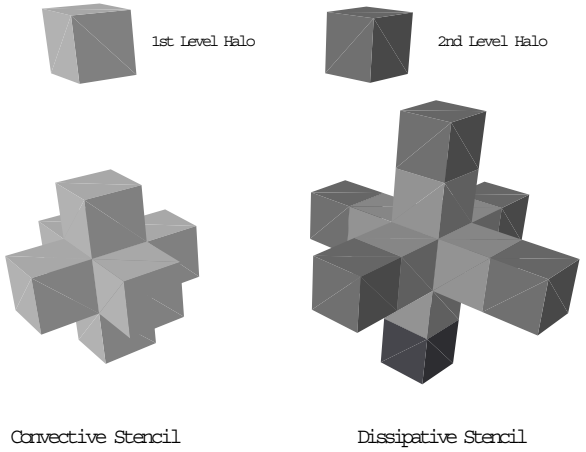


Fig. 1 Convective and Dissipative Discretization Stencils.

sequence, a two-level halo was sufficient to calculate the convective and dissipative fluxes for all cells contained in each processor. In the coarser levels of the multigrid sequence, a single level halo suffices since a simplified model of the artificial dissipation terms is used.

The actual communication routines used are all of the asynchronous (or non-blocking) type. In the current implementation of the program, each processor must send and receive messages to and from at most 6 neighboring processors (left and right neighbors in each of the three coordinate directions). The communication is scheduled such that at every instant in time pairs of processors are sending/receiving to/from one another in order to minimize contention in the communication schedule.

For a given number of subdomains in a calculation, there are several ways to partition the complete mesh according to the scheme explained above. Depending on the choice of partitions, the bounding surface area of each subdomain will vary, reaching a minimum when the sizes in each of the three coordinate directions are equal. Currently the partition of the global mesh is an input to the code, but work is in progress to determine, in a pre-processing step, the optimal block distribution in order to minimize the communication requirements.

SYN87P-MB. Multiblock Parallel Implementation

The essential algorithm (convective and dissipative flux calculation, multigrid, implicit residual averaging, etc.) is exactly the same as the one applied in the single block case. The only difference resides in the fact that an additional outer loop over all the blocks in the domain is added. The parallelization strategy, however, is quite different. Similarly to SYN87P, SYN87P-MB is parallelized using a domain decomposition model, a SPMD strategy, and the MPI Library for message passing. The partitioning of the mesh could be performed in the same fashion as in SYN87P for each of the blocks in the mesh. Since the sizes of the blocks can be quite small sometimes, further partitioning would severely limit the number of multigrid levels that can be used in the flow and adjoint solutions. For this reason, it was decided to allocate complete blocks to each processor.

The underlying assumption is the fact that there always will be more blocks than processors available. If this is the case, every processor in the domain would be responsible for the computations inside one or more blocks. In the case in which there are more processors than blocks available, the blocks can be adequately partitioned during a pre-processing step in order to at least have as many blocks as processors. This approach has the advantage that the number of multigrid levels that can be used in the parallel implementation of the code is always the same as in the serial version. Moreover, the number of processors in the calculation can now be any integer number, since no restrictions are imposed by the partitioning in all coordinate directions used by the single block program.

Processor Number	Percentage of Load
1	12.50000
2	12.50000
3	12.98701
4	12.01298
5	12.98701
6	12.01298
7	12.50000
8	12.50000

Table 1 Calculated Load Distribution on an 8 Processor Calculation

The only drawback of this approach is the loss of the exact load balancing that one had in SYN87P. All blocks in the calculation can have different sizes, and consequently, it is very likely that different processors will be assigned a different total number of cells in the calculation. This, in turn, will imply that some of the processors will be waiting until the processor with the largest number of cells has completed its work and parallel performance will suffer. The approach that we have followed to solve the load balancing problem is to assign to each processor, in a pre-processing step, a certain number of blocks such that the total number of cells is as close as possible to the exact share for perfect load balancing. For example, the mesh for a wing/body/nacelle calculation for the small business jet in the results section was made up of 72 structured blocks of different sizes. When 8 processors are used, the load balancing obtained can be seen in Table 1 to be quite close to exact. One must note that load balancing based on the total number of cells in each processor is only an approximation to the optimal solution of the problem. Other variables such as the number of blocks, the size of each block, and the size of the buffers to be communicated play an important role in proper load balancing, and are the subject of current study.

Now, within each processor, there will be several blocks that need to communicate with their neighboring blocks. The data for these neighboring blocks can reside in a different processor, and therefore, communication is necessary. In order to minimize communication cost, it was decided to pack all data that needed to be communicated from one processor to another in one single message, regardless of the number of blocks that resided in each of the processors. Within each processor, the data for the flow variables, adjoint variables, and grid locations is stored on a large one-dimensional array. In order to accomplish this type of communication, during the pre-processing step, each processor compiles a pointer list with all the entries in these large arrays that need to be sent to all other processors. Similarly, another pointer list for the locations of the data to be received is also setup. At the time of communication exchanges each processor communicates all the information for the blocks that it contains to those processors that need to receive it. Once again, the communication is implemented using the asynchronous (non-blocking) send and receive MPI constructs in order to be able to do some useful work while the information is being transferred.

Parallel Efficiency

For problems with a low task granularity (ratio of the number of bytes received by a processor to the number of floating point operations it performs), large parallel efficiencies can be obtained. Unfortunately, convergence acceleration techniques developed in the 1980s base their success on global communication in the computational domain. Thus, current multigrid and implicit residual smoothing techniques are bound to hinder parallel performance in traditional mesh sizes. For larger meshes used in viscous turbulent flow calculations, the granularity becomes low enough, and the parallel performance is quite high.

Several techniques can be applied to reduce the communication cost of the multigrid technique. Among these, one can consider completely eliminating communication at the coarser levels of the multigrid cycle (thus allowing each processor to operate independently with the multigrid forcing terms at interprocessor boundaries derived from the flow variables in the finest mesh). Alternatively, one can also avoid sending messages at the end of every stage in the Runge-Kutta time stepping. Past experience has shown[1] that these savings in communication cost are usually offset by a degradation in the convergence rate of the overall algorithm. Therefore, it was decided to allow message passing any time the flow variables were altered.

A common approach to improving the performance of parallel programs is to hide the communication (startup and completion) cost by carrying out a certain amount of useful computation while the communication is being completed. Opportunities for this type of *latency hiding* can be easily found in CFD programs based on explicit algorithms like ours. These algorithms lend themselves quite naturally to parallelism due mostly to the locality of their computational stencil. A way to implement *latency hiding* for these explicit algorithms is described below. Typically, the update of all cells in a given block (or partition of a block) is done in a single subroutine for all the cells in the block. After the cell values are updated, messages are sent in order to set the proper values in the halos of all the blocks in the calculation. However, only the new values of the *outer shell* of a processor are needed in order to start sending the information to the other processors. The update process can then be split into three parts:

1. Compute the fluxes and update the values for the layer of cells lying adjacent to the faces of each block.

2. Send the necessary messages to the neighboring processors/blocks using asynchronous calls.

3. Proceed to complete the update of the cells interior to each block.

4. Receive the messages that were sent in step 2 of this procedure.

If the amount of work needed for the update of the interior cells is comparable to the time necessary to complete the sending of the messages, the communication cost will be totally masked. In any case, since some work needs to be done to update the inner cells, at least part of the communication cost will be masked.

It is interesting to note that for a case using 16 processors, typical messages in the coarser levels of multigrid range from $1,728$ to 108 bytes in length. For an IBM SP2 system, the measured bandwidth and latency are 35 Mbytes/sec and 43 μsec respectively. Clearly, at the lower levels of multigrid, communication cost is latency dominated, and this fact gets accentuated when using a W-cycle instead of a V-cycle, since the coarser levels are traversed more often. Thus, the optimization of multigrid calculations on medium sized meshes becomes problematic.

Future trends in multiprocessor computing hardware will tend to increase the available bandwidth to a processor faster than the latency is expected to decrease.[5] Therefore, to take full advantage of the multigrid technique in a parallel environment, the avoidance of message passing at the coarsest levels of the sequence is of extreme importance. Tests have been conducted in which message passing is completely turned off at the coarser levels of the multigrid cycle. The result is a slight increase of performance (4-7%) with no degradation in the convergence ratio, as long as the levels in which the communication is turned off are far from the mesh in which we are performing the calculation: this has been found by experience to be at least two levels.

Implicit residual averaging is a technique that couples the residuals on the global mesh along computational directions. It can clearly be seen that a technique such as this will require the values of residuals which in the parallel context reside in different processors. Previous work[1] showed that for two-dimensional viscous flows, performing implicit residual averaging separately within each subdomain lead to a decrease in convergence rate of the overall algorithm. Testing on three-dimensional inviscid flows, however, showed instead that this technique can be applied with no noticeable loss in convergence rate. This has also proven to be true for viscous three-dimensional flows.[2] For this reason, implicit residual averaging is applied separately within each block (or partition of a block) in both SYN87P and SYN87P-MB.

The final formulas for the gradient of the cost function include a number of volume and surface integrals (see[11]) that can also be calculated in parallel. At the present time, only the solution of the flow and adjoint equations is performed concurrently. Since the solution of these two equations consumes a very significant amount of the total computational time (over 95 % for 8 processors) the parallel calculation of the gradient formulas was given second priority. For large number of processors (32 and above), the computational time involved in the calculation of the gradient becomes significant, and the parallelization of the gradient formulas will become necessary in order to preserve the overall high parallel performance of the complete method. These volume integrals are currently performed on a loop through all the cells in the global domain. The nature of the integrations is such that, from a domain decomposition point of view, the problem is nearly embarrassingly parallel, since the partial values of the integrals in each domain can be calculated without the need for any communication at all. Once the individual processors have completed their calculation, a single global reduce operation call can combine the results into the total gradients, which can now be used to compute the shape change at each design iteration.

Results

In this section we present results for the single block parallel implementation of the automatic design method using control theory, and preliminary results for the multiblock version.

SYN87P Results and Performance

SYN87P has been tested in conjunction with the serial version of the code until the same optimized results were produced by both computer programs. We will first present a sample calculation which used 16 processors of an IBM SP2 system. In this case, the wing planform was held fixed while the locations of the points on the surface of the wing were free to move with a minimum thickness constraint imposed on the possible variations of the geometry, in order to prevent the optimization method from driving the geometry towards a zero lift flat plate. The initial wing has a unit-semi-span, with 38 degrees of leading edge sweep. It has a modified trapezoidal planform, with straight taper from a root chord of 0.38, and a curved trailing edge in the inboard region blending into straight taper outboard of the 30 percent span station to a tip chord of 0.10, with an aspect ratio of 9.0. The initial wing sections were based on a section specifically designed by the first author's two-dimensional design method [9] to give shock free flow at Mach 0.78 with a lift coefficient of 0.6. This section, which has a thickness to chord ratio of 9.5 percent, was used at the tip. Similar sections with an increased thickness were used inboard. The variation of thickness was non-linear with a more rapid increase near the root, where the thickness to

chord ratio of the basic section was multiplied by a factor of 1.47. The inboard sections were rotated upwards to give the initial wing 3.0 degrees twist from root to tip. The two-dimensional pressure distribution of the basic wing section at its design point was introduced as a target pressure distribution uniformly across the span. This target is presumably not realizable, but serves to favor the establishment of relatively benign pressure distribution. The total inviscid drag coefficient, due to the combination of vortex and shock wave drag, was also included in the cost function. The calculations were performed with the lift coefficient forced to approach a fixed value by adjusting the angle of attack every fifth iteration of the flow solution. It was found that the computational costs can be reduced by using only 15 multigrid cycles in each flow solution, and in each adjoint solution. Although this is not enough for full convergence, it proves sufficient to provide a shape modification which leads to an improvement.

The initial geometry for a calculation decomposed into 16 subdomains is presented in Figure 2 where we can see both the surface of the wing and the symmetry plane. The lines on these two surfaces represent the footprints of the interprocessor boundaries for the 16 subdomains, which in this case are distributed in the following fashion: 4 wrapped around the wing airfoil sections, 2 in the direction normal to the wing, and 2 in the spanwise direction. Figures 6 and 7 show

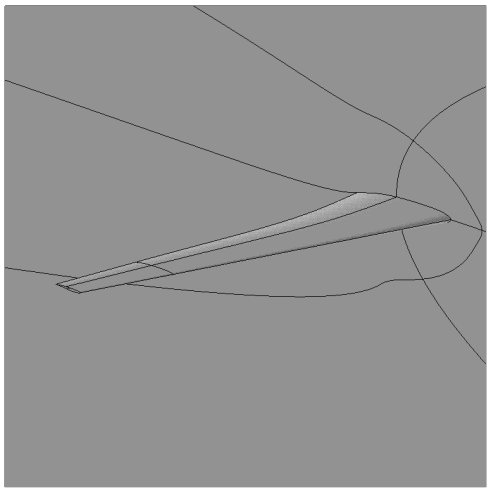


Fig. 2 Domain Decomposition for Initial Wing Configuration.

the result of a calculation at Mach number of 0.85, with the lift coefficient forced to approach a value of 0.5. This calculation was performed on a mesh with 192 intervals in the ξ direction wrapping around the wing, 32 intervals in the normal η direction and 48 intervals in the spanwise ζ direction, giving a total of 294912 cells. The wing was specified by 33 sections, each with 128 points, giving a total of 4224 design variables. The plots show the initial wing geometry and pressure distribution, and the modified geometry and pressure distribution after 40 design cycles. The total inviscid drag coefficient was reduced from 0.0207 to 0.0113. The initial design exhibits a very strong shock wave in the inboard region. It can be seen that this is completely eliminated, leaving a very weak shock wave in the outboard region. Figure 3 shows the variation of the coefficient of drag with each design iteration. It can be seen that the first few iterations have the most significant impact on the reduction of the coefficient of drag. After the drag has been reduced close to the final level, most of the time is spent in small changes in the pressure distribution which tends to a shock free solution. Next, we present results for the

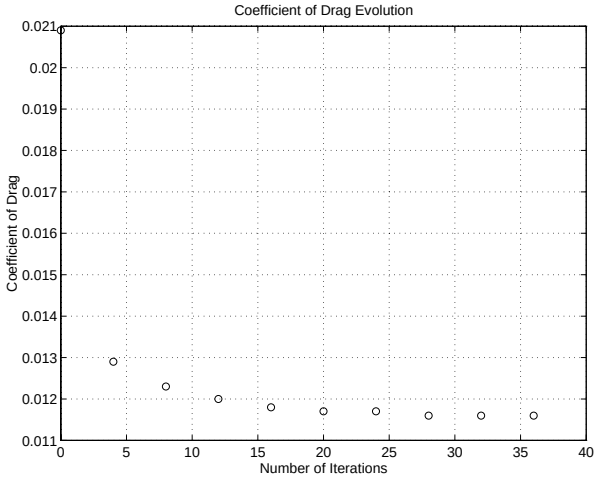


Fig. 3 Evolution of the Coefficient of Drag.

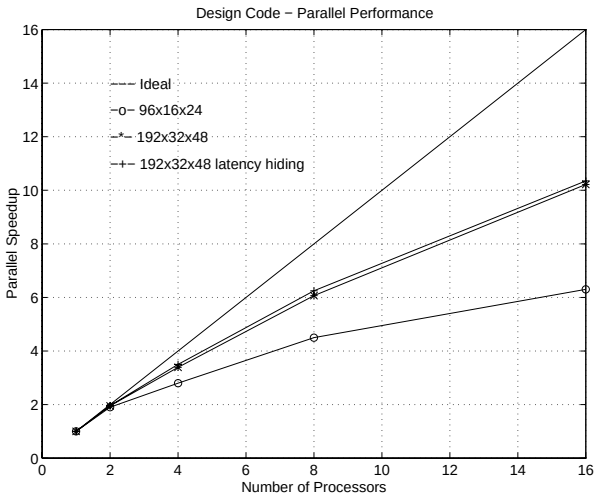


Fig. 4 Parallel Speedups for the Design Code SYN87P.

parallel performance of the complete design method. The summarized results can be seen in Figure 4 for both a $96 \times 16 \times 24$ and a $192 \times 32 \times 48$ mesh. The number of processors used in each calculation is represented in the abscissa and the parallel speedup is

shown in the ordinate. The parallel speedup is defined as the inverse of the ratio of execution times between the parallel code with a given number of processors and the most efficient serial version (without any message passing calls). Thus, the results for the single processor case are not those of the parallel code run for a single subdomain.

In all cases the maximum allowable number of multigrid levels for the 16 processor case is used (3 and 4 respectively) and a full W cycle is traversed (which poses the most severe communication requirements for a multigrid algorithm). It must be noted that these two test cases present an extremely demanding challenge for the parallel implementation. In the case of the coarser mesh, a relatively small mesh per processor is used. Moreover, in the coarsest levels of the multigrid sequence, very little computation is carried out while much of the time is wasted in sending pieces of information back and forth between processors. It must be also noted that the serial code is highly optimized, and any gains in the serial performance will usually detract from the parallel performance, since the cost of communication remains essentially fixed, while the amount of computation is drastically reduced.

In any case, parallel efficiencies of 77.5% and 66.25% can be achieved in the fine mesh for 8 and 16 processors respectively. It must be mentioned as an aside that the Navier-Stokes version of the flow solver has been also tested, and due to the decrease in granularity caused by the need of computing viscous fluxes, parallel performances around 80% can be obtained for 16 processors of an IBM SP2 system on this type of mesh.

The third curve in the plot shows the parallel speedup for the communication scheme that implements the latency hiding procedure described before. As can be seen, the results are only minimally improved from the previous ones. Although the gain in parallel performance due to the masking of the communication cost was expected to be higher, one must realize that, being the serial code a highly tuned program, destroying the stride used when computing the updates for all cells at a time hurts the performance of the code by incurring in a much larger number of cache misses. This option, though, remains a very valid approach for very large structured meshes or for unstructured mesh calculations, in which this idea fits quite nicely in the way in which the cells in a domain are updated.

SYN87P-MB Results and Performance

In this section we will present preliminary results of the parallel implementation of the automatic multiblock design method. This implementation is still in its early stages and further, more complete work will be reported at a later time.

The test case here presented attempts to improve the original business jet configuration by reducing its drag at a higher Mach number than the original design Mach number of 0.80. In this case the Mach number was set at 0.82 and C_L at 0.30. The design was run in fixed lift mode with C_D as the cost function to be minimized. For this case, a mesh with 72 blocks, $750,000$ cells, and 105 design variables using Hicks-Henne functions in the chordwise direction and linear lofting in the spanwise direction were used. To keep the wing from violating realistic constraints, the design variables were chosen so that the thickness distribution over the entire wing was preserved. This required some of the design variables to operate on up to four faces in different blocks simultaneously which reside in different processors. By appropriate selection of the design functions on each face it is ensured that different processors will update the geometry of mating blocks in the same fashion. The initial and final designs after 6 cycles are shown in Figure (8). Note that the shock on the wing upper surface has been largely eliminated over the entire span. The final configuration drag was reduced by 19%, most of which was due to the reduction of the drag on the wing. The reader is referred to[17] for a more complete description of this calculation.

In this case a mesh with 72 blocks was used. The block distribution between the 16 processors in the calculation can be seen in the last figure of the paper. This figure shows an isometric view of the geometry where the white lines denote the block boundaries in the mesh. Moreover, the faces in these blocks have been colored according to the processor they reside in. As one can appreciate it the generality of the method is such that quite complicated topological arrangements can be satisfied.

Finally we present parallel performance figures for the flow solver part of the multiblock design method. The parallel performance figures are presented for the case of a delta wing multiblock mesh with 36 blocks and a total of 518400 cells in the complete domain. The blocks were all of equal size and the computations were carried out for numbers of nodes in which the load balancing based on the number of cells in each processor was perfect. In Figure 5 we can see that the code scales quite nicely with a performance of 95% for 9 processors in the calculation. So far we have only preliminary timings for the complete parallel multiblock design method. These timings indicate that a complete design solution with 8 design cycles could be completed in under 3 hours using 16 processors of an IBM SP2 system. This type of fast turnaround is the objective that we were trying to achieve in order to make computational design tools truly useful in the industrial environment.

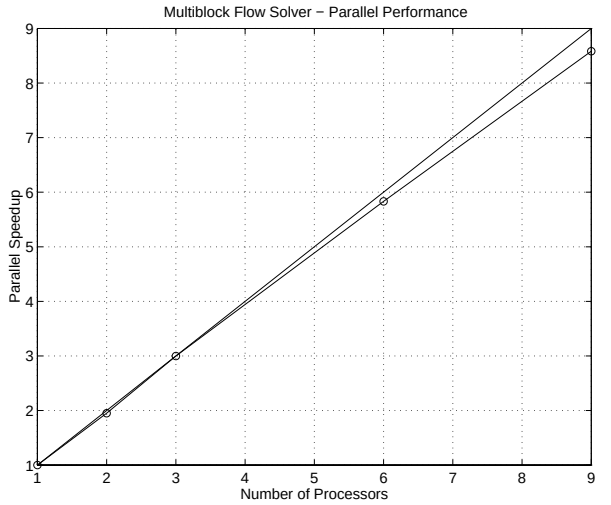


Fig. 5 Parallel Speedups for the Multiblock Flow Solver FLO87P-MB.

Conclusions

A parallel implementation of the automatic design optimization method has been developed using a domain decomposition approach and the MPI (Message Passing Interface) Standard for communication between processors. Both a single block and a multiblock version of the design philosophy have been implemented allowing for quick design optimization of isolated wings and full aircraft configurations. The results of the single block parallel code exactly reproduce those of the serial version, and fast turnarounds can be achieved using 16 or more processors. A typical design iteration can be completed in 66 seconds with 16 processors of an IBM SP2 system.

More importantly, the transition of CFD computer programs between traditional vector supercomputers and modern distributed memory machines has been successfully completed. Issues affecting the parallel performance of the method have been identified, and are the target of continuing research. Nevertheless, the parallelization strategy developed for the single block program carries through to the codes using multiblock topologies.

One important lesson learned during the course of this work is that sophisticated real-time parallel visualization tools (such as pV3[7]) are essential for the development and debugging of parallel programs. Advanced parallel visualization tools will enable on-line design and they will become necessary in an environment in which new optimized shapes are produced at about one minute intervals.

Acknowledgments

The authors wish to acknowledge the fact that the completion of this project was only possible due to a strong team effort. Members of this team included Luigi Martinelli, James Reuther, James Farmer, and the authors themselves. This research has benefited greatly from the generous support of the AFOSR under grant number AFOSR-91-0391. Computational time on the NAS SP2 system was provided as a part of the IBM-CRA initiative.

References

[1]J. J. Alonso, T. J. Mitty, L. Martinelli, and A. Jameson. A two-dimensional multigrid-driven Navier-Stokes solver for multiprocessor architectures. In *Proceedings of the Parallel CFD '94 Conference*, Kyoto, Japan, May 1994.

[2]Eran Arad. Visiting Scientist. Princeton University, 1995. Private communication.

[3]O. Baysal and M. E. Eleshaky. Aerodynamic design optimization using sensitivity anaysis and computational fluid dynamics. *AIAA paper 91-0471*, 29th Aerospace Sciences Meeting, Reno, Nevada, January 1991.

[4]W. S. Cheng, T. J. Mitty, and A. Jameson. AIRPLANE on the IBM parallel system SP1. In *Proceedings of the Parallel CFD '94 Conference*, Kyoto, Japan, May 1994.

[5]Hubertus Franke. IBM MPI-f developer. Private communication.

[6]A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, May 1993.

[7]R. Haimes. pV3: A distributed system for large-scale unsteady CFD visualization. *AIAA paper 94-0321*, 32nd Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1994.

[8]A. Jameson. Solution of the Euler equations by a multigrid method. *Applied Mathematics and Computations*, 13:327–356, 1983.

[9]A. Jameson. Automatic design of transonic airfoils to reduce the shock induced pressure drag. In *Proceedings of the 31st Israel Annual Conference on Aviation and Aeronautics, Tel Aviv*, pages 5–17, February 1990.

[10]A. Jameson. Optimum aerodynamic design via boundary control. Technical report, AGARD FDP/Von Karman Institute Special Course on Optimum Design Methods in Aerodynamics, Brussels, April 1994.

[11]A. Jameson. Optimum aerodynamic design using CFD and control theory. *AIAA paper 95-1729*, AIAA 12th Computational Fluid Dynamics Conference, San Diego, CA, June 1995.

[12]J.L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, New York, 1971. Translated by S.K. Mitter.

[13]Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, March 1994.

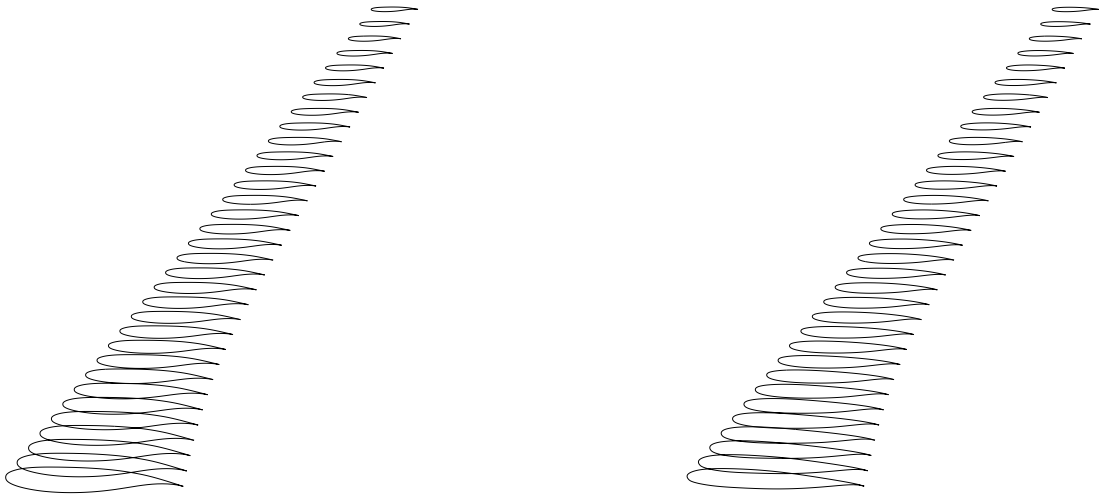
[14]H. S. Morse. *Practical Parallel Computing*. Academic Press, Cambridge, MA, 1994.

[15]O. Pironneau. *Optimal Shape Design for Elliptic Systems*. Springer-Verlag, New York, 1984.

[16]J. Reuther and A. Jameson. Aerodynamic shape optimization of wing and wing-body configurations using control theory. *AIAA paper 95-0213*, 33rd Aerospace Sciences Meeting and Exibit, Reno, Nevada, January 1995.

[17]J. Reuther, A. Jameson, J. Farmer, L. Martinelli, and D. Saunders. Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation. *AIAA paper 96-0094*, AIAA 34th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1996.

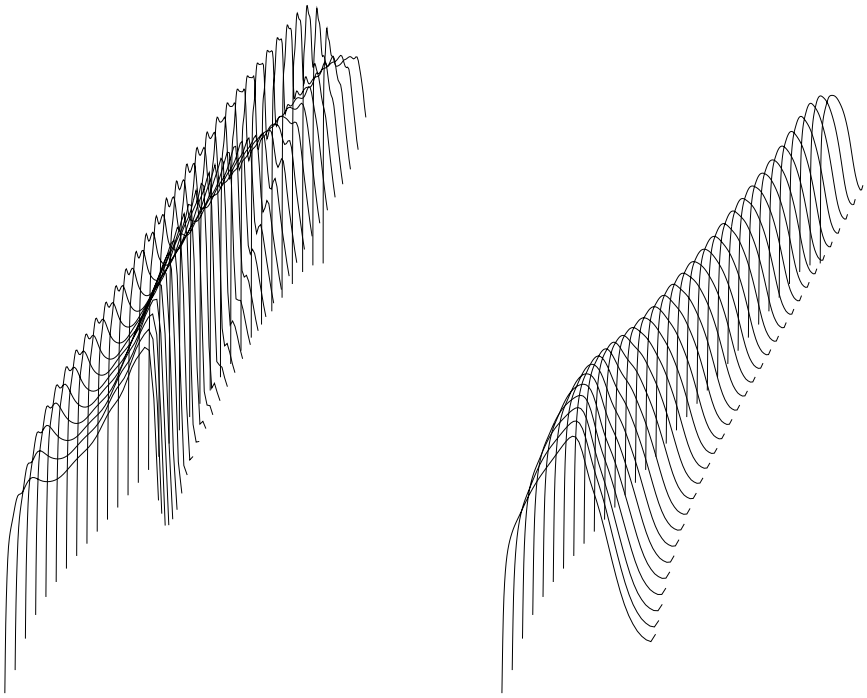
[18]S. Ta'asan, G. Kuruvila, and M. D. Salas. Aerodynamic design and optimization in one shot. *AIAA paper 91-005*, 30th Aerospace Sciences Meeting and Exibit, Reno, Nevada, January 1992.



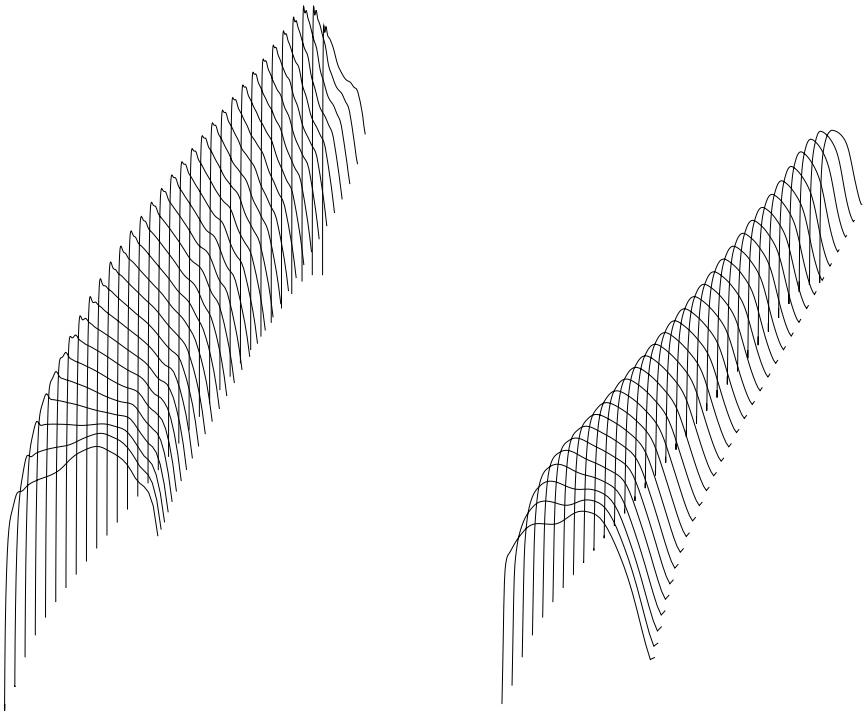
a) Initial Wing, $C_l = 0.5001$, $C_d = 0.0207$, $\alpha = -1.340°$

b) 40 Design Iterations, $C_l = 0.5000$, $C_d = 0.0113$, $\alpha = -.235°$

Fig. 6 Lifting Design Case, $M = 0.85$, Fixed Lift Mode. Drag Reduction

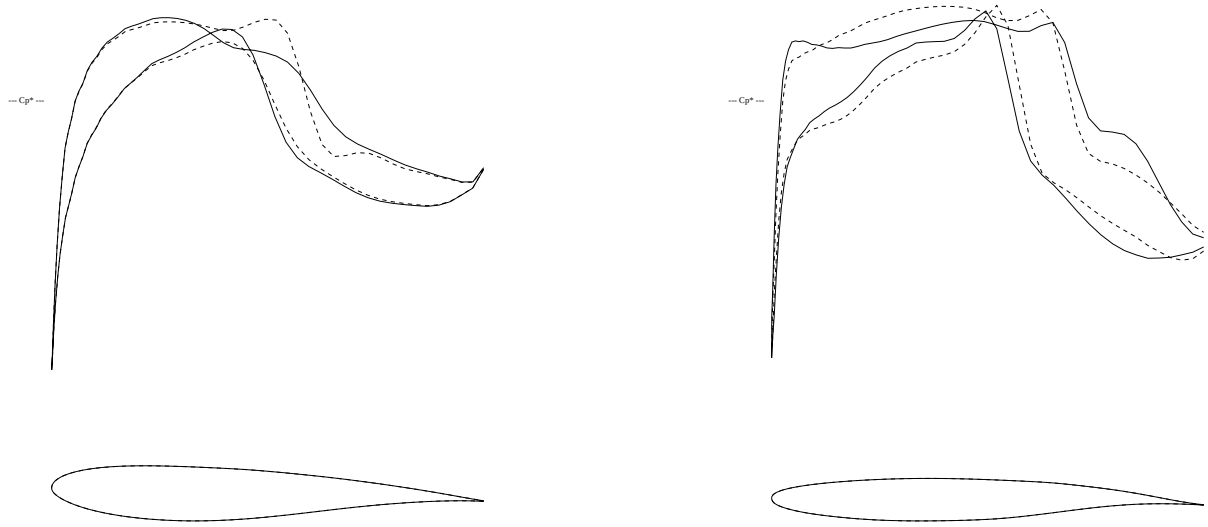


a) Initial Wing. Lifting Design Case, $M = 0.85$, Fixed Lift Mode. $C_L = 0.5001$, $C_D = 0.0207$, $\alpha = -1.340°$. Drag Reduction



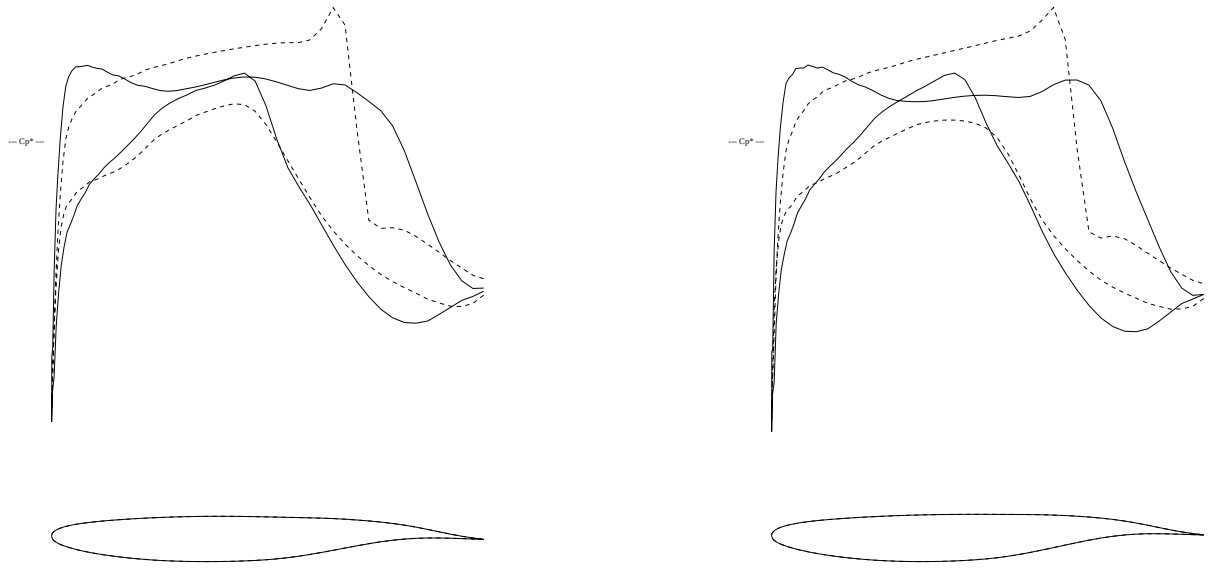
b) 40 Design Iterations. Lifting Design Case, $M = 0.85$, Fixed Lift Mode. $C_L = 0.5000$, $C_D = 0.0113$, $\alpha = -0.235°$. Drag Reduction

Fig. 7 Lifting Design Case, $M = 0.85$, Fixed Lift Mode. Drag Reduction



a) span station $z = 0.125$

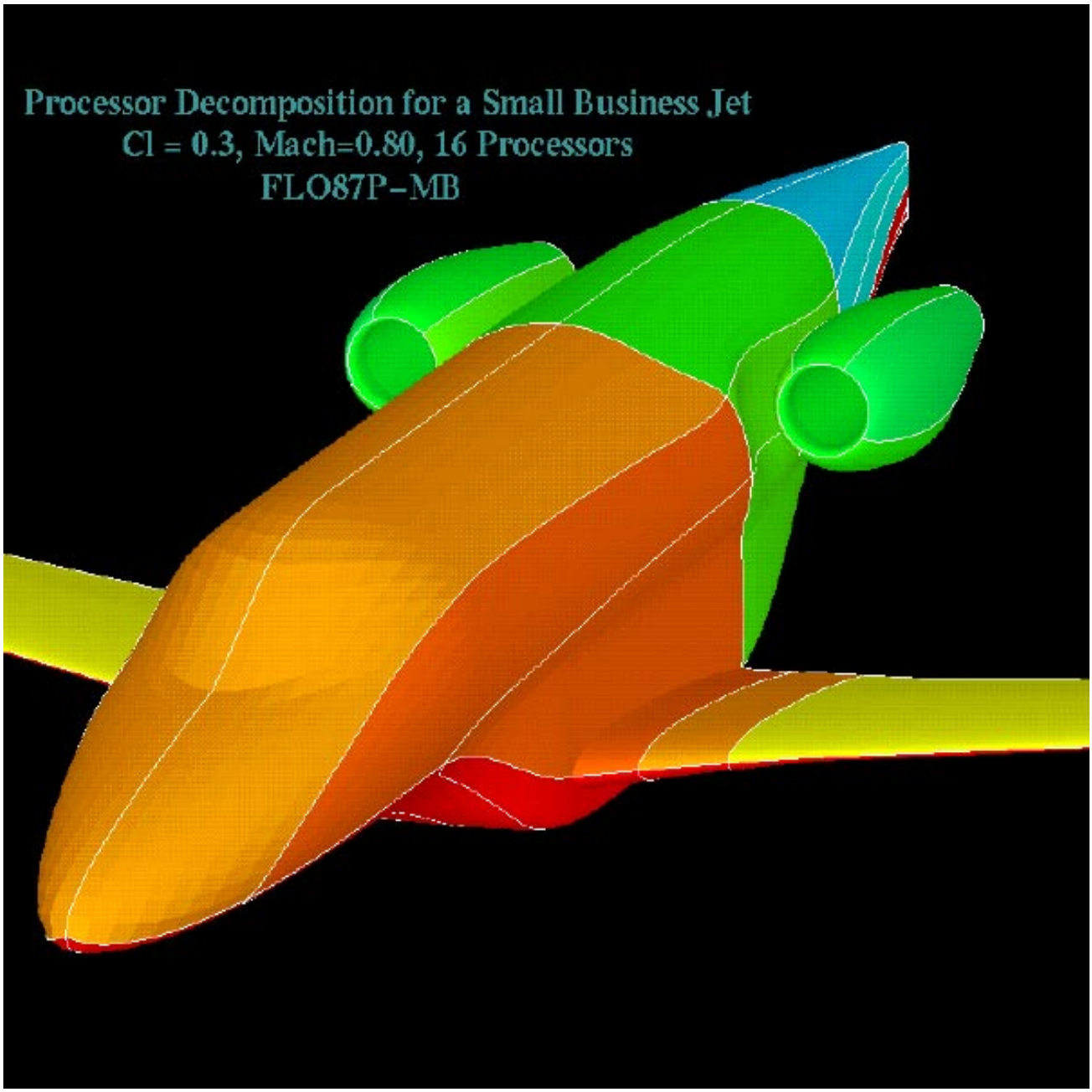
b) span station $z = 0.312$



c) span station $z = 0.559$

d) span station $z = 0.764$

Fig. 8 SYN87-MB, Fixed Lift Drag Minimization. 72 Block Mesh, 750 K mesh cells, $M = 0.82$, $C_L = 0.3$ 105 Hicks-Henne variables. - - -, Initial Pressures —, Pressures After 6 Design Cycles.


Processor Decomposition for a Small Business Jet
Cl = 0.3, Mach=0.80, 16 Processors
FLO87P-MB