



2

AD-A263 395



NRL/MR/6410--93-7178

A Parallel Implicit Incompressible Flow Solver Using Unstructured Meshes

RAVI RAMAMURTI
RAINALD LÖHNER

*Center for Reactive Flow and Dynamical Systems
Laboratory for Computational Physics and Fluid Dynamics*

January 17, 1993



93-08454

93-08454



35P

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE January 17, 1993	3. REPORT TYPE AND DATES COVERED Interim		
4. TITLE AND SUBTITLE A Parallel Implicit Incompressible Flow Solver Using Unstructured Meshes		5. FUNDING NUMBERS PE-63569E4 PR-M004 WU-M004		
6. AUTHOR(S) Ravi Ramamurti and Rainald Löhner				
7. PERFORMING ORGANIZATION NAME(S) and ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5320		8. PERFORMING ORGANIZATION REPORT NUMBER NRL/MR/6410-93-7178		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 N. Fairfax Drive, Arlington, VA 22203-1714		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) An incompressible flow solver based on unstructured grids is implemented on a parallel distributed memory computer architecture. An important phase in the flow solver is the solution of the elliptic equations for the velocities and pressure. This elliptic solver is parallelized and incorporated into both the explicit and implicit versions of the incompressible flow solver. Performance and scalability studies are carried out on both Intel iPSC 860 and the Intel Delta prototype, and show that the code is scalable. A parallelizable load balancing algorithm is developed to be used in conjunction with the incompressible flow solver. Steady and unsteady flows over a tri-element airfoil and NACA0012 airfoil are computed using the parallel incompressible flow solver.				
14. SUBJECT TERMS Parallel Computation Unstructured meshes Incompressible flows			15. NUMBER OF PAGES 33	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

CONTENTS

Introduction	1
The Incompressible Flow Solver	4
Domain Decomposition	4
Parallel Implementation	5
Results and Discussion	6
Explicit Incompressible Flow Solver	8
Implicit Incompressible Flow Solver	9
Summary and Conclusions	10
Acknowledgements	10
References	10

DTIC QUALITY INSPECTED 4

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

A PARALLEL IMPLICIT INCOMPRESSIBLE FLOW SOLVER USING UNSTRUCTURED MESHES

Ravi Ramamurti and Rainald Löhner*

Laboratory for Computational Physics and Fluid Dynamics,
Naval Research Laboratory, Washington, D.C. 20375

* The George Washington University, Washington, D.C. 20052

Introduction

Recent innovations in microprocessor technology have resulted in a variety of architectures for parallel computers. The two most commonly available and used are the single instruction, multiple data (SIMD) architecture, in which an instruction is carried out simultaneously on multiple sets of operands, and the multiple instruction, multiple data (MIMD) architecture, where several different instructions may be concurrently operating on several distinct sets of operands. Examples of SIMD machines are the Connection Machines CM-1, CM-2 and CM-200 from Thinking Machines, Inc., and the MasPar-1 from MasPar. Examples of MIMD machines are the Intel hypercubes and Meshes (Gamma, Delta prototypes, Paragon), as well as the nCUBE and Parsytech hypercubes. The MIMD multiprocessor distributed memory architecture has seen the largest growth in the recent years and has made significant advances in user-friendliness.

For efficient use of any parallel computer, first an appropriate architecture has to be chosen for a given algorithm, or the algorithm has to be tailored to suit the given architecture. Secondly, the problem to be solved is split into several pieces. Each piece is then handed over to a processor with interprocessor information transfer provided. Finally, all pieces are assembled. All these steps should be done without incurring substantial overhead both in terms of reduced computational efficiency and increased communication costs.

Typical problems of interest are unsteady aerodynamic flows and turbulent separating flows over complex vehicles such as submarines. The end users of codes developed for these problems are the aerodynamic and hydrodynamic vehicle designers. To be successful in a time-constrained design environment a code must be capable of rapid mesh generation and flow computation so that an adequate range of alternative configurations can be studied in a timely manner.

For the simulation of flows about complex geometries such as the fully appended submarine, an unstructured grid approach offers the greatest flexibility with the fewest degrees of freedom. Furthermore, the method allows straightforward adaptive meshing strategies for dynamic resolution in transient problems. This paper describes the parallel implementation

of FEFLOIC, which is an implicit finite-element incompressible transient Navier-Stokes code for the simulation of 2-D flows. The important aspects of this algorithm are the unstructured mesh generation and the implicit flow solver. These aspects dictate which type of parallel computer is best suitable. For example, the mesh generation using the advancing front algorithm [1] will require different tasks to be performed in different subdomains. This will necessitate the use of a MIMD architecture. The implicit flow solver employs linelets as a preconditioner to achieve improved convergence rates [2,3]. The use of a SIMD architecture would imply that either linelets have to be used in all subdomains or the linelets are not used anywhere. The latter case implies that the flow solution is advanced in an explicit manner, resulting in a reduced computational efficiency. Therefore, in order to retain an implicit solver in regions such as the boundary layer and the capability to explicitly advance the flow solution in regions where the cell sizes are large, one has to resort to a MIMD architecture. Another disadvantage of the SIMD type of computer is that the communication is inefficient if it is not between nearest neighbours. In an unstructured mesh, the communication pattern is quite irregular. However, mapping techniques [4] can be applied to assign vertices of the mesh to processors of the computer, thus minimizing the communication cost. This mapping is quite costly and is not pragmatic in situations where adaptive remeshing is essential, for example, tracking of vortices in the wake.

The present research effort is directed towards the parallel simulation of transient flows with high Reynolds numbers. In particular, the goal is to simulate transient separating flow about a fully appended submarine and a self-consistent maneuvering trajectory. Accurate representation of the very near-wall behaviour of the velocity field will be critical in realistically simulating the unsteady three dimensional separation which exists on maneuvering submarines. This will necessitate the use of increasingly fine elements ($y^+ < 10$) toward the wall. This precludes the use of an explicit solver due to the prohibitively small timesteps associated with the elements in the near wall region. Hence in the present formulation, the pressure as well as the advection-diffusion terms of the Navier-Stokes equations are treated in an implicit manner. The elliptic equations for pressure and the velocities are solved using a preconditioned conjugate gradient algorithm. Details of the flow solver are given by Löhner and Martin [2,3]. This algorithm has been successfully evaluated by Ramamurti and Löhner [5] for several flow problems, such as the flow over a backward facing step, the flow past a circular cylinder and unsteady flow over airfoils.

For the above algorithm to be parallelizable, it should be broken into several smaller components and executed without incurring substantial overhead. Load balancing and communication are, therefore, important issues to be addressed in order to reduce this overhead. The primary objective in load balancing is to divide the workload equally among all the

processors. For field solvers based on grids, the computational work is proportional to N^p , where N is the number of elements. The exponent $p = 1$ for explicit time-marching or iterative schemes; $p > 1$, for implicit time-marching schemes or direct solvers. This implies that optimal load balancing is achieved if each processor is allocated the same number of elements. Besides the CPU time required in each processor to advance the solution one time-step, communication overhead between the processors should be taken into account. This overhead is directly proportional to the amount of information that needs to be transferred between processors. For field solvers based on grids, the amount of information that needs to be transferred is proportional to the number of surface faces in each of the subdomains. Therefore, an optimal load and communication balance is achieved by allocating to each processor the same number of elements while minimizing the number of surface faces in each subdomain. The load balancing algorithm used in the current work is described by Löhner et. al. [6]. This algorithm is based on the concept that elements are exchanged between the subdomains along the interfaces. The final subdivision having balanced workload is achieved through an iterative process. A heuristic method is employed to minimize the surface to volume ratio, i.e., the communication to computational load ratio. Furthermore, the decomposition insists on simply connected subdomains, again to reduce communication costs. Additional renumbering of the subdomains is done in an effort to minimize the number of communication hops between subdomains, a step important for a mesh type architecture such as the Intel Delta prototype.

In this paper, a description of the implicit flow solver and the load balancing algorithm are given. An important phase in the the flow solver is the solution of the elliptic equations for the velocities and pressure. A preconditioned conjugate gradient algorithm is employed to solve these equations. As a first step of the parallelization of this elliptic solver, a model problem of solving the heat conduction equation is considered. Results are obtained for the problem of a circular cylinder in a heated bath. Next, an explicit version of the incompressible flow solver is parallelized. This is applied for solving steady flow over a NACA0012 airfoil at zero angle of attack. For time accurate simulations, this explicit algorithm is not suitable since, as discussed earlier, it would require prohibitively small timesteps. Hence, the implicit flow solver with linelets as the preconditioner for solving the elliptic equations is parallelized. With some initial studies, it was quite apparent that the linelets should not be split between subdomains. Hence, the partitioning algorithm was modified to accomodate a set of complete linelets within a subdomain. The algorithm was then tested via simulation of unsteady flow past NACA0012 airfoil at $\alpha = 20^\circ$. Scalability and performance studies were done for all of the above-mentioned algorithms.

The Incompressible Flow Solver

The incompressible Navier-Stokes equations in the arbitrary Lagrangian Eulerian (ALE) form can be written as

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}_a \cdot \nabla \mathbf{v} + \nabla p = \nabla \cdot \sigma , \quad (1a)$$

$$\nabla \cdot \mathbf{v} = 0 , \quad (1b)$$

where

$$\mathbf{v}_a = \mathbf{v} - \mathbf{w} . \quad (1c)$$

Here p and σ denote the pressure and the stress tensor scaled by density, $\mathbf{v}$ the flow velocity and $\mathbf{w}$ the mesh velocity.

The governing equations (1a-b) are first discretized in time and the resulting equations are linearized. This is followed by the discretization in space. Linear elements with equal order interpolation for velocities and pressures are used. The resulting matrix system is solved sequentially using a projection-like method [7,8]. Thus, the velocities are updated first. This is followed by the solution of a Poisson-type equation for the pressures. The role of this step is to project the predicted velocity field into a divergence-free field. From the pressures, a new velocity field is obtained. This two-step process can be repeated several times until convergence is obtained. However, it is seldom applied in practice: our experience indicates that one pass is sufficient for accurate solutions. The large linear systems of equations that appear during both the steps are solved iteratively with a preconditioned conjugate gradient algorithm. As a preconditioner, linelets are employed. More details of the discretization, the construction of the linelets and the solution of the Poisson equation are given in [2].

Domain Decomposition

A simple algorithm based on the 'wavefront'-type scheme is used to obtain a subdivision into N_{DOMN} subdomains. Briefly, this algorithm can be described as follows. We assume that the work required by each element, a desired work per domain, and the elements that surround each point are given. To start the process, any given point can be selected. All the elements surrounding this point are then marked as belonging to the present subdomain. The cumulative work for the elements of this domain is updated. The points of each of these elements which are unsurrounded yet, are stored in a list LHELP. If the desired work per domain has been exceeded, a new domain is started. Otherwise, the next point to be surrounded is selected from the order of creation list LHELP. The procedure continues until all

elements have been allocated. More details of this domain splitting algorithm are described in [9].

The load balancing algorithm assumes that an initial subdivision with N_{DOMN} subdomains is supplied through the domain splitting algorithm, or a parallel grid generator [9], or from a previous timestep of a flow computation. Given the measure of work required by each element, the workload in each subdomain is summed up. The surplus/deficit of this workload with the desired average workload is computed for each element. A deficit difference function is computed in each element that reflects the imbalance in surplus/deficit between the element and its neighbours. Elements are then added to subdomains with negative deficit function. This process is repeated until a balanced subdivision is obtained. The choice of the deficit difference function and details of the algorithm are described in [6].

Parallel Implementation

Having partitioned the domain into several subdomains, each subdomain is assigned to a processor. The subdomains formed are such that there is one layer of elements overlapped between adjacent subdomains. The information at the vertices associated with this layer of elements are communicated between processors.

An important phase in the flow solver algorithm is the solution of the elliptic equations using the preconditioned conjugate gradient algorithm (PCG). This algorithm has been described by Saad [10].

Given an initial guess x_0 to the solution of the linear system $Ax = b$, the PCG algorithm is as follows.

1. Compute the preconditioner based on the linelets M .
2. Start the process by setting $r_0 = b - Ax_0$, $p_0 = z_0 = M^{-1}r_0$.
3. Iterate until convergence
 - (a) $w = Ap_i$
 - (b) $\alpha_i = (r_i, z_i)/(w, p_i)$
 - (c) $x_{i+1} = x_i + \alpha_i p_i$
 - (d) $r_{i+1} = r_i - \alpha_i w$
 - (e) $z_{i+1} = M^{-1}r_{i+1}$
 - (f) $\beta_i = (r_{i+1}, z_{i+1})/(r_i, z_i)$
 - (g) $p_{i+1} = z_{i+1} + \beta_i p_i$

In the above algorithm, the dot products in step 3b and 3f are potential bottlenecks on many parallel or vector machines. This is because when all the vectors in the algorithm can be split equally among the processors, dot products require global communication. Therefore, the algorithm is synchronized at these two steps and a global sum of the scalars α and β

are obtained. This is important for the stability of the PCG algorithm. After step 3g, the relevant components of the vectors p and x are exchanged between the processors. To facilitate the transfer of information between the various processors, a list of points that are to be sent by a processor and those to be received by it are maintained. These lists consist of the processor to which the information has to be sent or received from and the local pointers. The information that is sent is then packed into a common block and communicated synchronously. On an Intel iPSC 860, this is done using CSEND and CRECV across processors.

The communication overhead can be reduced by performing computations in regions which do not depend on the interface boundaries. Then communication can be done in an asynchronous manner by posting all the IRECVs before the CSENDS on an iPSC 860. Fyfe [11] has compared the communication cost between the asynchronous and synchronous communications in the parallel version of the non-orthogonal Flux Corrected Transport algorithm based on structured grids. The gain through asynchronous communication was about 5% of the total time.

In order to take advantage of this asynchronous communication, first the interior points of a subdomain should be handled separately from the points along the communication interfaces. In an unstructured grid, this would result in an additional computational overhead. Secondly, the dot products in steps 3b and 3f of the PCG algorithm have to be completed before anything else can be done and no other computation can be performed while they are being computed. Saad [10] has outlined a few ways of overcoming this difficulty but warns that they could lead to an unstable algorithm. Hence, in the present effort, synchronous communication was deemed the most appropriate for this algorithm.

Results and Discussion

Model Problem

The parallel implementation of the PCG algorithm was first tested via application to a model problem of solving the heat conduction equation:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot k \nabla T + S, \quad (2a)$$

$$T = T_0(t) \text{ on } \Gamma_D, \quad \mathbf{n} \cdot k \nabla T = q_0(t) \text{ on } \Gamma_N. \quad (2b)$$

Here ρ, c_p, T, k, S denote the density, specific heat at constant pressure, temperature, conductivity, sources respectively. The boundary conditions are prescribed temperature T_0 on Dirichlet boundaries Γ_D and prescribed heat flux q_0 on Neumann boundaries Γ_N . As with the

incompressible flow solver, linear finite elements are used, and the discretization is obtained from the standard Galerkin weighted residual method.

The problem considered for this purpose, is to obtain a steady state temperature distribution around a cylinder immersed in a hot bath. In order to obtain a steady state solution, the explicit version of the elliptic solver with diagonal preconditioning (no linelets) is employed. For scalability studies of this algorithm, a coarse grid consisting of 1,526 points and a fine grid consisting of 9,767 points were chosen. The performance of the explicit solver in terms of both the speed-up (S) and the CPU time taken per point per timestep (τ) are shown in figures 1a and 1b respectively. These results are also summarized in Table 1. The performance of the algorithm both with and without I/O operations from the disk are shown. For a computationally bound problem, such as the simulation of a transient flow, the performance obtained by not considering the I/O will be a true representative value of the overall performance of the solver. From Fig. 1a, it can be seen that the S obtained using the coarse grid is close to the ideal linear curve for 4 processors and drops thereafter. Also, it is clear that the performance in terms of S for 16 processors is decreased from 8.0 to 6.2 when I/O time is considered. This is because of the fact that the number of points in each subdomain is only around 100, and therefore, the I/O time is an overwhelming part of the total time. The I/O overhead can be reduced if one were to use unformatted read/write operations. As a finer grid is used, the algorithm scales very well and the S is improved considerably, achieving a value of almost 12.7 for 16 processors, compared to a value of 8.0 obtained using the coarse grid. Even for the finer grid employed, the number of points per subdomain with 16 processors is only about 600, out of which almost 10% of these points are located along the communication interfaces. Hence, the deviation of the performance of the solver without I/O when 16 processors are employed, is largely due to the increased communication/computation ratio. All of the performance results obtained in the present study are obtained using 64-bit floating-point arithmetic. This is important because the 64-bit arithmetic is the appropriate one for this algorithm, and performance results based on 32-bit arithmetic could be misleading [12] if one were to compare it to other systems such as the CRAY.

The finer grid that was employed in this study and the temperature distribution after 500 timesteps using 8 processors are shown in Figs. 2a and 2b respectively. It can be seen from Fig. 2b that the solution has achieved steady state. This solution was identical to the one obtained using a single domain. This provides reasonable confidence in the parallel implementation of the elliptic solver.

Explicit Incompressible Flow Solver

The parallel elliptic solver was incorporated into the incompressible flow solver. Results were obtained for steady flow past a NACA0012 airfoil at $\alpha = 0^\circ$ and a Reynolds number $Re = 300$. For this viscous flow, the grid that was employed is semi-structured and consists of 2,992 points and 5,860 triangular elements. This coarse grid was chosen so as to fit into one 8K processor on the Intel iPSC 860 (Gamma) available at the Naval Research Laboratory. The PCG algorithm uses diagonal preconditioning, which is equivalent to an explicit time-marching scheme. In order to achieve steady state, the maximum allowable timestep in each element is employed.

The performance results of this explicit incompressible flow solver are shown in Fig. 3 and are also summarized in Table 2. The effect of compiler optimization on the performance is also shown. A speed-up of 10.18 is obtained using 16 processors. From these results it is clear that the algorithm is scalable. The use of compiler optimization is to reduce the S obtained with 16 processors to 8.98. This reduction is due to the fact that with compiler optimization, the computational time per processor is decreased while the communication overhead remains a constant. It is clear from Table 2 that for 16 processors, the CPU time per point per timestep is reduced by about 20% with the use of compiler optimization.

The Laboratory for Computational Physics and Fluid Dynamics at NRL has, as a result of DARPA and NRL support, a 32-node Intel iPSC860 Touchstone Gamma. DARPA has also made computing time available to us on the 512 node Intel Delta prototype at CalTech. Scalability studies for 2,4,8,16 and 32 processors were carried out on the NRL Gamma prototype and a 64 processor computation was carried out on the Delta. For this purpose a finer grid consisting of approximately twice the number of points of the coarse grid was employed. The performance results on the Delta and the comparison with the Gamma are shown in Fig. 4. It is clear that the code is scalable. It can be seen that as the number of processors is increased for a constant problem size, the S deviates from the ideal linear curve. This is a result of the increased communication/computation ratio. There is considerable improvement in S as the problem size is increased, as would be expected. Even for this finer mesh, the performance drops down when the number of processors employed is greater than 16. This is again due to the fact that the communication overhead is overwhelming the total computational cost. There is clearly no benefit to be obtained by increasing the number of processors further for this constant size problem. Figures 5a and 5b show the coarse grid and the steady state pressure in the vicinity of the airfoil. The subdomain interfaces are superimposed on the pressure contours in Fig. 5b.

This explicit flow solver was then applied to solve the inviscid flow past a complex landing

configuration tri-element airfoil. The grid that was employed, the partitioning consisting of 32 subdomains are shown in Figs. 6a and 6b respectively. The grid consists of 8,952 points and 17,127 elements. The results obtained at $\alpha = 5^\circ$ after 500 timesteps are shown in Fig. 7. This figure shows the pressure, the contours of absolute velocity and the velocity vectors with the interface boundaries superposed on each of them. Figure 7c shows the presence of a recirculating region behind the main section of the airfoil. The total CPU time taken for the 500 steps is 14 minutes and 6 seconds resulting in $\tau = 0.189$ mSec/point/timestep.

Implicit Incompressible Flow Solver

For the simulation of unsteady separated flows, the typically fine elements in the boundary layer would force prohibitively small timesteps with an explicit solver for the advection terms. Hence, the implicit flow solver is parallelized next. Here, the linelets are used as a preconditioner in the PCG algorithm. The parallel algorithm was applied for solving the unsteady flow past the NACA0012 airfoil at $\alpha = 20^\circ$. The grid that was employed is the coarse grid consisting of 2,992 points. This grid was partitioned as before, and each subdomain handed to a processor. Linelets were then constructed in each of these subdomains and the flow was advanced in each of these subdomains. It was quite apparent that the PCG algorithm was not stable, and the maximum residue in the equation solved would occur near the communication interface where a linelet is split. Hence, the domain partitioning algorithm was modified to accomodate a complete set of linelets within a subdomain. This was done by supplying to the domain splitting algorithm the additional information about the points that would be a part of a linelet. First, a point along a linelet is chosen. All the elements surrounding this point are then marked belonging to the present subdomain. The next point is selected from the order of creation of this linelet. The process is continued until all the points along this linelet are exhausted. When the number of elements belonging to this subdomain exceeds a set average value, the subdomain counter is updated. In regions of flow where the large aspect ratio cells are not required, such as the region outside the boundary layer and the wake, the advection terms can be advanced in an explicit manner and no linelet preconditioning is necessary. These regions are subdivided next.

The comparison of the performance of the implicit and the explicit versions of the code is shown in Fig. 8. It is clear that there is very little difference in S obtained due to the addition of the linelet preconditioning. One main reason for this is in the boundary layer regions only one set of linelets, aligned closely along the normal to the airfoil, is employed. This results in a tridiagonal system to be inverted at the preconditioning step (Steps 2 and 3e) of the PCG algorithm. The inversion of this system is very efficient. Another reason is that the linelets are confined to regions of high aspect ratio cells, which are typically in the

boundary layer and the wake.

Figure 9a shows a grid consisting of 6,785 points and 13,376 elements that was employed for computing the flow over NACA0012 airfoil at $\alpha = 20^\circ$ at $Re = 300$. This flow adapted grid was obtained from a previous run for the same flow configuration on a vector machine. This grid was then subdivided into 32 subdomains. Fig. 9b shows the domain decomposition in the vicinity of the airfoil. Figures 9c-e show the pressure, vorticity and the velocity vectors of the unsteady flow after 25 characteristic times. The Strouhal number St for this flow is approximately 0.43. The CPU time taken for 500 timesteps is 13 minutes and 12.9 seconds resulting in $\tau = 0.2337$ mSec/point/timestep. A grid consisting of 10,952 points and 21,640 elements was employed for computing the flow over a NACA0012 airfoil at $Re = 1.4 \times 10^6$ using the CRAY YMP, Convex C210 and the Intel iPSC 860. This performance study showed that the CPU time taken per point per timestep (τ) on the Intel iPSC 860 using 32 processors is 0.2 mSec. compared to 0.16 mSec. on one processor YMP. The corresponding value of τ for the Convex is 1.46 mSec.

Summary and Conclusions

An incompressible flow solver based on unstructured grids was successfully implemented on a MIMD architecture, viz., Intel iPSC 860. The parallel implementation of the elliptic solver was tested using the heat conduction equation as a model problem. The parallelized elliptic solver was then incorporated into the explicit and implicit versions of the incompressible flow solver. Performance studies were performed on all of these algorithms. Scalability studies were performed on both the Gamma and the Delta prototypes, and show that the code is scalable. A parallelizable load balancing algorithm was developed to be used in conjunction with the incompressible flow solver. Future developments will be focused on the incorporation of adaptive regridding in 2-D and the parallel implementation of the 3-D incompressible flow solver.

Acknowledgements

This work was supported by the DARPA Submarine Technology Program with Mr. Gary Jones as the program manager. The authors wish to acknowledge useful discussions throughout this work with Dr. W.C. Sandberg of NRL.

References

1. Löhner, R. and Parikh, P., "Three-Dimensional Grid Generation by the Advancing Front Method," *Int. J. Num. Meth. Fluids* 8, pp. 1135-1149, 1988.

2. Löhner, R. and Martin, D., "An Implicit, Linelet Based Solver for Incompressible Flows," *Advances in Finite Element Analysis*, FED Vol. 123, Editors: M.N. Dhaubhadel et. al., ASME Publication, New York, 1991.
3. Martin, D. and Löhner, R., "An Implicit Linelet-Based Solver for Incompressible Flows," AIAA-92-0668, 1992.
4. Dahl, E.D., "Mapping and Compiled Communication on the Connection Machine," *Proceedings of Distributed Memory Computing Conference V*, IEEE Computer Society Press, April 1990.
5. Ramamurti, R. and Löhner, R., "Evaluation of an Incompressible Flow Solver Based on Simple Elements," to be presented at the ASME Winter Annual Meeting, Anaheim, CA, November 1992.
6. Löhner, R., Ramamurti, R. and Martin, D., "A Parallelizable Load Balancing Algorithm," to be presented at the AIAA Aerospace Sciences Meeting, Reno, NV, 1993.
7. Chorin, A.J., "Numerical Solution of the Navier-Stokes Equations," *Math. Comp.* **22**, pp. 745-762, 1968.
8. Donea, J., Giuliani, S., Laval, H. and Quartapelle, L., "Solution of the Unsteady Navier-Stokes Equations by a Fractional Step Method," *Comp. Meth. Appl. Mech. Eng.* **30**, pp. 53-73, 1982.
9. Löhner, R., Camberos, J. and Merriam, M., "Parallel Unstructured Grid Generation," *Comp. Meth. Appl. Mech. Eng.* **95**, pp. 343-357, 1992.
10. Saad, Y., "Krylov Subspace Methods on Supercomputers," *SIAM J. Sci. Stat. Comput.* **10**, No. 6, pp. 1200-1232, 1989.
11. Fyfe, D.E., private communication, 1992.
12. Bailey, D.H., "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers," *Supercomputing Review*, pp. 54-55, August 1992.

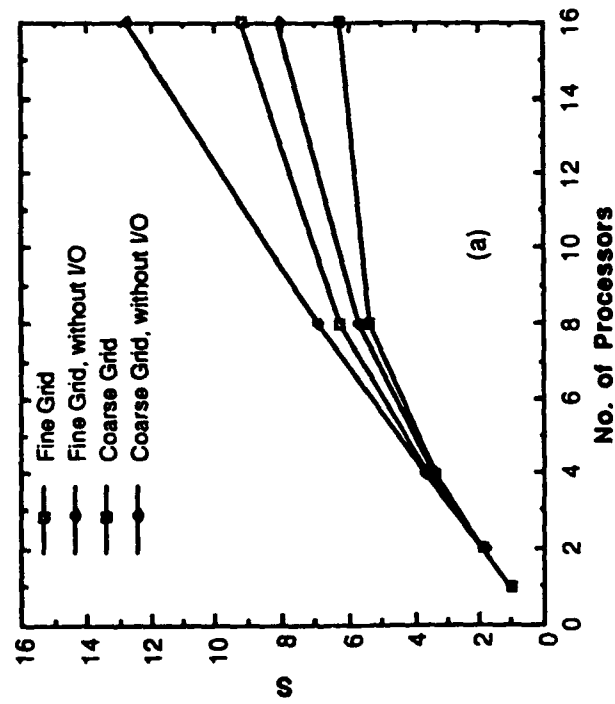
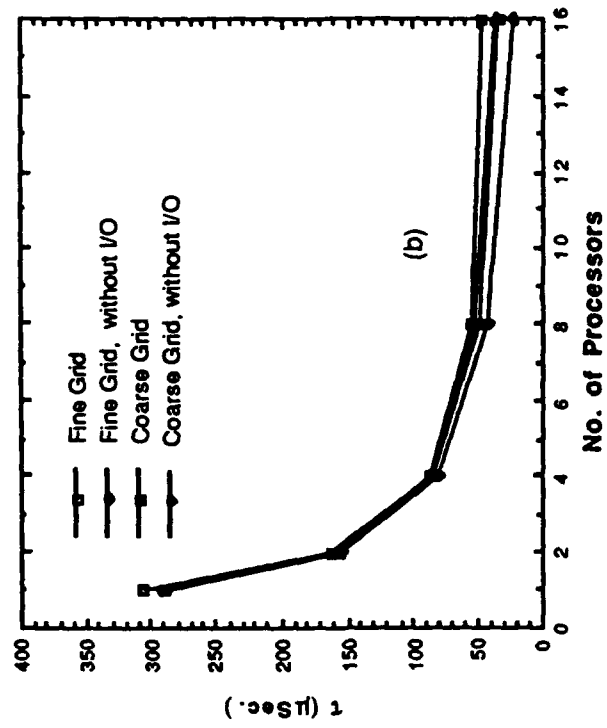


Fig. 1. Performance of FEHEAT

No. of Proc.	Fine Grid		Coarse Grid	
	τ_{avg}	τ_{avg}^*	τ_{avg}	τ_{avg}^*
1	305.126	291.916	290.068	288.118
2	162.568	154.752	155.716	153.936
4	85.783	80.287	87.281	85.028
8	48.897	42.563	54.620	50.763
16	33.167	22.986	46.779	35.928

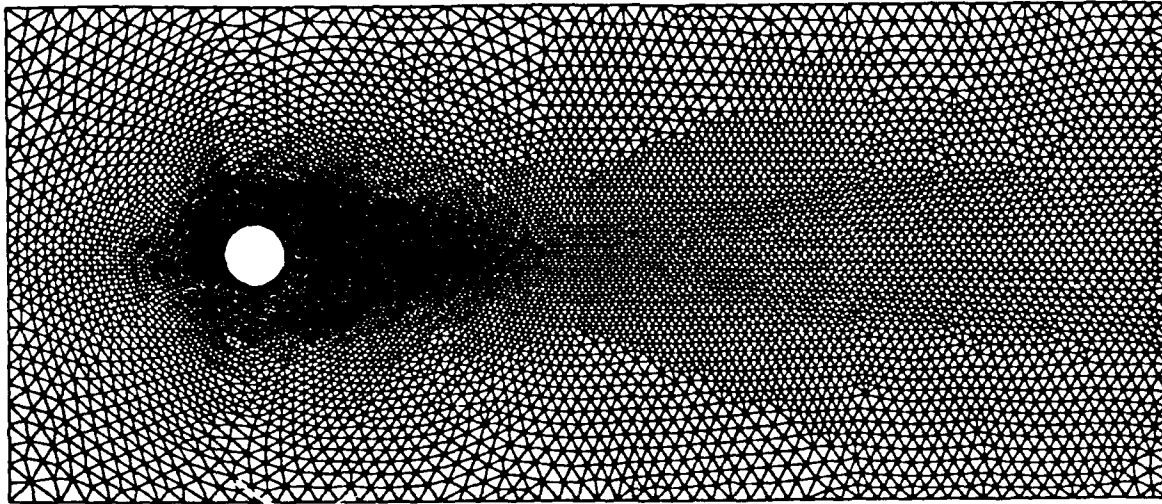
S = Speed-up

No. of Proc.	Fine Grid		Coarse Grid	
	S	S^*	S	S^*
1	1.000	1.000	1.000	1.000
2	1.877	1.885	1.863	1.872
4	3.537	3.636	3.323	3.388
8	6.240	6.858	5.311	5.676
16	9.200	12.699	6.201	8.019

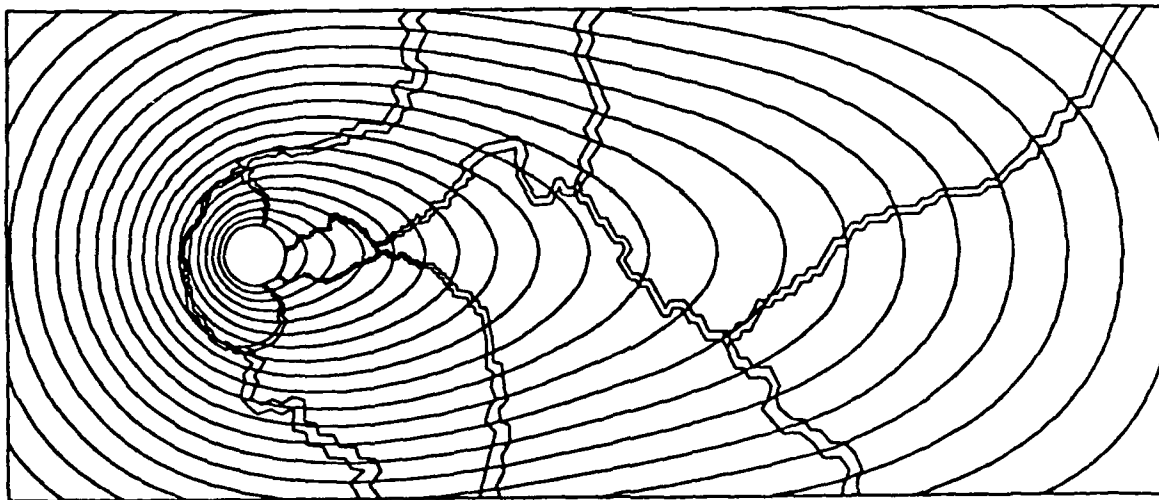
τ = cpu time/point/timestep (μ Sec.)

* denotes timings without I/O

Table 1. Performance of FEHEAT on iPSC 860



a. Grid, npoin= 9,767, nelem= 29,236, ndomn= 8



b. Temperature Distribution, min= 0.0, max= 334.0, $\Delta T = 16.7$

Fig. 2. Model Problem of Heat Conduction around a Cylinder

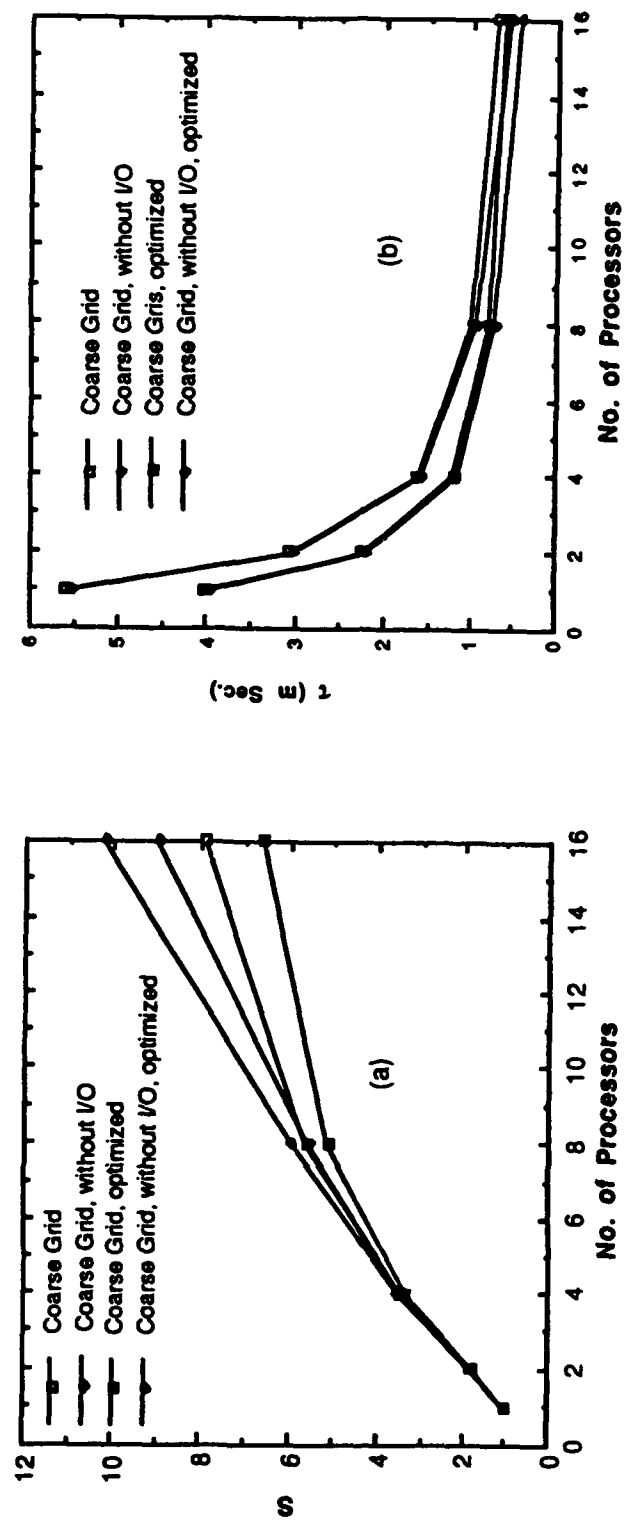


Fig. 3. Performance of the Explicit Flow Solver

No. of Proc.	Without Optimization		Optimized	
	τ_{avg}	τ_{avg}^*	τ_{avg}	τ_{avg}^*
1	5.584	5.527	4.019	3.955
2	3.055	3.015	2.230	2.182
4	1.614	1.576	1.216	1.149
8	0.998	0.934	0.784	0.713
16	0.707	0.543	0.608	0.440

S = Speed-up

No. of Proc.	Without Optimization		Optimized	
	S	S^*	S	S^*
1	1.000	1.000	1.000	1.000
2	1.828	1.833	1.802	1.813
4	3.459	3.507	3.305	3.441
8	5.594	5.917	5.123	5.550
16	7.903	10.180	6.606	8.980

τ = cpu time/point/timestep (mSec.)

* denotes timings without I/O

Table 2. Performance of FEIC25 on iPSC 860

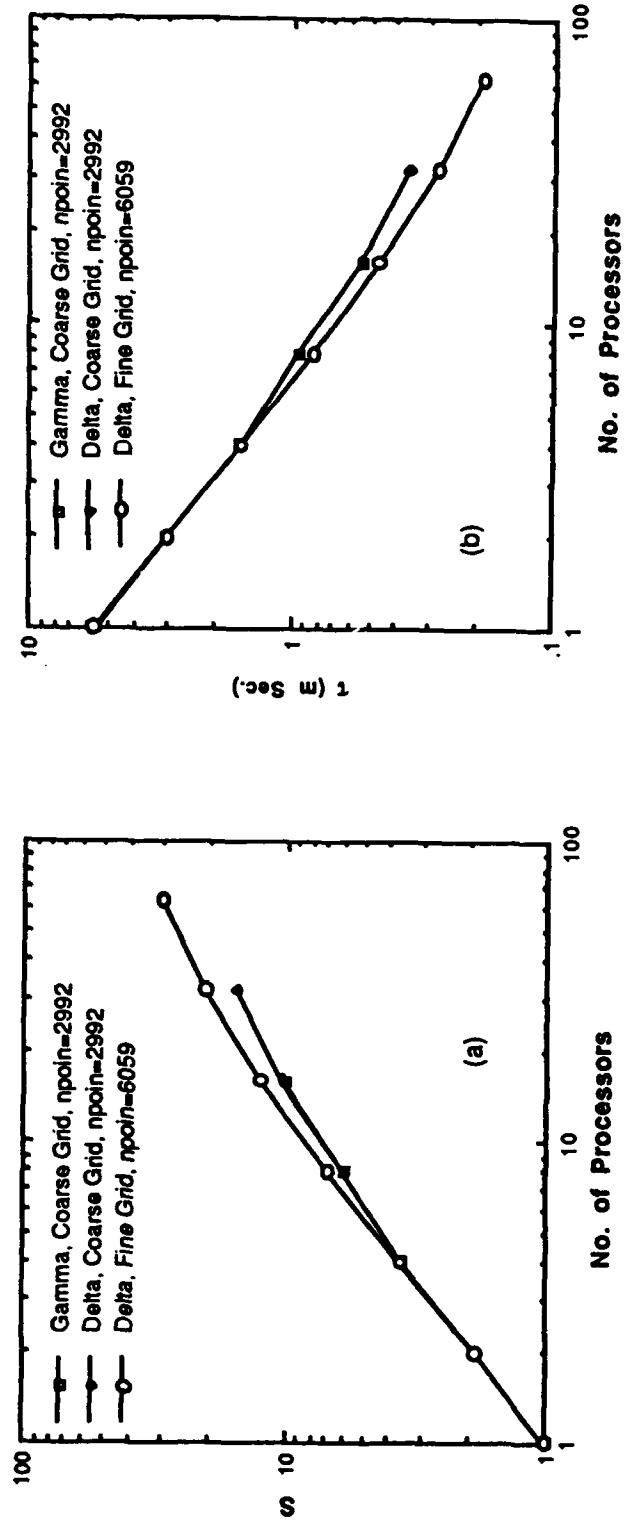


Fig. 4. Performance of the Explicit Flow Solver on Gamma and Delta

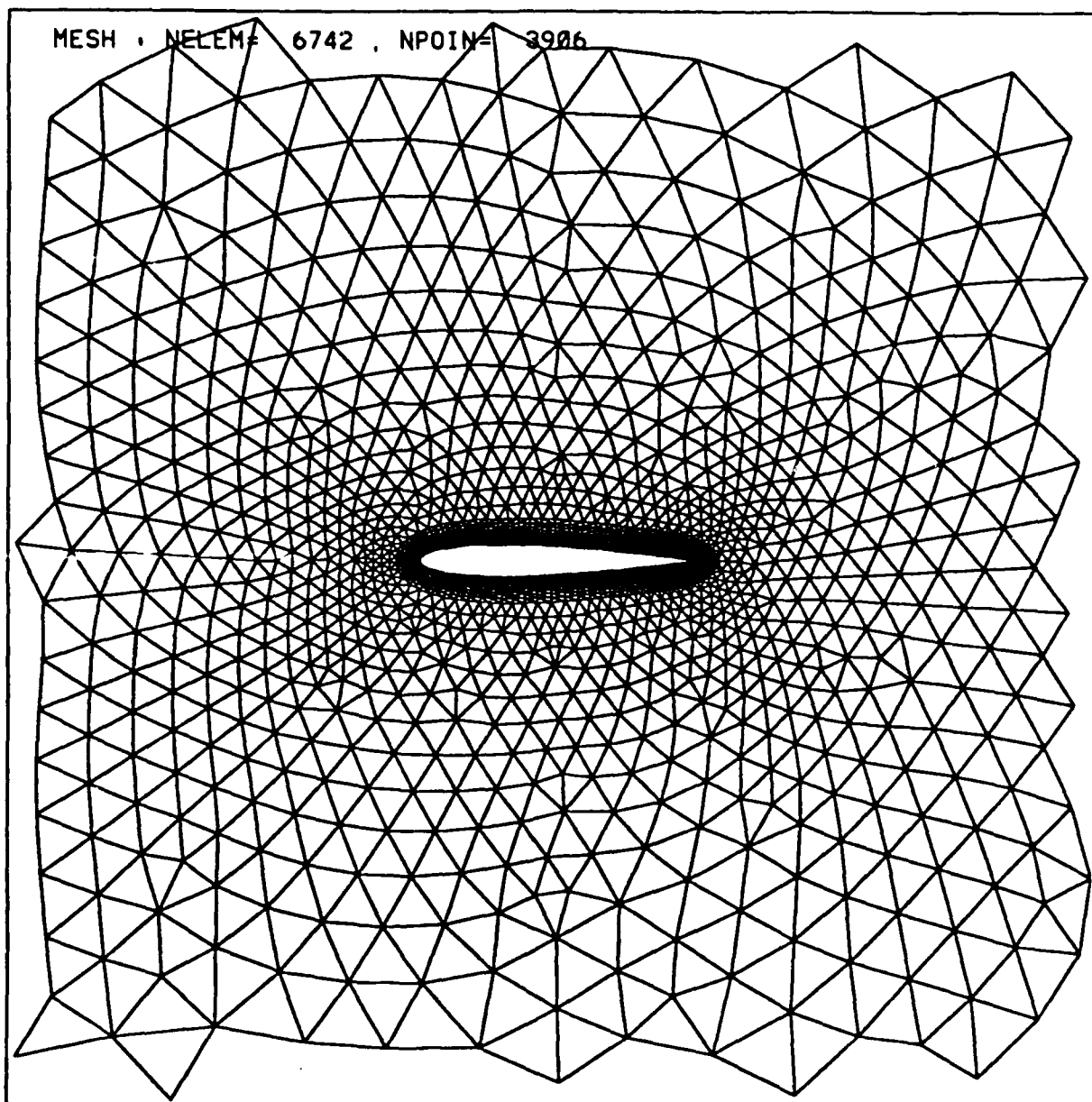


Fig. 5. Flow Past a NACA0012 Airfoil, $\alpha = 0^\circ$

a. Grid, npoin = 2992, nelem = 5860, ndomn = 16

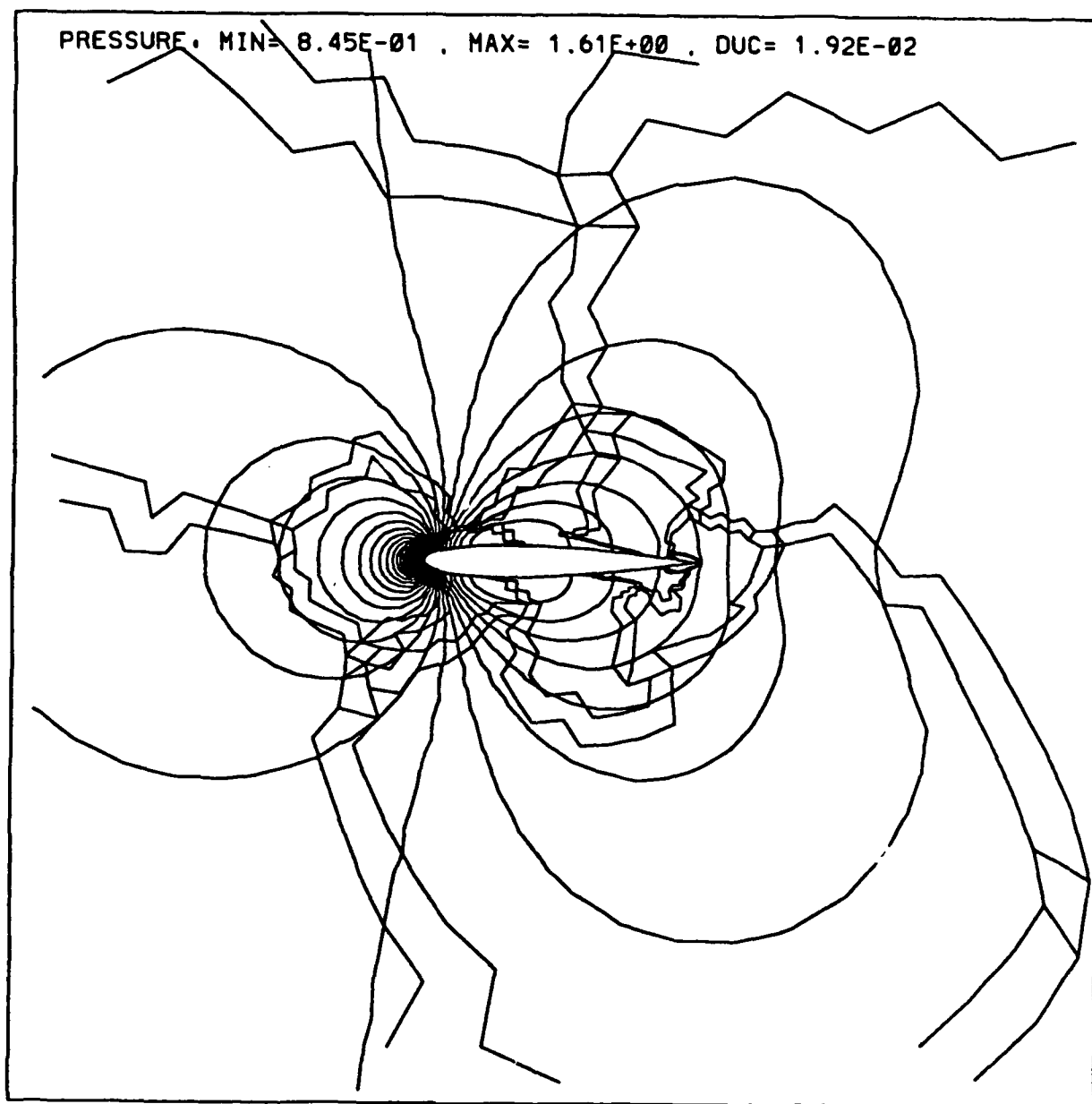


Fig. 5. Flow Past a NACA0012 Airfoil, $\alpha = 0^\circ$

b. Pressure, min.= 0.845, max.= 1.61, $\Delta p = 1.92E-02$

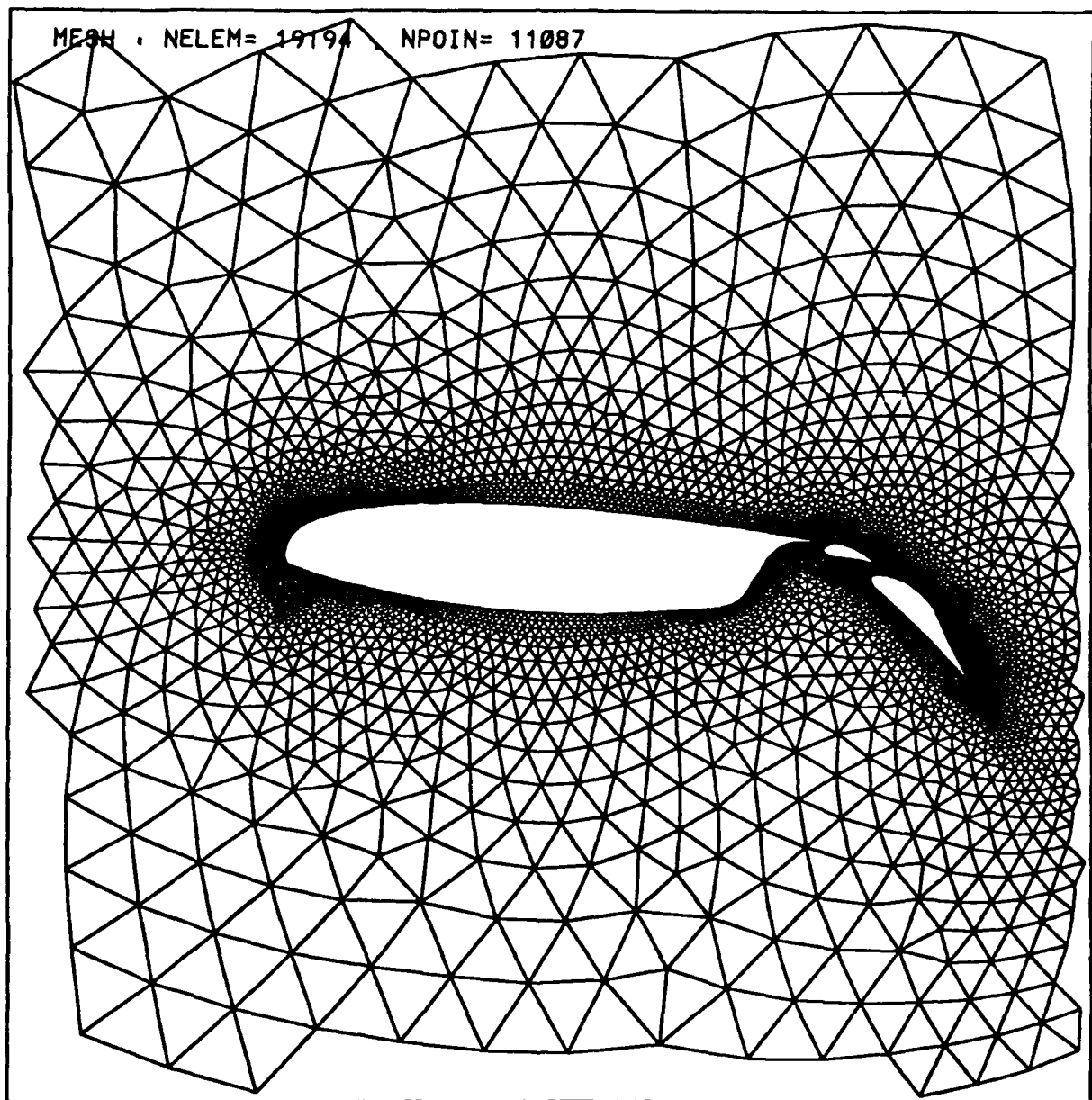


Fig. 6a. Unstructured Grid Around a Tri-Element Airfoil

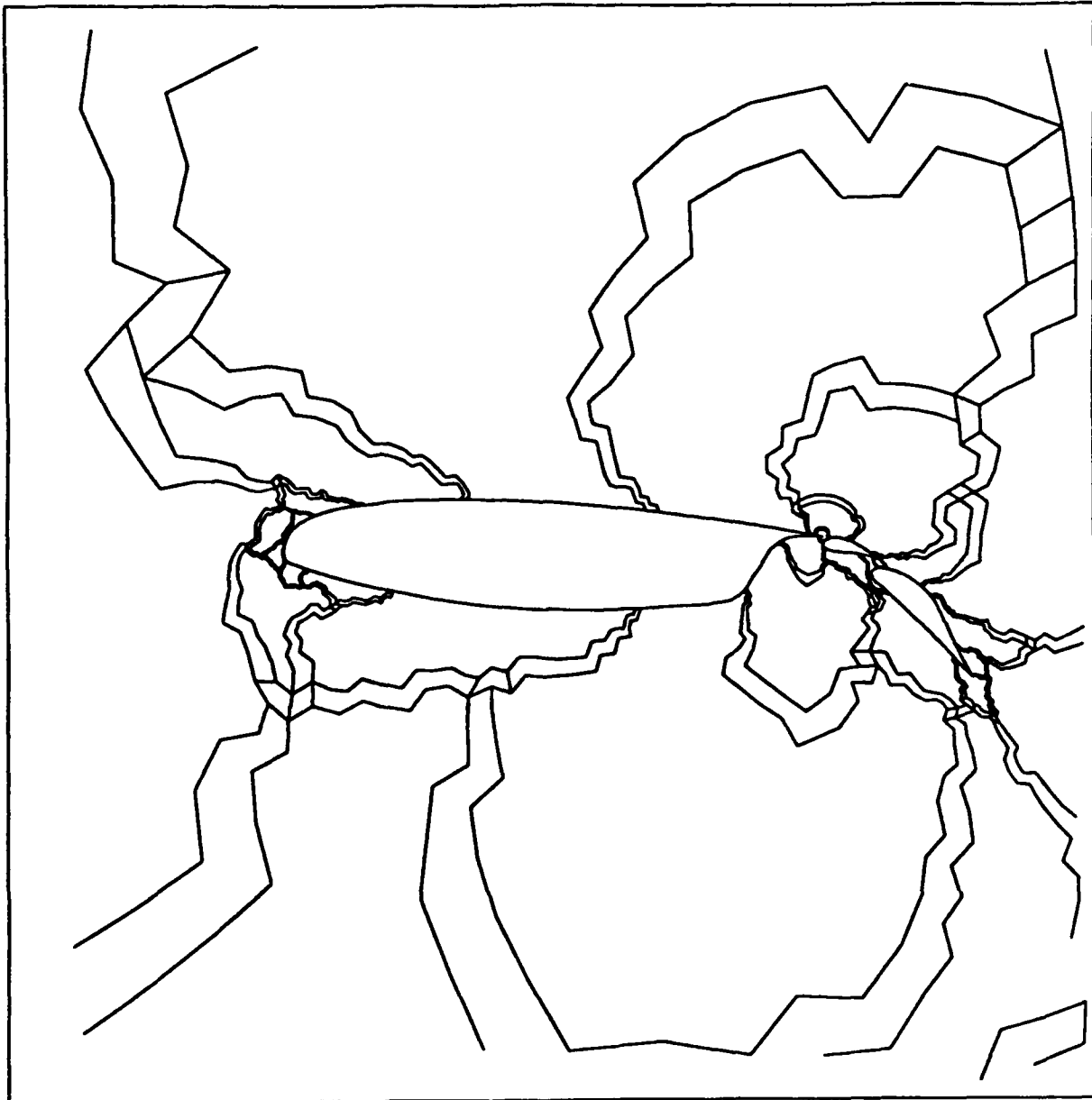


Fig. 6b. Domain Decomposition of the Grid around a Tri-Element Airfoil

ndomn = 32

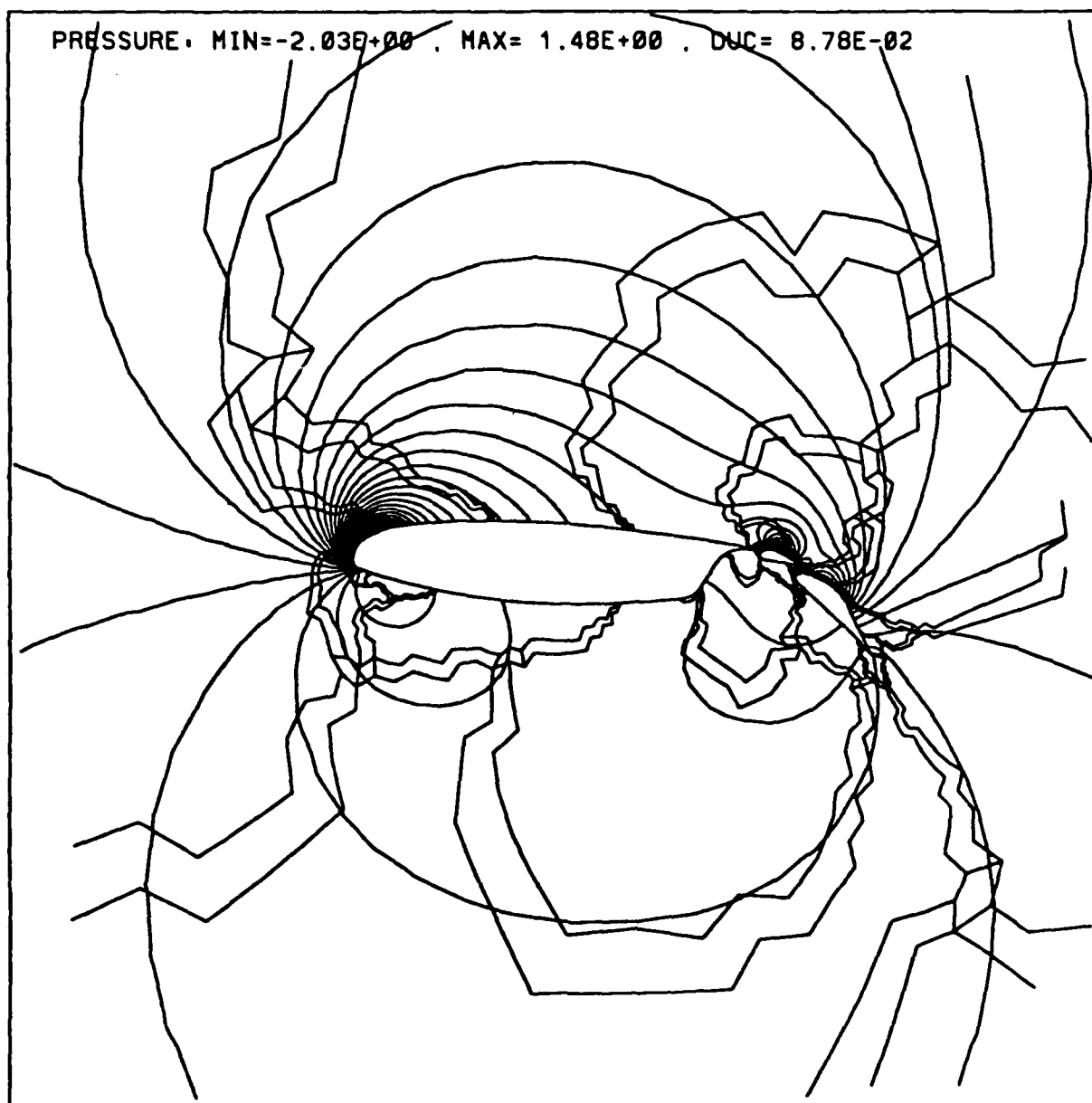


Fig. 7. Flow Past a Tri-Element Airfoil, $\alpha = 5^\circ$

a. Pressure, min.= -2.03, max.= 1.48, $\Delta p = 8.78E-02$

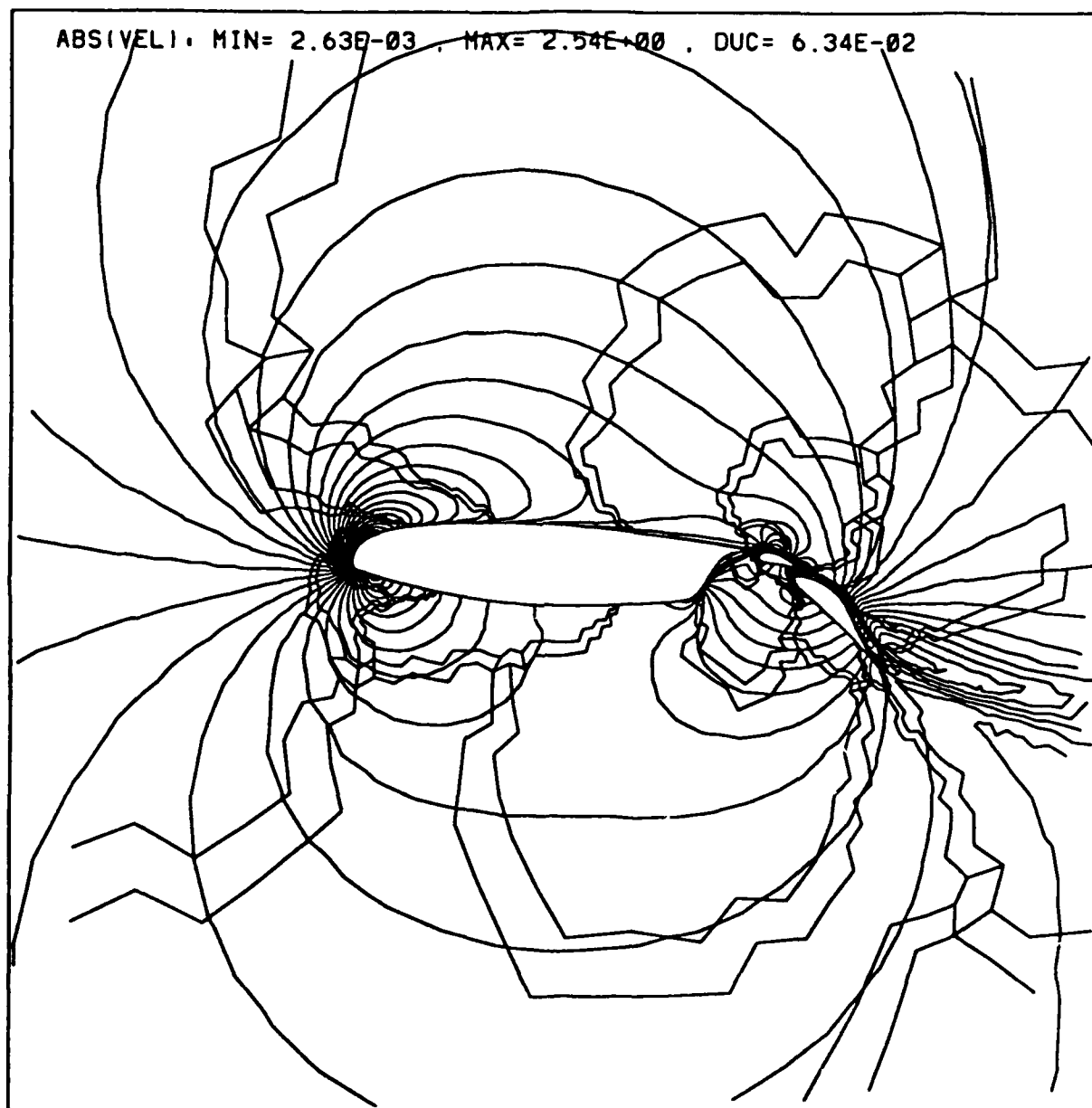


Fig. 7. Flow Past a Tri-Element Airfoil, $\alpha = 5^\circ$

b. Absolute Velocity, min.= 2.63E-03, max.= 2.54, $\Delta V = 6.34E-02$

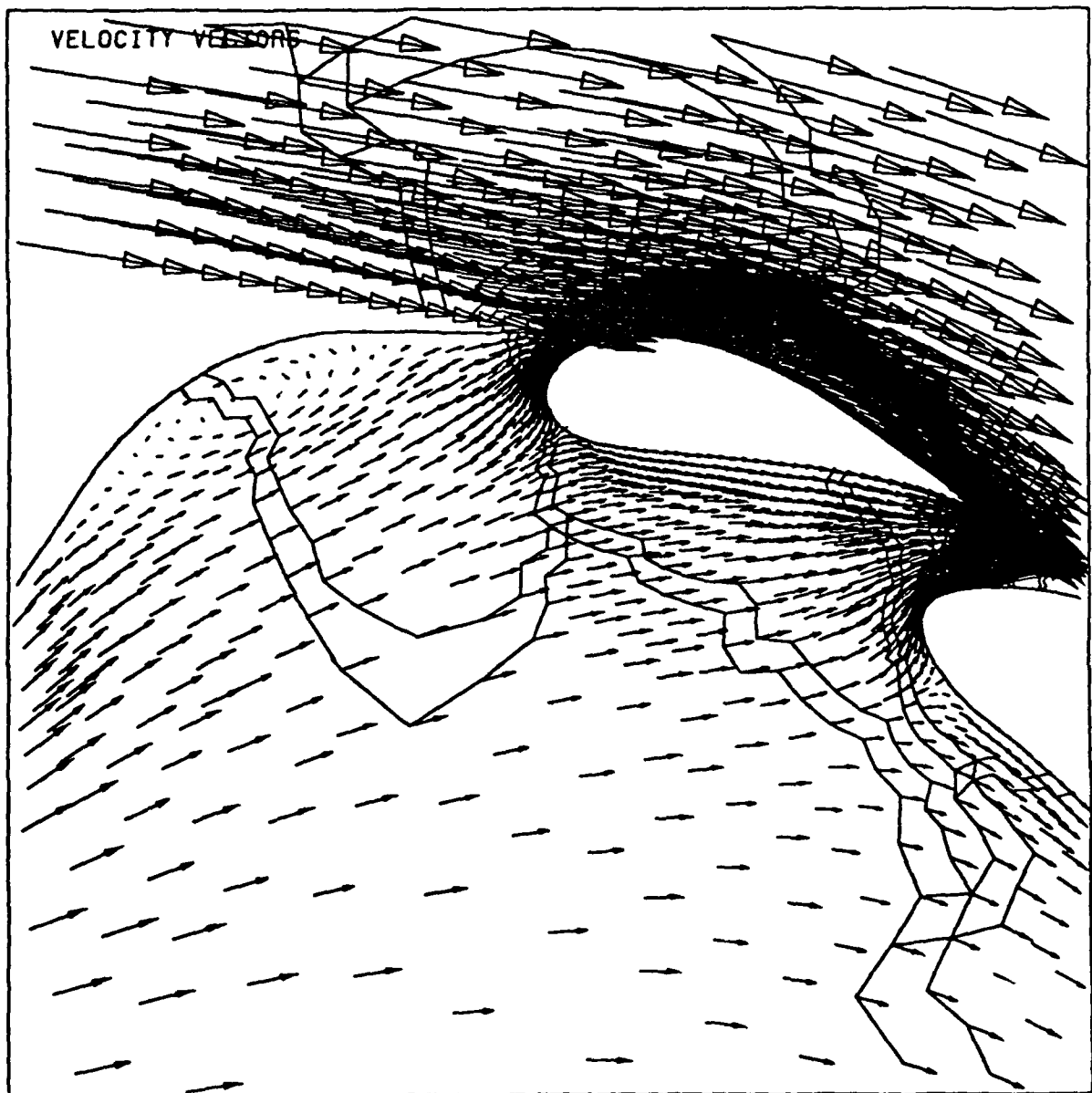


Fig. 7. Flow Past a Tri-Element Airfoil, $\alpha = 5^\circ$

c. Velocity Vectors

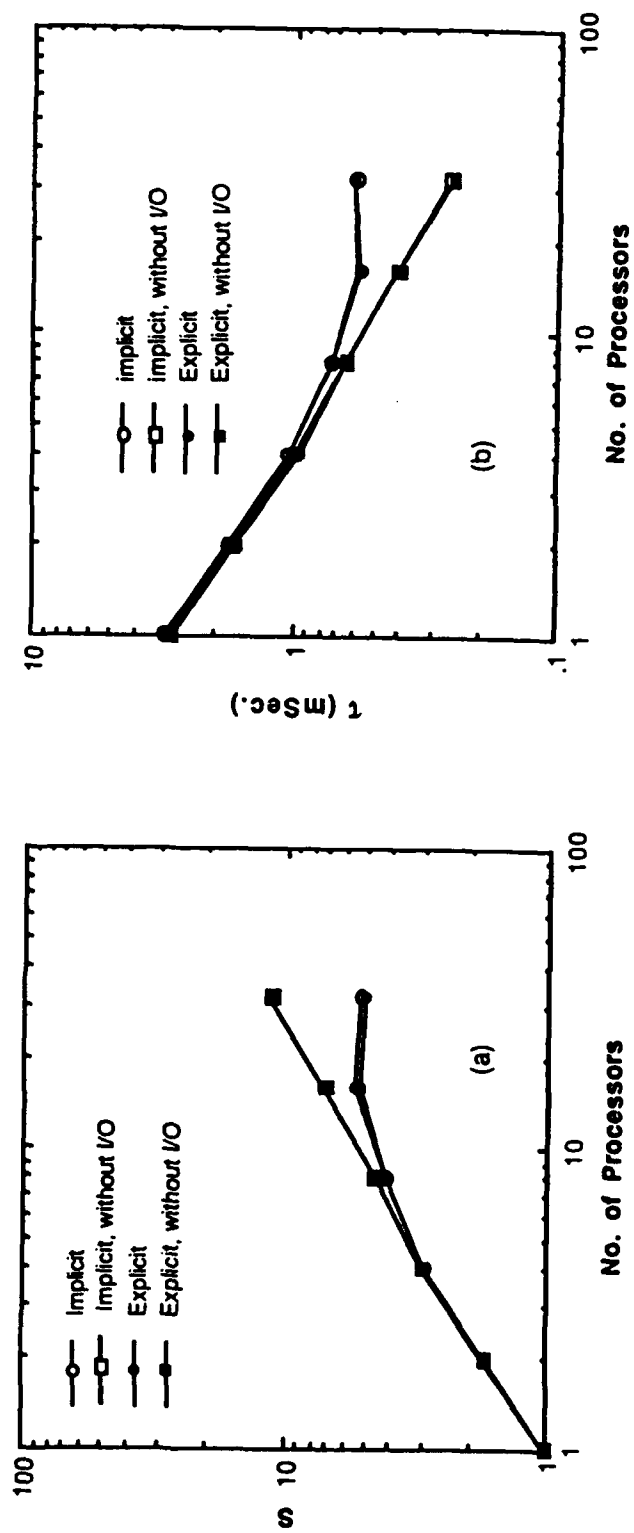


Fig. 8. Performance of the Implicit Flow Solver

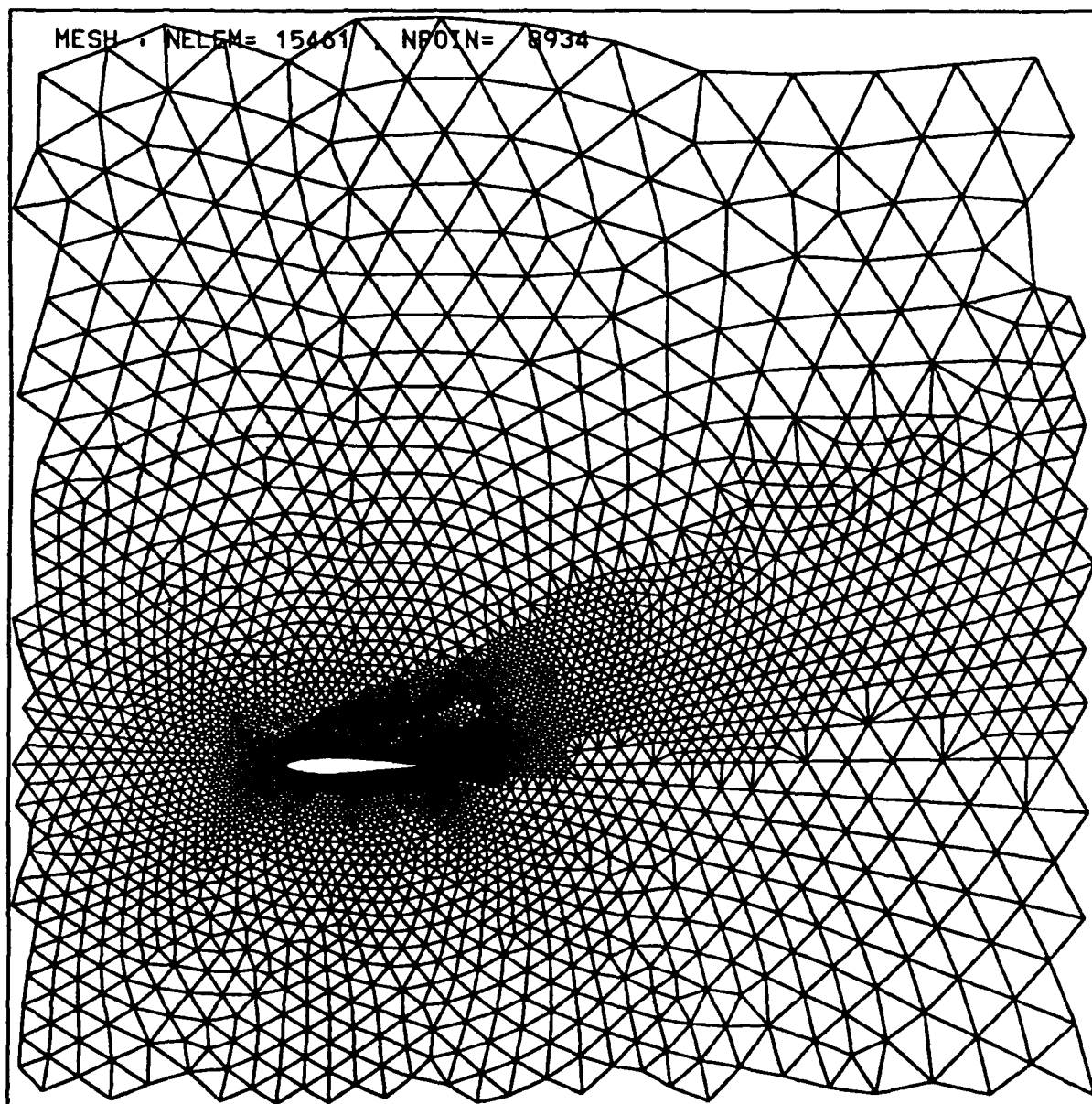


Fig. 9. Flow Past a NACA0012 Airfoil, $\alpha = 20^\circ$

a. Grid, npoin = 6,785, nelem = 13,376



Fig. 9. Flow Past a NACA0012 Airfoil, $\alpha = 20^\circ$

b. Domain Decomposition, ndomn = 32

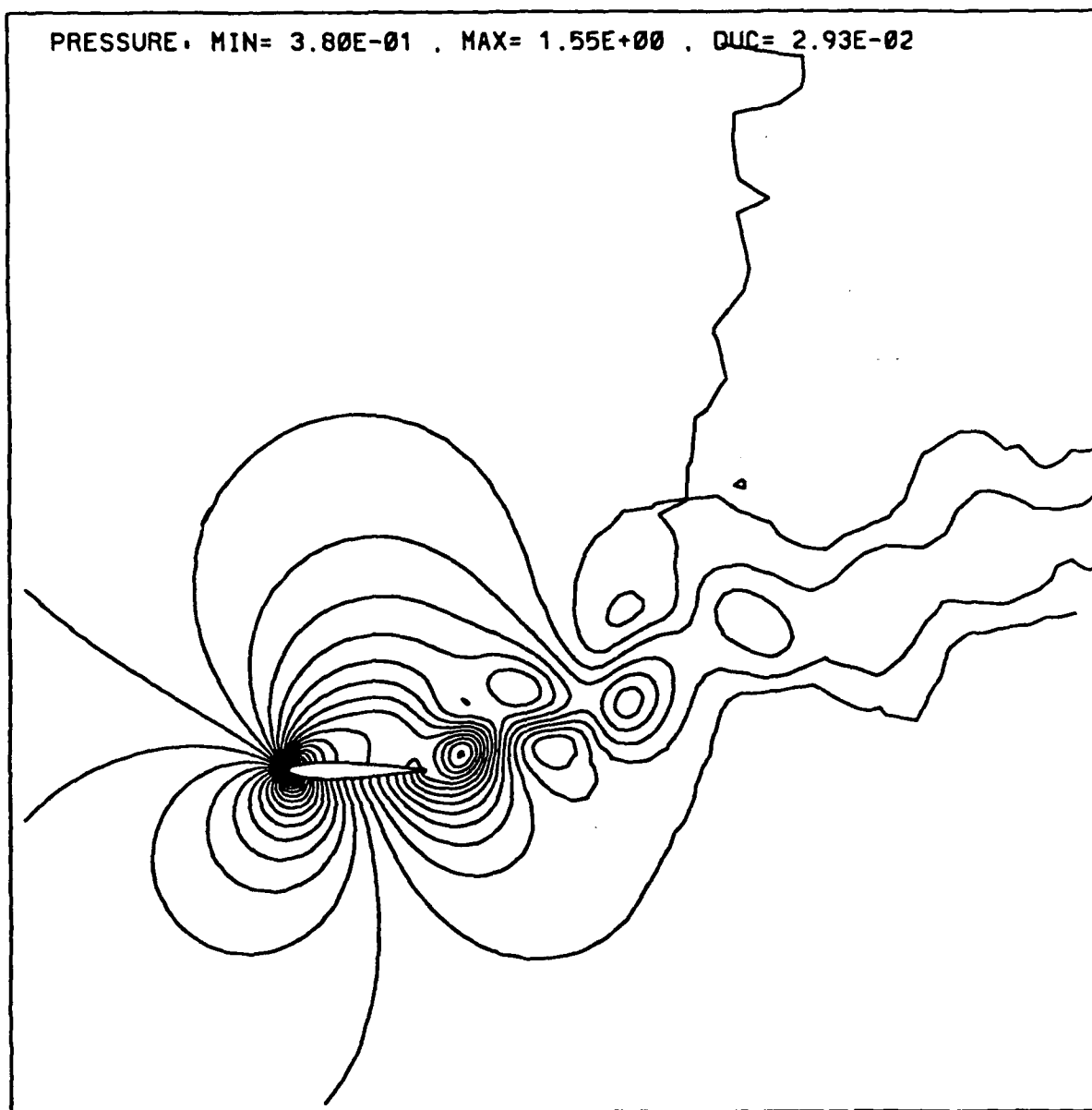


Fig. 9. Flow Past a NACA0012 Airfoil, $\alpha = 20^\circ$

c. Pressure, min.= 0.38, max.= 1.55, $\Delta p = 2.93E-02$

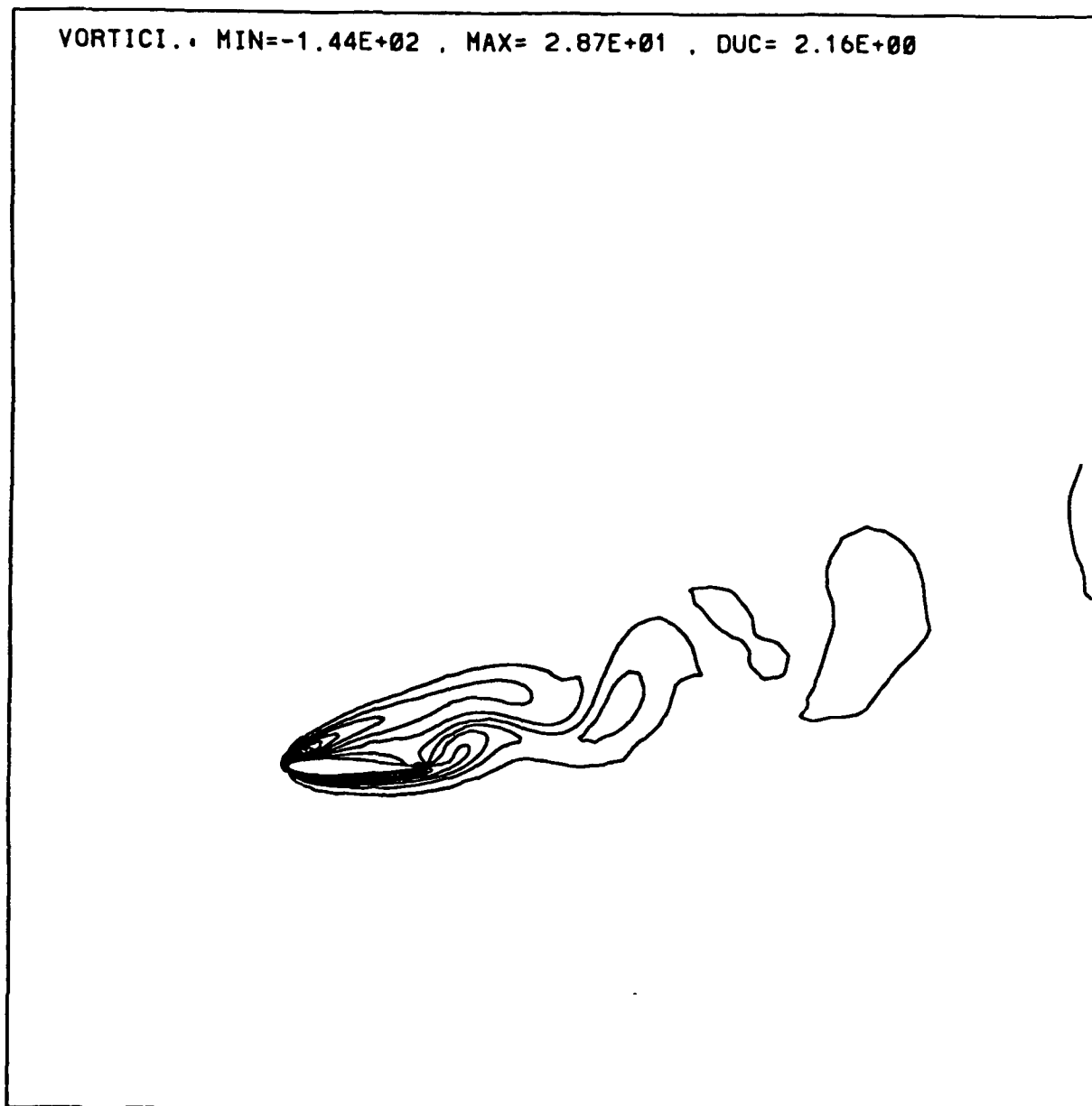


Fig. 9. Flow Past a NACA0012 Airfoil, $\alpha = 20^\circ$

d. Vorticity, min.= -1.44E+02, max.= 2.87E+01, $\Delta\omega = 2.16$

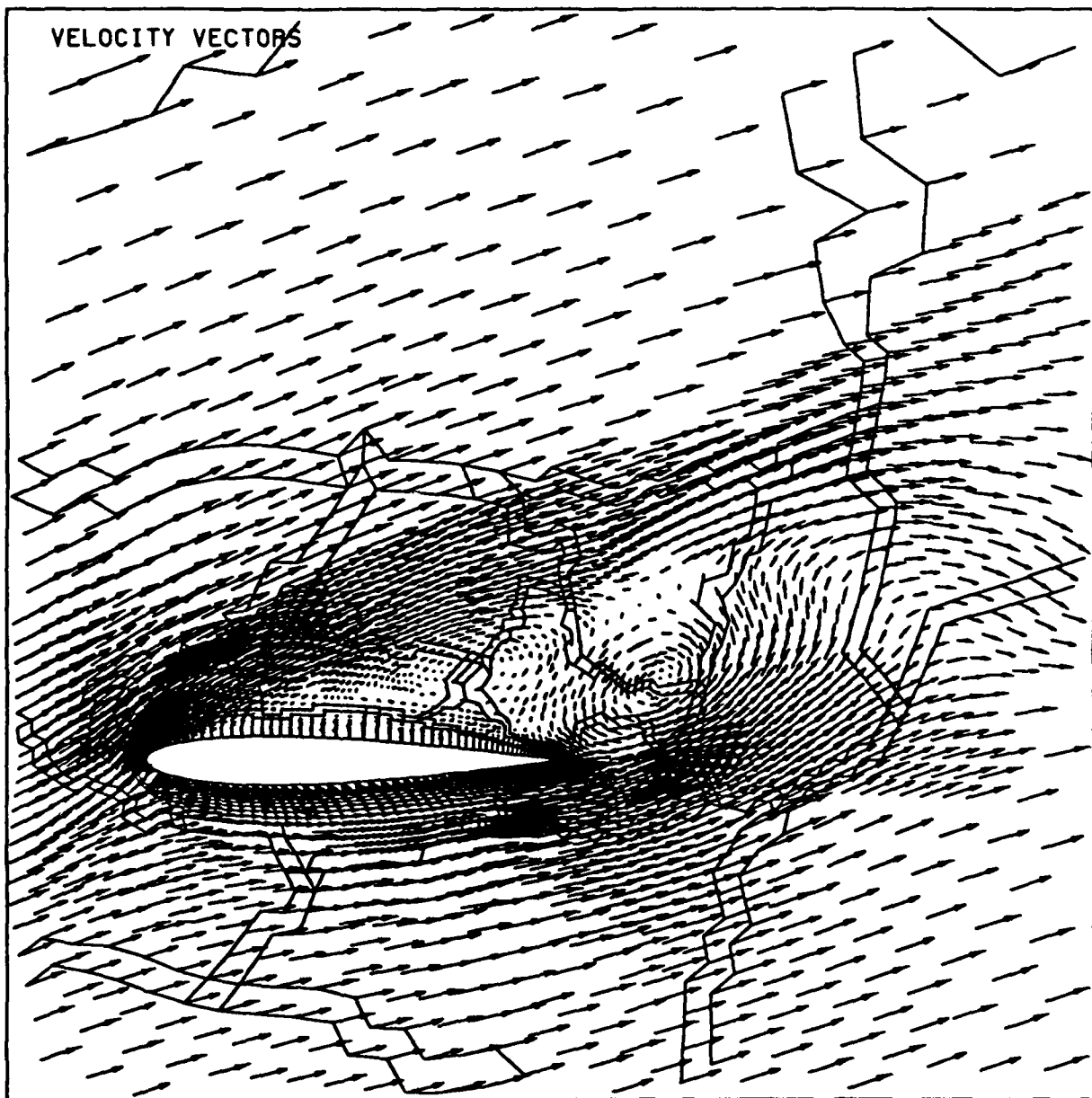


Fig. 9. Flow Past a NACA0012 Airfoil, $\alpha = 20^\circ$

e. Velocity Vectors