NASA TECHNICAL MEMORANDUM

NASA TM X- 73904

COPY NO.

NASA TM X- 73904

SOLUTION OF A LARGE HYDRODYNAMIC PROBLEM USING THE STAR 100 COMPUTER

K. James Weilmuenster
Lona M. Howser

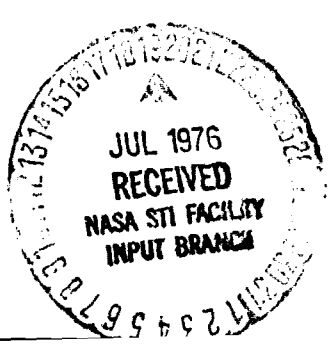
May 1976

N76-26429

(NASA-TM-X-73904) SOLUTION OF A LARGE HYDRODYNAMIC PROBLEM USING THE STAR-100 COMPUTER (NASA) 30 p HC $4.00 CSCL 20D

Unclas
G3/34 44513

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
LANGLEY RESEARCH CENTER, HAMPTON, VIRGINIA 23665



1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM X- 73904		

4. Title and Subtitle	5. Report Date
Solution of a Large Hydrodynamic Problem Using the STAR 100 Computer	May 1976
	6. Performing Organization Code: 6430

7. Author(s)	8. Performing Organization Report No.
K. James Weilmuenster Lona M. Howser	

9. Performing Organization Name and Address	10. Work Unit No.
NASA Langley Research Center Hampton, VA 23665	506-26-10-01
	11. Contract or Grant No.

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics & Space Administration Washington, DC 20546	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The advent of vector processing computer systems has brightened the prospects for investigating large hydrodynamics problems which are beyond the computational scope of current computers. Problem formulation, particularly in the area of data organization, is crucial to the effective use of vector processing machines. A representative hydrodynamics problem, the shock initiated flow over a flat plate, was used for exploring data organizations and program structures needed to exploit the STAR-100 vector processing computer. A brief description of the problem is followed by a discussion of how each portion of the computational process was vectorized. Finally, timings of different portions of the program are compared with equivalent operations on serial machines. The speed up of the STAR-100 over the CDC 6600 program is shown to increase as the problem size increases. All computations were carried out on a CDC 6600 and a CDC STAR 100, with code written in FORTRAN for the 6600 and in STAR FORTRAN for the STAR 100.

17. Key Words (Suggested by Author(s)) (STAR category underlined)	18. Distribution Statement
Fluid Mechanics Numerical Analysis Computer Programing and Software	Unlimited.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	28	$3.75

*Available from { The National Technical Information Service, Springfield, Virginia 22151
STIF/NASA Scientific and Technical Information Facility, P.O. Box 33, College Park, MD 20740

INTRODUCTION

The advent of vector processing computer systems has brightened the prospects for investigating large hydrodynamics problems which are beyond the computational scope of current computers. Problem formulation, particularly in the area of data organization, is crucial to the effective use of vector processing machines. A representative hydrodynamics problem, the shock initiated flow over a flat plate, was used for exploring data organizations and program structures needed to exploit the STAR-100 vector processing computer. A brief description of the problem is followed by a discussion of how each portion of the computational process was vectorized. Finally, timings of different portions of the program are compared with equivalent operations on serial machines. The speed up of the STAR-100 over the CDC 6600 program is shown to increase as the problem size increases. All computations were carried out on a CDC 6600 and a CDC STAR 100, with code written in FORTRAN for the 6600 and in STAR FORTRAN for the STAR 100.

LIST OF SYMBOLS

a	dummy variable
k	thermal conductivity
M	number of grid points in the y direction
N	number of grid points in the x direction
P	static pressure
t	time
T	Temperature
u	x component of velocity
v	y component of velocity
x	distance parallel to the freestream
y	distance normal to the freestream
β	compression parameter in equation (1)
Δx	spatial increment in x
$\Delta\eta$	spatial increment in η
η	non-dimensional distance normal to the freestream
μ	viscosity
ρ	density
ψ	dummy variable
χ_{max}	maximum length parallel to the freestream
η_{max}	maximum length normal to the freestream in the transformed coordinate

Problem Description

The establishment of a test gas flow over a body in an impulse facility such as a shock tube, a shock tunnel or an expansion tube is the result of the asymptotic relaxation of an initially unsteady flow to a steady state.

Several analytical papers have been written on the establishment of a shock-induced laminar boundary-layer on a flat plate (refs. 1, 2, 3). Analyses in these publications were based on large freestream unit Reynolds numbers which allowed the effects of a boundary-layer induced leading-edge shock and its interaction with the boundary layer to be neglected. However, when the freestream unit Reynolds number becomes sufficiently small, these effects can have an appreciable impact on the development of the shock-induced, as well as the steady-state, flow over a flat plate. A study of these leading edge effects led to the flow field code described in this paper.

The effects of shock boundary-layer interaction on the flow over a flat plate have been extensively investigated. The remarks found in ref. 4 provide a good, basic discussion of the subject. Briefly, if the freestream unit Reynolds number is sufficiently small, the boundary-layer growth at the leading edge of the plate will be great enough to make the leading edge appear to the flow as a blunted body thus creating a leading-edge shock whose strength decreases with distance from the plate and eventually degenerates to a Mach wave. At steady state, the flow along the plate can be divided into two regions: (1) a merged region near the leading edge where there is no separation between the shock and boundary-layer and (2) a weak interaction region in which the shock and boundary layer are distinct.

The dominant features of the unsteady flow field are shown on figure 1. Here, a normal shock wave is moving from left to right across the plate into a gas at rest while the leading edge shock is beginning to form due to the initial boundary layer growth. That portion of the flow near the leading edge will be approaching a steady state while the rest of the flow field is unsteady.

Method of Solution

The unsteady flow field for this problem can be generated using a finite difference representation of the unsteady, compressible, two dimensional Navier-Stokes equations.

A two-step Lax-Wendroff finite difference scheme as outlined in ref. 5 has been used as the algorithm for solving the Navier-Stokes equations. Two modifications have been made to the method of ref. 5. First, the transport properties - viscosity and thermal conductivity - have been made a function of the local temperature. Secondly, a coordinate transformation has been made in the y direction. The transformation, in the form

$$\eta = \ln (\beta y + 1) \qquad (1)$$

allows a larger nodal packing at the wall. Also, for unequal y increments, the differencing can be done in equal increments of η which greatly simplifies the algorithm. Details of the equations and differencing algorithm may be found in appendix A.

The computations have been carried out over a rectangular grid system which represents a sharp leading-edge flat plate 15.24 cm in length and a vertical height above the plate of 4.67 cm. The computational grid is illustrated in figure 2. Nodal points upstream of the plate are initialized with post-shock conditions and those on the plate are initialized with the unshocked conditions. Symmetry conditions are imposed along the line $y = 0$ for nodes in front of the plate throughout the computations.

The upper boundary was considered an out flow boundary with values along it being determined by extrapolation from the interior points. The right hand boundary was moved downstream as the normal shock moved to the right until a maximum grid size of 200 x 28 was reached. Along the plate,

the no-slip conditions $u = 0$ and $v = 0$ and the adiabatic wall condition $\frac{\partial T}{\partial y} = 0$ were imposed while the wall density was found from a second order extrapolation from the interior points.

Code Vectorization

System Characteristics:

The STAR-100 is a large scale, high speed, logical and arthmetic computer utilizing many advanced design features such as vector processing and virtual addressing. It also contains vector arithmetic and functional units designed for pipeline operations on a vector, where a vector is defined as a set of contiguous elements. The vector instruction performs operations on ordered scalars which are elements of the vector. The vector instructions read the scalars from consecutive storage locations over a specified address range called a field, perform the designated operation on each set of operands, and stores the results in consecutive addresses of a resultant field beginning at a specified starting address, i.e., one vector instruction operates on two vectors and stores a vector result. The starting address and vector length are contained in one 64 bit word which describes the vector and is called a vector descriptor.

The STAR instruction set includes, in addition to vector add, subtract, multiply and divide, other vector instructions that are useful when vectorizing this and similar problems. They will be described at appropriate points in the text. The equation used to obtain timings of STAR vector instructions is of the form

$$T = s + \alpha \ell$$

where T is the time in clocks (one clock equals 40 nanoseconds), s is the start up time which is different for each vector instruction, α is a constant which depends on the particular instruction and word length of the

operand and ℓ is the length of the vector. For example, timing for the vector add instruction using 64 bit operands is

$$T = (96 + \ell/2) \text{ clocks.}$$

For a vector computation, it is more efficient to use one long vector than to repeat the computation several times using shorter length vectors. With the shorter vector length the vector startup time is multiplied by the number of times the vector instruction is repeated.

Storage Considerations:

As with serial computer systems, the core storage requirements on a vector computer must be taken into account. Even with virtual memory it is necessary to be aware of the storage needed for solving the problem. More core storage is needed to effectively use the vector version of the problem than is needed if it were run on a serial computer. When a FORTRAN vector expression of more than two terms is evaluated, one or more temporary vectors is used to store the intermediate result. In addition, some terms need to be repeated to form a long vector of the same length as the other computation to avoid doing many repeated computations with short vectors. As an example, the expression $\sum_{i=1}^{i=10} a_i b_{i,1} + \sum_{i=1}^{i=10} a_i b_{i,2} + \ldots$ should be written as

$$\sum_{j=1}^{j=J} \sum_{i=1}^{i=10} a_{i,j} b_{i,j} \quad \text{where } a_{1,1} = a_1, a_{1,2} = a_2 \ldots a_{2,1} = a_{2,2} = a_2 \ldots$$

etc. The forming of the longer vector can be done by repeated vector transmits where the short vector is moved into the longer vector at appropriate locations.

Important factors in vectorizing the code are recognizing where vector arithmetic exists and creating temporary vectors where possible. These temporary vectors consist of vector expressions which are common to several equations. The temporary vectors are computed only once, then used in

subsequent calculations. These temporary vectors cause more storage to be needed, but avoid repeating vector calculations. Equivalencing of some temporary storage areas is used in the code to minimize the total storage needed.

The present discussion is limited to a grid size where all the variables necessary for computation will fit in central memory, i.e. the total storage required must be less than 500 K words. In this problem it is desirable to treat all the points in the whole mesh (grid) as one vector. This gives vectors of length $\ell = M \times N$ to use in the vector computation. These are the maximum length vectors that can be obtained and will be more efficient on a vector computer than using vector length of M or N.

Initially, the manner in which the data is to be stored internally in the computer must be determined. The option of storing by what will be defined as row or column storage is available. Referring to fig. 2 row storage means a vector is defined in the X direction. The first element of the vector is at the point $x = 0$, $\eta = 0$, the second element of the vector is at the point $x = \Delta x$, $\eta = 0$, etc. until $x = x_{max}$, then the next element in the vector is the point $x = 0$, $\eta = \Delta\eta$ and continues in this manner for the entire grid. Column storage means a vector is defined in the η direction. The first element of the vector is at the point $x = 0$, $\eta = 0$. The second element of the vector is at $x = 0$, $\eta = \Delta\eta$ etc. until $\eta = \eta_{max}$, then the next element in the vector is the point $x = \Delta x$, $\eta = 0$ and continues in this manner for the entire grid.

When choosing the direction of storing the vector, the following were considered:

1) how does the storage affect the boundary computation;
2) what length vectors can be used;
3) how does the storage affect increasing the number of grid points used during the computation as the right hand boundary moves downstream.

Row storage means the boundary along $\eta=\eta_{max}$ and along $\eta=0$ are vectors of length N and the boundaries could be computed using vector computations. The boundary at $\eta=\eta_{max}$ is a straightforward extrapolation while the boundary computations at $\eta=0$ consist of lengthy equations involving many computations. The boundary along x=0 is held constant, and the values associated with it would be destroyed after each time step if vectors were of length N x M. Since the elements along x=0 are not stored in contiguous locations, redefining the elements would require special considerations. Increasing the number of grid points at which calculations are made would not be reasonable for row storage if the entire grid was treated as one vector. This would be increasing the length of the vector by inserting elements within the vector. If vectors of length N-1 were used for all the computations the grid could be increased in the x direction by simply increasing the length of the vector and the additional points would be the elements at the end of the vector. Also the boundary at x=0 would not be disturbed. Therefore, for a reasonable vector formulation using row storage, the vector computations would use vectors of length N-1 and would be repeated M-2 times (calculations at the upper and lower boundary are considered separately).

Column storage means the boundary along x=0 is a vector. These values remain constant, so the vector computation can actually start with the second column $x = \Delta x$. The vectors run in the column direction so the boundary conditions along x=0 are never destroyed. The elements are not stored in contiguous locations along the boundaries $\eta = 0$ and $\eta=\eta_{max}$, so they will require special consideration. The entire grid can be treated as a vector of length M x N and when the number of points in the grid is increased it means only increasing the vector length by M elements. This essentially added one column to the grid.

The program was coded using column storage. The deciding factor here in using the column storage as defined is not the boundary computation, which would probably be more advantageous using row storage, but the value of using the longer vectors for the interior computation and the ease of adding to the grid size.

Interior Point Computations - The basic finite difference equations used in the problem formulation vectorize readily so all computations for the interior points can use vector instructions. The difference equations are of the forms

$$\psi(t + \Delta t, x) = \psi(t,x) - \left(\frac{\Delta t}{2\Delta x}\right) \cdot (\psi(t, x + \Delta x) - \psi(t, x - \Delta x))$$

which consists of a vector subtract, a multiply of a constant, $(\Delta t/2\Delta x)$, and a vector, and then a second vector subtract.

The computation of the interior points begins with the element at the point $\eta = \Delta\eta$, $x = \Delta x$, using the storage defined as column storage and a vector length of $\ell = M * (N-1) - 2$. Using this vector length, all the points in the grid except the boundary along $x=0$, the point $\eta=0$, $x = \Delta x$ and the point $\eta=\eta_{max}$, $x = x_{max}$ are computed, see figure 2. This means that incorrect computations are made at the boundaries along $\eta = 0$ and $\eta=\eta_{max}$. In order that a vector calculation of length ℓ could be used, these calculations were made even though incorrect results were stored. The STAR has control vector capability which prohibits the storage of certain elements in a vector. This capability could have been used to prevent incorrect results from being stored along the boundaries; but in using this feature, the calculations are still performed even though the results are not stored. Since values for these boundary elements are not used before correct values are computed, it did not seem necessary to use the control vector feature. After the calculation of the interior points have been made, the values at the boundaries are computed and stored over the incorrect results.

Boundary Conditions:

With the method of storage defined as column storage used for the computations along $\eta = 0$ and $\eta=\eta_{max}$, it does not appear that vector computations could be used because the elements are not in contiguous locations. The boundaries require considerable computations and it would

be desirable to use vector computations. To compute the values along $\eta = 0$, the values at $\eta = \Delta\eta$ $\eta = 2\Delta\eta$, and $\eta = 3\Delta\eta$ are used. If the values along $\eta = \Delta\eta$, $\eta = 2\Delta\eta$, and $\eta = 3\Delta\eta$ were each represented by vectors, vector computations could be made which would result in a vector whose elements are the values of the boundary values along $x=0$.

In the evaluation of the boundary conditions extensive use is made of two instructions, transmit indexed list and transmit list indexed, each using one operand where the elements are not vectors. The transmit indexed list transmits noncontiguous information referenced by an index vector to a vector. This is essentially a gather technique, where information is gathered from noncontiguous locations to form a vector. The transmit list indexed instruction performs the reverse process, where the elements of a vector are transmitted to noncontiguous locations in memory indicated by an index vector. This is essentially a scatter to memory. These two instructions enable a user to form vectors if they do not exist, use vector arithmetic, then scatter the vector result to nonvector storage.

The vector instruction transmit indexed list is used to take elements along $\eta=\Delta\eta$ and form a vector. The same instruction is used to form vectors from the elements $\eta=2\Delta\eta$ and other vectors from the elements along $\eta=3\Delta\eta$. Once all the vectors needed in the computation are formed, they are used in expressions to compute a vector result of the boundary condition. This vector is then scattered back to the regular grid where the boundary along $\eta=0$ is not a vector. This is accomplished by the transmit list to indexed vector which scatters a vector to noncontiguous locations. Along the $\eta=\eta_{max}$ boundary a similar technique is used where the noncontiguous elements are made into a vector. These vectors are used in vector computations to compute a vector result which is then scattered back to the noncontiguous locations along the boundary.

Calculations Requiring Other Vector Instructions:

Conditions where two different paths may be followed after a comparison has been made on vector elements might be referred to as a vector IF statement. Such a situation is encountered when computations involving

real gas flows are made. Here, some parameters, such as transport properties, are prescribed by curve fits over different temperature ranges rather than an all inclusive analytic function. The vector IF statement does not exist in STAR FØRTRAN. However, the equivalent logic can be constructed from the vector instructions compare, compress and merge. The elements of a vector are first compared with the elements of another vector or a constant and a bit vector is the result. The elements of the bit vector are 1 or 0 depending upon whether the compare condition was true or false. This bit vector is then used with the original vector with two compress instructions to form two vectors. Separate computations are then performed, each using the appropriate vector. The bit vector can then be used to merge the two resultant vectors to form a single resultant vector of the original length.

Results

The shock induced flow over a flat plate is representative of a class of flow field problems for which no solutions have been obtained due to the lack of computational resources.

This program was chosen for coding because: 1) the solution requires integrating the full two-dimensional Navier-Stokes equations; 2) the solution allowed a variable computational field size to be used; and 3) the solution required the determination of realistic boundary conditions.

Three versions of the code were written: 1) a scalar version to be run on the CDC 6600 serial machine; 2) a fully vectorized version to be run on the STAR 100; and 3) a vector version which used scalar calculations to determine the boundary values.

The fully vectorized version of the code accommodated a maximum flow field size of 100 x 28 which allowed adequate resolution of the flow. This version of the code required a total of 44 dimensioned variables and a core storage size of 260 K as opposed to the serial version of the code which required only 25 dimensioned variables and a core storage of 65 K, but used disks for storage of most of the data.

On the STAR 100, two magnetic core storage page sizes are available for virtual address references. They are referred to as a large page size,

containing 65,536 64 bit words, and a small page size, containing 512 64 bit words. The selection of the page size to be used is an option under program control. Initially, small pages were used in the program. However, when using large pages there are fewer pages involved; therefore, fewer searches need to be made to relate a central memory address to a virtual memory address on a particular page. When the program was executed using large instead of small pages, an improved efficiency of about one-third was gained in the CPU time. Thus, large pages have been used in obtaining the following results.

A comparison of the running time for the scalar and fully vectorized versions of the code are summarized in table 1. Results for both the RUN and FTN compilers are given. Computations made with the FTN compiler utilized the highest optimization level. Since the computational field size is dynamic, the tabulated times shown represent the total CPU time required for the field to reach the given size. The computational speed advantage of the vector system over the serial system is obvious, with the increase in speed directly proportional to the size of the computation field, i.e. vector length.

A comparison of the time to do identical work for the vector and scalar versions of the boundary value computations is tabulated in table 2. A ten fold increase in the boundary vector length increases the required CPU time by 50% for the vector calculation whereas the identical scalar work requires approximately a ten fold increase in the CPU time.

Table 3 tabulates a breakdown of the total CPU time for the fully vector and scalar boundary calculation versions of the code. For the vector version of the code, the total CPU time spent doing the boundary calculations as a percentage of the total CPU time decreases dramatically as the number of grid points in the computational field becomes large when compared to the number of elements in the boundary vector. On the other hand, the total CPU time required to do the scalar boundary calculations remains at approximately 50% of the total CPU time.

As previously stated, the computation times quoted for the vector machine are for a program which resided entirely in control memory. To analyze the effect on computational speed of a program which required storage in excess of the central memory size, the grid size in the present code was increased so the program code and data would not fit in core. Thus, the virtual memory capability of the operating system was used to access, as needed, data or code which resided outside of central memory. The computational times were compared to the time required to do equivalent work using the in-core program. Equivalent CPU times were obtained, but the total wall clock time for the out-of-core run increased initially even though short vector computations were used. The increase was a factor of over 35 after only 200 time steps. Data were stored and used in such a random manner that the major portion of the computation time was used for getting the needed variables into core. The increased wall clock time proved this program to be impractical for obtaining solutions if the grid size is such that the code and data cannot fit in central memory.

The program has been modified in an attempt to minimize the use of wall clock time. The computational grid was broken up into blocks and the computation of the grid within each block was computed for a time step. The block size was determined and the storage was arranged so that all the variables necessary for the computation of the block would fit in memory. At the beginning of each block computation, the data for that block would have to be accessed initially, but during the computation of the block, no data outside of central memory would have to be accessed. Preliminary runs show that executing the modified program in this manner resulted in equivalent CPU times, but only a factor of five increase in the wall clock time required for identical work using the unmodified in-core program.

Figures 3 and 4 show some typical results generated by the code. Figure 3 is a plot of isotherms in the flow-field at a given instant of time. The leading edge of the plate is located at the left hand side of the figure. Here, the leading edge shock is located at the isotherm concentration at

the right hand side of the figure. The time dependency of the flow is illustrated by transverse velocity profiles for increasing times at a given x location.

CONCLUDING REMARKS

It has been shown that a flow field problem of such size as to be impractical for routine running on a serial processing machine can be coded and run in an efficient manner on a vector processing machine.

To extract this high performance from a vector processing machine such as the STAR-100, careful consideration must be given to the formulation and flow of the code being written to avoid such obvious inefficiencies as repeating identical vector calculations to the more subtle relationship between data storage and boundary calculations. Even when the vector lengths used are relatively short, the vectorized version of the code is clearly superior to serial code.

The vectorized code has demonstrated that branch operations encountered in imperfect gas calculations can be efficiently vectorized. It has also been shown that the calculation of boundary conditions require careful consideration, but can be done efficiently using the vector processor.

When the number of boundary points is small, there is no apparent advantage to vectorizing the boundary value calculations. However, when the number of boundary points becomes large, scalar calculations can require more than 200 percent of the time required to do the identical work (including time required to construct the vectors) using vector instructions.

It has been demonstrated that a complex, two-dimensional, fluid flow problem can be run quickly on a vector processing machine without exceeding the core storage requirements of the system. Larger two-dimensional and axisymmetric flows as well as three-dimensional flows will certainly require more mass data storage than can be handled by the system core size. Preliminary results have shown that modifications can be made to the current vectorized code to handle an out of core size problem and still maintain efficient use of a vector computer.

APPENDIX A

The Navier-Stokes equations for two-dimensional, unsteady, compressible flow can be written in the following form:

$$W_t = F_x + G_y + S_{1_x} + S_{2_y} \tag{1}$$

Here,

$$W = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \quad F = - \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E+P) \end{bmatrix}, \quad G = - \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v(E+P) \end{bmatrix},$$

$$S_1 = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ -q_x + u\tau_{xx} + v\tau_{xy} \end{bmatrix},$$

$$S_2 = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ -q_y + v\tau_{yy} + u\tau_{yx} \end{bmatrix}$$

where

$$E = P\,(e + 1/2\,(u^2 + v^2)),$$

$$\tau_{xx} = \mu\,(4/3\,u_x - 2/3\,v_y),\quad \tau_{yy} = \mu(4/3\,v_y - 2/3\,u_x)$$

$$\tau_{xy} = \tau_{yx} = \mu\,(u_y + v_x)$$

$$q_x = -\,kT_x,\quad q_y = -\,kT_y\ .$$

Introduction of the transformation

$$\eta = \ell n\,(\beta y + 1)$$

leads to

$$\frac{\partial}{\partial y} = \beta e^{-\eta}\ \frac{\partial}{\partial \eta}$$

and

$$\frac{\partial^2}{\partial y^2} = \beta^2 e^{-\eta}\ \frac{\partial}{\partial \eta}\ \left[e^{-\eta}\ \frac{\partial}{\partial \eta}\right]$$

so that equation (1) takes on the form

$$\omega_t = F_x + \beta e^{-\eta}\,G_\eta + S_{1_x} + \beta e^{-\eta}\ S_{2_\eta}\ . \qquad (2)$$

The stress terms are redefined as

$$\tau_{xx} = \mu\,(4/3\,u_x - 2/3\,\beta e^{-\eta}\,v_\eta)\ ,$$

$$\tau_{yy} = \mu\,(4/3\,\beta e^{-\eta}\,v_\eta - 2/3\,u_x)\ ,$$

$$\tau_{xy} = T_{yx} = \mu\,(\beta e^{-\eta} u_\eta + v_x)\ ,$$

$$q_x = -\,kT_x,\quad q_y = -\,k\beta e^{-\eta} T_\eta\ .$$

The equation (2) is differenced in the following manner:

$$W^{1/2}_{\pm 1/2,\,0} = 1/2\,(W^{0}_{\pm 1,0} + W^{0}_{0,0}) + \Delta t \left\{ \langle F_x \rangle^{0}_{\pm 1/2,0} + \beta e^{-\eta} \langle G_\eta \rangle^{0}_{\pm 1/2,0} + \langle S_{1_x} \rangle^{0}_{\pm 1/2,0} + \langle S_{2_y} \rangle^{0}_{\pm 1/2,0} \right\}$$

for the value of W at $t + \Delta t/2$, $x = \frac{\Delta x}{2}$ and $\eta = \eta$.
Permuting the indexes gives the value of W at $t + \Delta t/2$, $x = x$ and $\eta = \eta \pm \Delta\eta$.
For any function ψ the first order differences are given by

$$\langle \psi_x \rangle^{0}_{\pm 1/2,0} = \pm \frac{1}{\Delta x} \left[\psi^{0}_{\pm 1,0} - \psi^{0}_{0,0} \right]$$

$$\langle \psi_\eta \rangle^{0}_{\pm 1/2,\,0} = \frac{1}{4\Delta\eta} \left[\psi^{0}_{0,\,+1} + \psi^{0}_{\pm 1,\,+1} + \psi^{0}_{0,-1} + \psi^{0}_{\pm 1,\,-1} \right]$$

and similarly for $\langle \psi_x \rangle^{0}_{0,\,\pm 1/2}$ and $\langle \psi_\eta \rangle^{0}_{0,\,\pm 1/2}$.

Also, $\langle S_{1_x} \rangle^{0}_{\pm 1/2,0} = 1/2 \left[\langle S_{1_x} \rangle^{0}_{\pm 1,0} + \langle S_{1_x} \rangle^{0}_{0,0} \right]$

and similarly for $\langle S_{2_\eta} \rangle^{0}_{\pm 1/2,\,0}$, $\langle S_{1_x} \rangle^{0}_{0,\pm 1/2}$ and $\langle S_{2_\eta} \rangle^{0}_{0,\,\pm 1/2}$

The second order differences are expressed in the following way for any function ψ

$$\left\langle (a\psi_x)_x \right\rangle^0_{0,0} = \frac{1}{\Delta x^2}\left[a^0_{+1/2,\,0}\left(\psi^0_{+1,0} - \psi^0_{0,0}\right) - a^0_{-1/2,0}\left(\psi^0_{0,0} - \psi^0_{-1,0}\right)\right]$$

$$\left\langle (a\psi_x)_\eta \right\rangle^0_{0,0} = \frac{1}{4\Delta x \Delta \eta}\left[a^0_{0,+1/2}\left(\psi^0_{+1,0} + \psi^0_{+1,+1} - \psi^0_{-1,0} - \psi^0_{-1,+1}\right) - a^0_{0,-1/2}\left(\psi^0_{+1,0} + \psi^0_{+1,-1} - \psi^0_{-1,0} - \psi^0_{-1,-1}\right)\right]$$

and similarly for $\left\langle (a\psi_\eta)_\eta \right\rangle^0_{0,0}$ and $\left\langle (a\psi_\eta)_x \right\rangle^0_{0,0}$ where

$$a^0_{\pm 1/2,0} = 1/2\,(a^0_{0,0} + a^0_{\pm 1,0})$$

and

$$a^0_{0,\pm 1/2} = 1/2\,(a^0_{0,0} + a^0_{0,\pm 1}) .$$

The value of W determined at the intermediate points are then used to define new W's in the $t + \Delta t$ time plane at $x = x$ and $\eta = \eta$ according to the equation

$$W^{1}_{0,0} = W^{0}_{0,0} + \Delta t \left\{ \left\langle F_x \right\rangle^{1/2}_{0,0} + \beta e^{-\eta} \left\langle G_\eta \right\rangle^{1/2}_{0,0} + \left\langle S_{1_x} \right\rangle^{0}_{0,0} + \beta e^{-\eta} \left\langle S_{2_\eta} \right\rangle^{0}_{0,0} \right\}$$

where

$$\left\langle F_x \right\rangle^{1/2}_{0,0} = \frac{1}{\Delta x} \left[F^{1/2}_{+1/2,0} - F^{1/2}_{-1/2,0} \right]$$

and

$$\left\langle G_\eta \right\rangle^{1/2}_{0,0} = \frac{1}{\Delta \eta} \left[G^{1/2}_{0,+1/2} - G^{0}_{0,-1/2} \right] .$$

REFERENCES

1. W. J. Cook and G. T. Chapman, The Physics of Fluids, 15, 12 (1972).

2. R. N. Gupta and R. L. Trimpi, 9th International Shock Tube Symposium (Stanford Univ., U.S.A., 1973), p. 449.

3. D. E. Abbott, J. D. A. Walker and H. T. Liu, 9th International Shock Tube Symposium (Stanford Univ., U.S.A., 1973), p. 462.

4. W. D. Hayes and R. F. Probstein, Hypersonic Flow Theory (Academic Press, New York, 1959).

5. H. U. Thommen, ZAMP, 17, p. 369 (1966).

Number of Grid Points In Computational Field	Total CPU Time, Sec.		
	6600		STAR
	Compiler		
	Run	FTN	
168	3.002	1.707	0.2218
280	19.009	10.830	1.5450
336	44.819	24.640	2.2060
504	151.88	87.640	6.2220
2800	21600.	*	321.60

*Data not available.

Table 1.- Comparison of CPU Times Required for Solutions on the 6600 and Star Machines.

Boundary Vector Length	CPU Time In Seconds	
	Vector	Scalar
7	.009326	.0075
9	.009550	.0101
10	.009612	.0113
11	.009899	.0126
13	.010037	.0151
15	.010246	.0176
19	.010575	.0226
32	.011705	.0388
50	.013209	.0613
71	.014966	.0875
82	.015997	.1012

Table 2.- A Comparison of Identical Work for Scalar vs. Vector Boundary Condition Computations.

	VECTOR BOUNDARY CONDITIONS (B.C)			SCALAR BOUNDARY CONDITIONS		
Boundary Vector Length	Total CPU	Total CPU spent doing B.C.	% of total time doing B.C.	Total CPU	Total CPU spent doing B.C.	% of total time doing B.C.
11	4.3748	1.9023	43.5	4.3278	1.8537	42.8
26	29.601	10.387	35.1	39.315	20.096	51.1
50	96.917	27.407	28.3	154.01	84.470	54.8
75	197.55	47.372	24.0	342.01	191.85	56.1

Table 3.- CPU Utilization, Scalar vs. Vector Boundary Calculations.

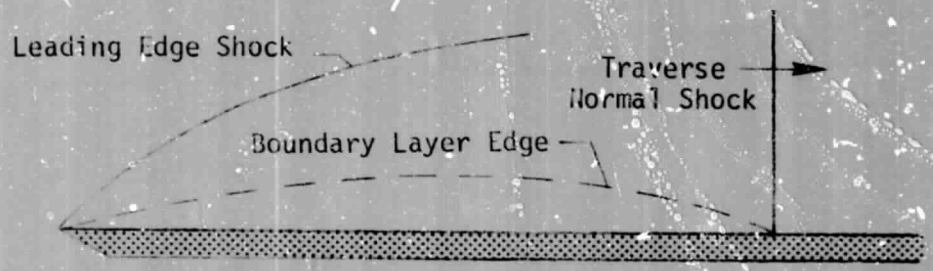


Figure 1.- Schematic diagram of flat plate flow.

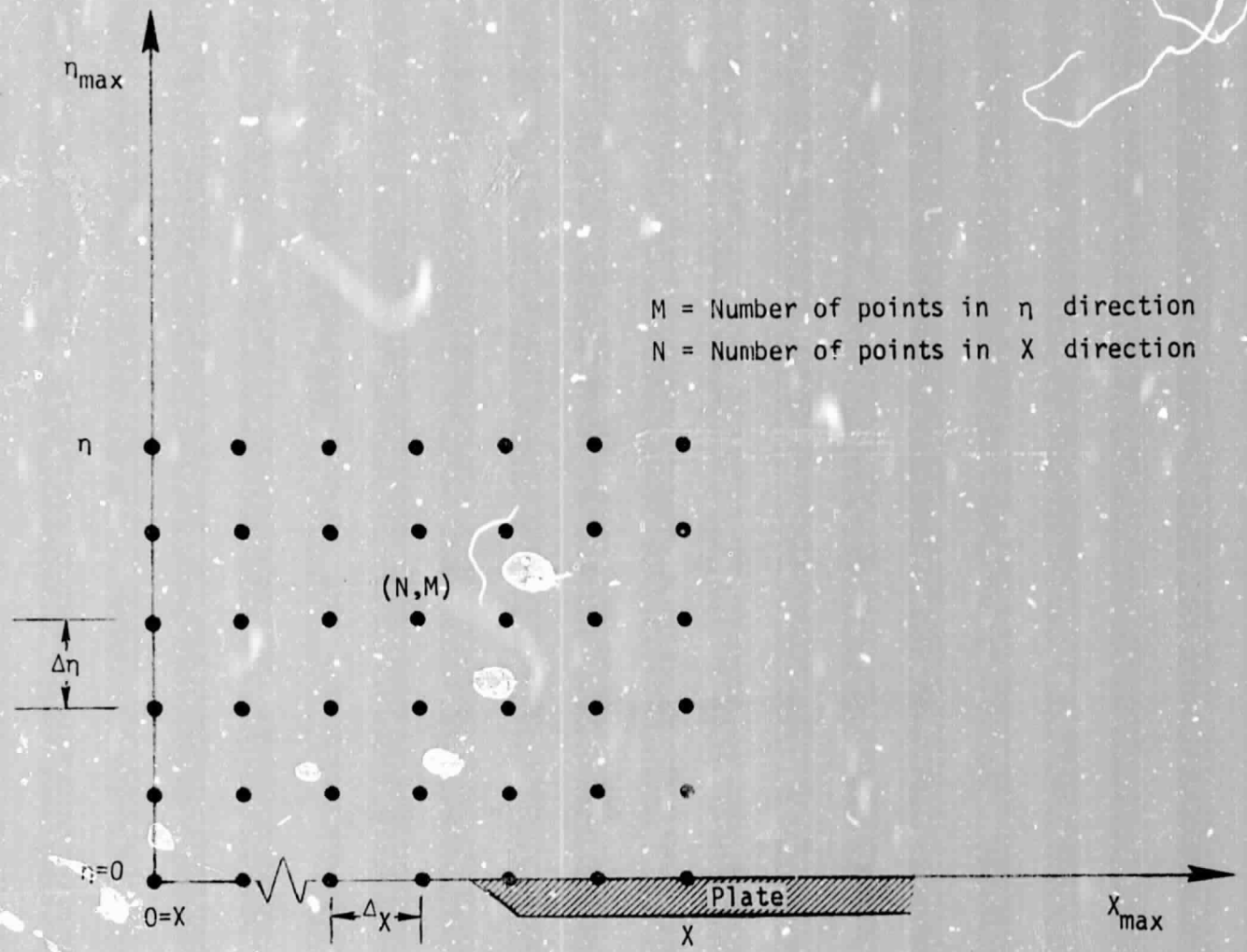


Figure 2.- Computational grid.

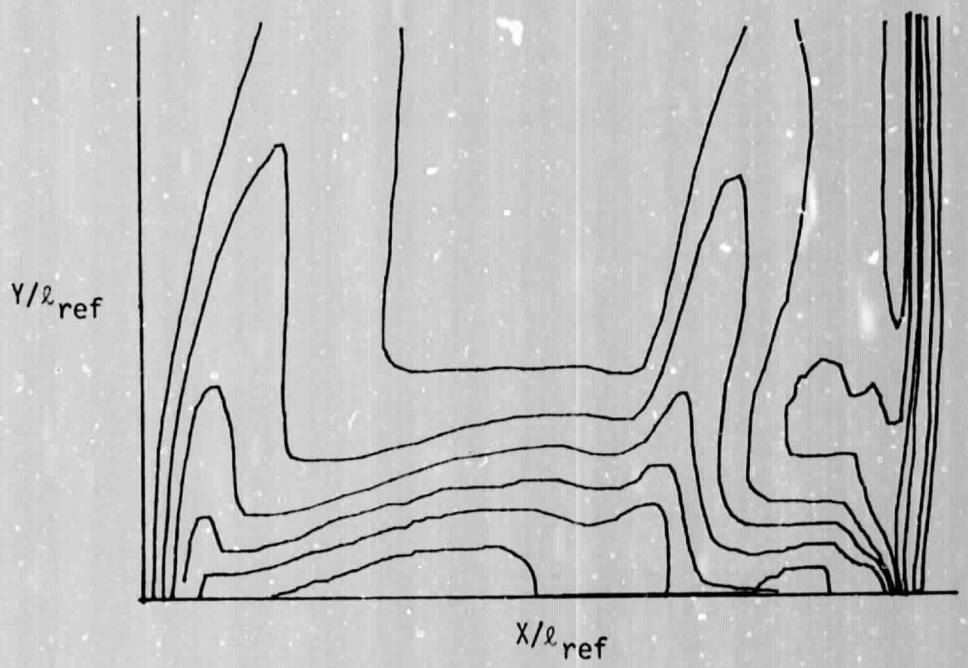


Figure 3.- Flow field isotherms at an instant in time.

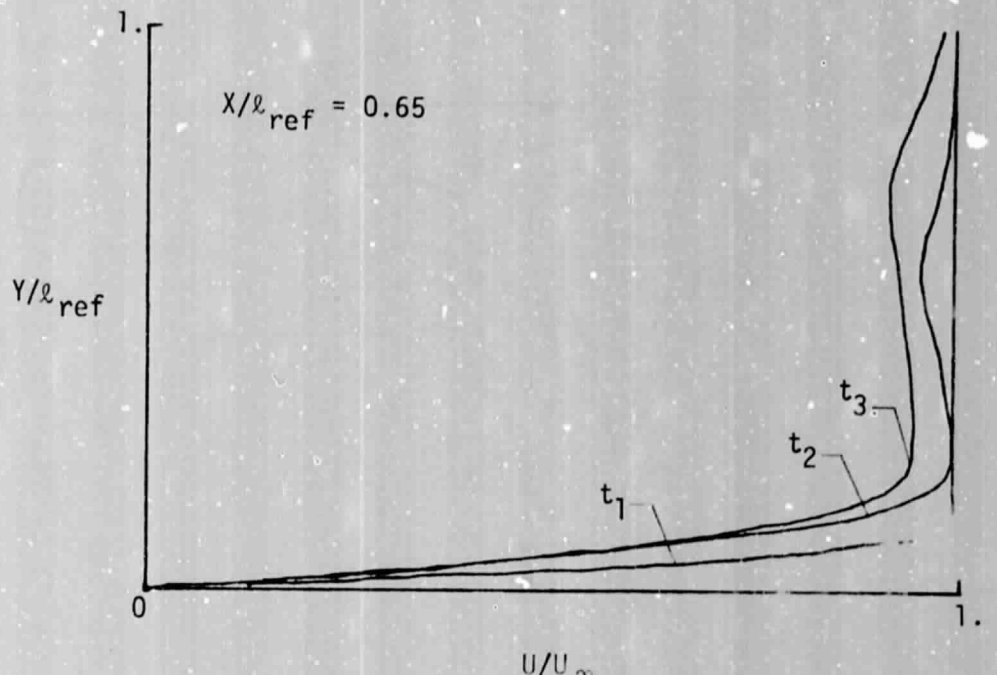


Figure 4.- Time-dependent velocity profiles, $t_1<t_2<t_3$.