Techniques for using the System/360 Operating System for implementing a large fluid dynamics program are discussed as a prototype for the solution of other coupled nonlinear partial differential equations.

Presented also are methods for real-time interaction with the problem solution.

A graphic analysis program for producing computer animated motion pictures is outlined.

System aspects of large-problem computation and display

by J. E. Fromm and D. E. Schreiber

Progress of scientists in producing numerical solutions to fluid dynamics problems by digital computer parallels advancements in the design and programming of the computers themselves. The reason fluid dynamics problems continue to absorb the computational power of available computing systems is that the equations of fluid dynamics are extraordinarily complex. Further, comparitively few solutions for fluid dynamics problems are known. Therefore, digital computers have been programmed to simulate the fluid flow, using the equations as models. John von Neumann, for example, thought that the simulation of global atmospheric circulation would provide a suitable problem for these machines. Over the years, many programs have been written for solving meteorological problems so that today the U. S. Weather Bureau routinely produces long-range forecasts. More programs for solving fluid dynamics problems will continue to be written because progress in the solution of these problems is expected to continue to be coupled to advances in computing system capability. Augmenting advances in computing systems, the vastness of fluid dynamics problems has motivated a number of scientists to develop more efficient methods of using the computing potential available to them. Kolsky,[1] for example, discusses methods for improving the logical formulation of large problems and illustrates his techniques by the solution of meteorological problem.

This paper discusses several facilities available through the System/360 Operating System (OS/360) that have made easier

and more productive the implementation and running of a program for computing the time-dependent equations for viscous fluid flow. Toward this end, the fluid flow program has been designed to run under versions of OS/360 that are capable of multitasking operation, i.e. multitasking with a fixed number of tasks (MFT) or multitasking with a variable number of tasks (MVT). The application has been programmed almost entirely in FORTRAN. Assembler language is used only where necessary or where highly desirable features of the operating system are unavailable through FORTRAN. Program compatibility has allowed us to successfully run the application on System/360 Models 50, 65, 67, 91, and 195.

equations and simulation

The implemented program computes a stream function by the following time-dependent vorticity equation that has been derived through Navier-Stokes equations for two dimensional flow:

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}=-\omega \tag{1}$$

where

ψ is the stream function

and

ω is the vorticity.

We also define the two velocity components

$$u=\frac{\partial\psi}{\partial y} \tag{2}$$

and

$$v=-\frac{\partial\psi}{\partial x}$$

where

u is the x component of velocity

and

v is the velocity component in the y direction.

Since a viscous fluid adheres to a wall and has no velocity relative to the wall, a viscous fluid has internal shearing stresses associated with it. The effect of such viscosity on vorticity is expressed by the following equation:

$$\frac{\partial\omega}{\partial t}+\frac{\partial(u\omega)}{\partial x}+\frac{\partial(v\omega)}{\partial y}=\nu\nabla^2\omega \tag{3}$$

where

ν is the kinematic viscosity.

This set of coupled nonlinear partial differential equations is solved by computer simulation through the use of finite-difference techniques. The program simulates the solution in a rectangular space that is spanned by an M-by-N point rectangular mesh. Values for the stream function and vorticity are computed at the mesh points. The simulation is performed in the following manner:[2]

1. Compute the stream function using Equation 1.
2. Compute the velocity components using Equation 2 and an optimum time-advance interval, thereby assuring mathematical stability.
3. Use this interval to advance time.
4. Compute the vorticity at the advanced time using Equation 3.
5. Return to step 1.

Since this kind of program consumes a great deal of computer time, we discuss our planning of the types and organization of data in storage that enable the compiled FORTRAN program to run efficiently. Easy modification of the computational algorithm is designed into the program by functionally partitioning the computation into subroutines. Also, to aid the scientist working with the program, the computer is used to perform some routine program initializations, simply because in that way fewer human errors occur. Finally, although the output of the program is numerical, these data can be also given graphic interpretations that are meaningful to the scientist. A graphic analysis program provides interactive data reduction and analysis and computer animated motion pictures.

Data types and storage

Since finite difference techniques are used to solve the entire set of equations, the program must have data storage for each of the functions for at least one time step and for all $M \times N$ mesh points. These data are stored in arrays of doubly subscripted, single-precision, floating-point variables. Single precision variables are used because six decimal places far exceed the precision of either the measurements or the experimental apparatus available. There would seem to be little profit, in terms of the quality of the computation, to be gained by going to double precision variables and arithmetic operations.

The computational arrays vary in size from around 1000 to 4000 mesh points. An increase in the number of mesh points provides higher resolution, which however, increases simulation costs by requiring the processing of more data points and the use of smaller time steps to assure mathematical stability. For an $N \times N$ square mesh, the number of operations per time step is proportional to N^2 and the maximum stable time step is proportional to

$1/N$. Thus, for a given simulation, doubling the resolution of the mesh increases the computation by a factor of 8 (i.e., 4 times as many computations per time step and 2 times as many steps).[3]

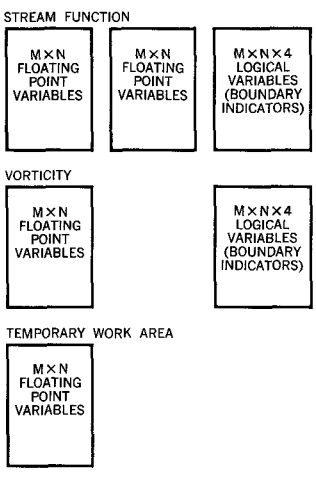
boundary indicators

The finite-difference method is one that treats boundary value problems. To make the program as general as possible, it is designed to handle boundary specifications for points throughout the mesh. Boundaries define an obstacle in or on the periphery of the flow region. To indicate that such a boundary exists and what type of condition occurs there, arrays of indicators are used. Since the information carried by these indicators is not quantative but logical (yes or no), FORTRAN logical variables of length one are used. Four such indicators are used for each point in the mesh. This choice of data representation wastes seven-eights of the main storage occupied, and the data could just as well be handled on a bit rather than byte basis. The testing of bit indicators from FORTRAN, however, involves a great deal of data manipulation in FORTRAN to isolate the bit to be tested. One solution is to use a linkage to and return from an assembler language function that performs the test and returns a code that FORTRAN retests. Alternatively, one could modify the FORTRAN compiler so as to recognize a new data type, LOGICAL*1/8, and insert the appropriate operation, Test Under Mask Immediate, inline into the compiled program. These three schemes have been rejected either because of the large number of instructions to be executed to obtain the information or because of the loss of OS/360 compatability. By this decision, we choose faster program execution over efficient utilization of main storage.

storage allocation and handling

In our problem, each of the two variables—stream function and vorticity—requires one or more data arrays, depending on the computational algorithm used. As indicated in Figure 1, both the stream function and the vorticity occupy $M \times N$ floating-point, single precision words of storage. In addition, each variable requires one array of boundary indicators occupying $M \times N \times 4$ bytes of storage. The computation also uses a temporary work area. The required arrays are allocated dynamically at program execution time with the appropriate arrays and array dimensions being passed through calls to the computational subroutines. Thus after a subroutine is coded and debugged, it can be stored in a library for future use without any concern as to the array size.

Figure 1 Storage allocation



Since FORTRAN does not provide for dynamic storage allocation and since it further requires that array addresses and dimensions can be passed down only from calling routine to called routine, a short main program in assembler language is necessary. This program first makes a FORTRAN-required call to IBCOM# to establish the FORTRAN SPIE routines. The assembler language program then calls the FORTRAN subroutine DATAIN, which reads data cards that specify the parameters of the computation and

the data array dimensions. DATAIN further computes some constants required by the algorithms, thus performing that computation only once for the entire execution of the fluid dynamics program. DATAIN stores the data that it has read and computed into common areas, and returns to the assembler language main program. This program then computes the number of bytes of main storage required for data and indicator arrays and issues a GETMAIN macroinstruction. Upon return from the supervisor, the assembler language program computes the addresses required to partition the allocated storage into the individual arrays and stores these addresses into a FORTRAN call argument list. The assembler language program then calls a FORTRAN subroutine and passes the list to the subroutine. This subroutine is in fact the FORTRAN main program, which organizes the computational steps and calls the other subroutines for executing the computational algorithms.

Program design

The array handling and allocation features thus implemented have permitted easy experimentation with array size to determine their effect on results of a simulation. These features also permit experimentation with different computational algorithms, some of which—such as Fourier methods—may place restrictions on the dimensions of data arrays. The combination of ease of boundary specification provided by the indicator arrays and of data array specification through the input data stream have permitted quick setup of the program to simulate different problems.

program setup

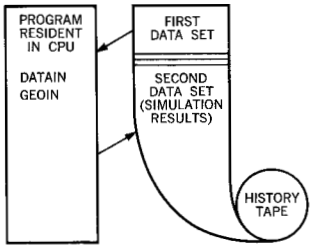
The indicator arrays, which remain fixed during a computation, are set up only once—just prior to entering the computational loop—by a subroutine called GEOIN. The subroutine GEOIN reads data records that specify the end points and conditions on the boundaries, and sets the appropriate indicators and functional values as specified in the boundary description. Boundaries are linear and are restricted to the horizontal, vertical, or diagonal with respect to the mesh. The simulated angle of a diagonal boundary need not be 45 degrees. It is possible to specify in the input read by DATAIN the ratio of delta x to delta y of the mesh, thereby changing the simulated angle of a diagonal boundary by either compressing or stretching the mesh. All diagonal boundaries, however, in a given simulation must be at the same angle.

program execution

A single simulation may take several thousand time steps and consume several hours of System/360 Model 91 Central Processing Unit (CPU) time. Running in a job shop environment with other tasks executing concurrently, the simulation program may

stay in the system for several tens of hours to complete a computation. In such an environment, this type of program can have a lower cost to run than the normal batch jobs provided that it runs at the lowest system priority, taking up only otherwise unused CPU cycles. In such a case, the real cost to run the program is the cost of main storage occupied and of allocated devices—such as tape drives—that cannot be shared with other tasks. Since we make production runs in this manner, the simulation program is designed to resist data loss in the event of system failure and to be easily restartable—preferably without human intervention other than rereading the job into the system.

Figure 2 Simulation program execution



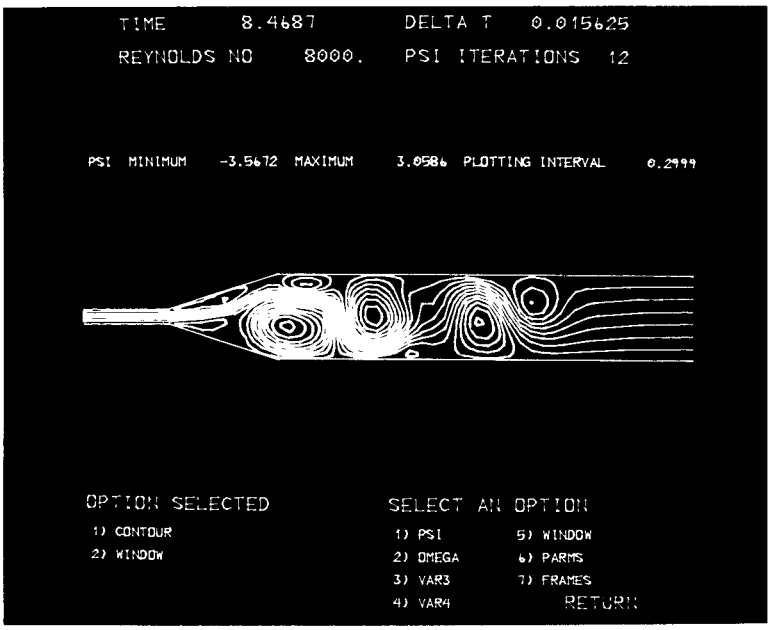
The basis of the restart capability of the program is that of writing out all the data arrays on tape every few time steps as shown in Figure 2. These arrays are written into a sequential data set, and thus also constitutes a time history of the simulation. The first volume of tape associated with a given computation contains the following two data sets: (1) the data records read by DATAIN and GEOIN, and (2) the results of the simulation. When a computation is restarted, the simulation program reads the first data set to establish the program state that existed when the computation began. The program then scans down the second data set until it finds one of the following: end of file, I/O error in that data set, or a simulated time greater than that specified on a second input source to the program. Having satisfied one of these three conditions, the program reads the data arrays from the tape, thus reestablishing the program as it was.

The feature of entering a computational restart time facilitates program experimentation. If a new algorithm is being tested, one may use a prior computation as a starting point for further experimentation and comparison. The runs required to develop and debug the new technique begin at the same advanced time in such a simulation. When a new program is perfected, it can be used to continue the simulation, thus providing a good comparison between a new and an old technique.

Graphics analysis

History tapes have additional uses besides providing a restart capability in that they constitute self-contained machine-readable archives of computations suitable for later processing by data reduction and analysis programs. A program has been written to perform some of this analysis interactively through an IBM 2250 display console. Interactivity has the same value to a simulation (numerical experiment) as it does to a physical experiment. In both cases, the experimenter has some a priori knowledge of what he is looking for. However, the precise techniques and parameters that yield most clearly the sought-for information are un-

Figure 3 Stream function contour map



known. Cut and try, the usual approach to experimental interaction, is facilitated if it can be completed in one run on the computer. Further, the ability to provide graphic interpretations of the data make the analysis even more fruitful. Our graphic analysis program produces contour maps[4] of up to four different data arrays. Provided through the program via the 2250 display console are the following functions:

- Interactively controlling the data to be contour mapped
- Presenting the region of the array to be plotted
- Plotting parameters
- Reading a history tape and printing the set of data arrays for a given simulated time

The presentation of the computational results in the form of contour maps gives quantitative insight into the problems being studied. As illustrated in Figure 3, a contour map of the stream function shows the fluid flow structure. At any point, the tangent to a contour of the stream function is parallel to the velocity vector. If a steady flow is being simulated, a particle initially on a stream function contour continues to travel along that contour. The fluid flows that we are simulating are generally not steady, however, and this interpretation cannot be made. Additional information provided by a stream function contour map is the relative velocity of the flow. A region with closely spaced stream function contours has a higher velocity than one containing fewer contours.

The interpretation of vorticity contour maps, an example of which is shown in Figure 4, is more complicated. A gradient in the vorticity, revealed by the presence of contours, indicates a nonlinear velocity profile along a line orthogonal to the contours. Vorticity contour maps give insight into the interaction of the flow with the boundaries. Computer animated motion pictures of vorticity contour maps discussed later in this paper, increase our insight into these processes.

The run-time interactive graphic facility illustrated in Figure 5 is implemented by running two separate jobs that share three separate data sets located on direct-access devices. One job is the simulation program, which writes the results of the computation into one of the direct-access data sets every simulated time step. The other job is the graphic analysis program, which runs at a higher priority than the simulation and reads the computation data set periodically to determine if its contents have changed. If an update has occurred, the graphic analysis program produces a contour map of the updated data set. At the discretion of the 2250 display console operator, interaction with the graphic analysis program immediately produces the desired display. The other two data sets are used to exchange data between both jobs. The second data set roughly corresponds to the data records read by DATAIN, and the third data set corresponds to the records read by GEOIN.

Before an access is made to any of these data sets, the program desiring to access it enqueues it through the use of the OS/360 ENQ macroinstruction. After the program is through using the data set it similarly dequeues (DEQ) that resource. The ENQ and DEQ macroinstructions are issued from assembler language subroutines. The objective in implementing ENQ/DEQ in this manner is the efficient use of storage and graphic devices. Since both jobs run independently of each other, they can be started and terminated separately. Thus the scientist can monitor the computation for some time then terminate the graphic analysis program without terminating the computation, and return at a later time to resume his observations. During the time that the graphic analysis program is not running, the CPU, main storage, and the 2250 are available to other users.

interaction data reduction

Initially, we postulated that interactive graphic analysis would have value during run time as well as during later analysis because of the capability of monitoring a computation from the 2250 display console while the computation was actually taking place and because of the desirability of modifying parameters of the computation interactively during the run. This capability was designed into the data structures in peripheral storage and also into the programs, but it was never implemented. Operational experience has shown us that for currently available computers

Figure 4 Vorticity contour map

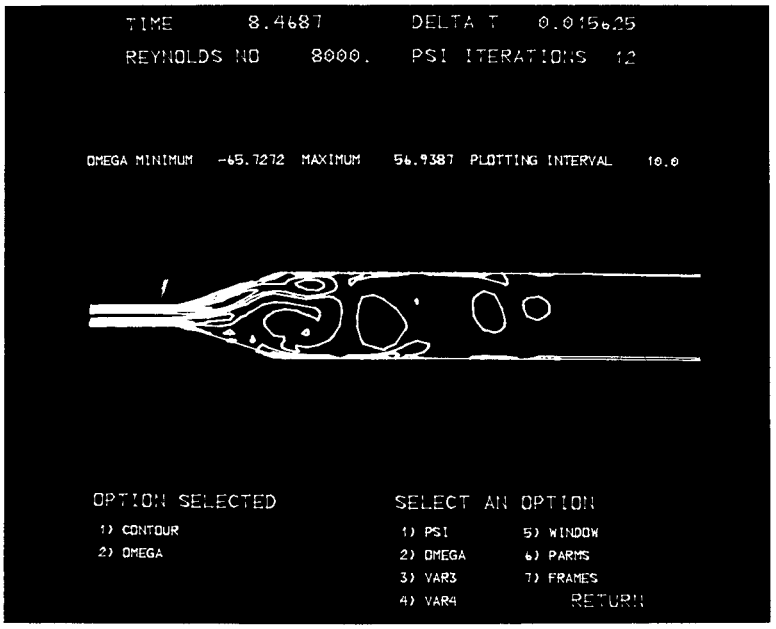
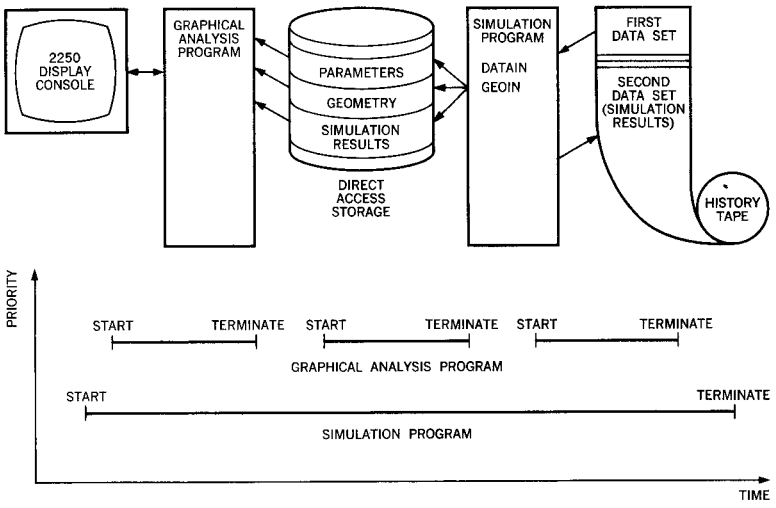


Figure 5 Interactive graphic simulation monitor and control



such a facility is not required. The technique would use an indicator as the first data item in the records in peripheral storage. The indicator would be set to a specified value by either the graphics or the simulation program, depending on which program had performed the latest update of the data set. Each routine would have to read and test the indicator before rewriting the data set.

During long production runs, the progress of the computation is sporadic because of the contention of the fluid flow simulation with other concurrently running programs for the CPU. The experimenter is unwilling to wait five minutes until the data for the next plot are computed. Interaction to correct data or programming errors during the start of a run is meaningful, but the chance is small that interaction is required after a run is well underway. Opportunities to run an application diminish quickly, however, as requirements for its running increase. Programming an application that requires several hundred-thousand bytes of main storage and one or two tape drives to run on System/360 Model 91 with two-million bytes of main storage is not severe. However, when one adds one-hundred thousand bytes of storage required for the interactive capability, a 2250, and the physical presence of the scientist to the other requirements, the rewards of interaction are not worth the increased difficulty. Future generations of higher-speed computers may improve the marginal utility of on-line interaction.

computer animation

Given a sequence of photographs that progressively vary only slightly from one another, the natural impulsive desire is to make a motion picture. We photograph the sequence of contour maps of simulations directly from the screen of the 2250. The photography is performed under program control, and the number of frames of film exposed from a single plot can be specified interactively from the 2250 display console.[5]

Films of these simulations are most useful because they display a voluminous amount of information in a short time in an easily recognized and studied format. A single simulation may compute 10^7 numbers, the meaning of which the viewer sees and recognizes in one or two minutes when they are presented in the form of a motion picture. Obviously, this is a tremendously powerful medium of communication between the computer and the user. After a film is produced, one can study the results of a simulation at a very low cost. Understanding a simulation usually requires multiple viewings of the graphic output. Motion picture playbacks are ideally suited to multiple viewing because they are less prone to failure than computers, easier to schedule, and can be operated by amateurs. A collateral use for graphic analysis films is the presentation of computational results to both the layman and the scientist. Although the scientist gains more from computer animations, the layman feels thoroughly competent to comment critically on any subject presented to him through the medium of motion pictures. Presentation of the same data through the vehicle of equations and numbers terrifies him into silence.

Animation of the stream function display shows the location of circulations of the fluid by the presence of families of closed con-

tours. Contours beginning and ending on the boundaries show either the path of flow through the simulation (if the boundary condition simulates fluid flowing into and out of the simulated region) or the line of no flow (if the boundary is a rigid wall). Through the vehicle of animation, contour maps of vorticity come to life. The mathematical construct of vorticity is generated at the boundary by the shearing that takes place between a rigid wall and the flowing fluid. After it is created, vorticity travels by convection in the flow and spreads via diffusion mediated by viscosity. Using animation, one sees vorticity created, convected, and eventually dissipated in the fluid. One also observes vorticies coupling and influencing each other's motion through the simulated region, and then coalescing into a single vortex. Further, since vorticity is convected by the flow, its motions in low-viscosity simulations closely parallel those of tracer particles. One can obtain a good idea of the motion of parts of the fluid by following the motions of a single vortex from the time it detaches from the wall where it was generated until dissipation or coalescence causes its identity to disappear.

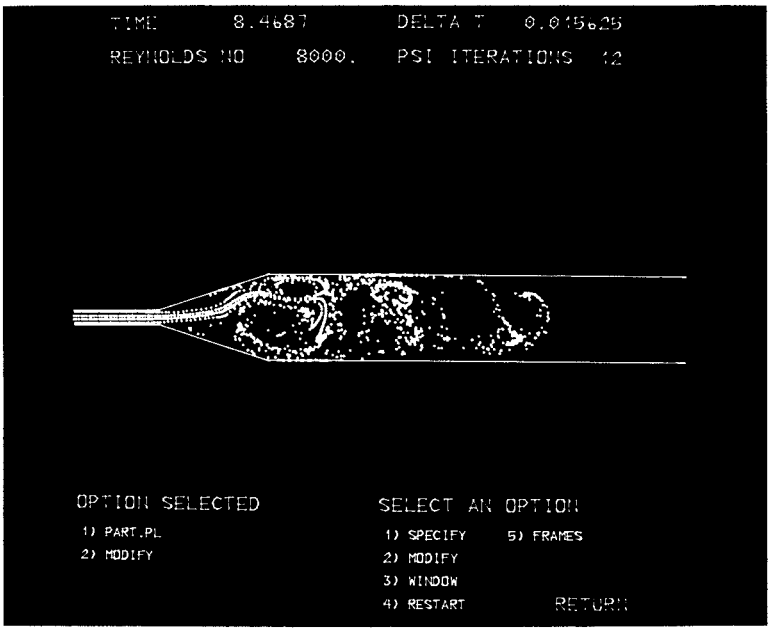
particle displays

The graphic routines so far described permit us to display and study the theoretical constructs of stream function and vorticity. Such displays are certainly of value in studying the simulation. Proof of the simulation technique, however, lies in comparison with experiment. An experimental technique widely used for studying flow structure is that of tracers such as dye, smoke, or metalic particles that are convected by the flow. The experimenter observes flow structure as patterns revealed by the tracer or by following the motions of individual tracer particles.

We have implemented an analogous facility within our graphic program that permits the scientist to interactively define up to ten particle lists. Each list is unique in that associated with it there is a set of characteristics that direct the computation to be performed on that list. For example, a list can be defined that simulates the action of a dye or smoke injector by specifying the location of the source of injected particles with respect to the computational mesh. A graphic display of a run using a source list is shown in Figure 6. Another kind of list can be defined that consists of a straight line of M particles at time T. As simulated time passes, the particles making up the line are convected by the flow, and the line is generally deformed into a curve. After sufficiently long simulated time, the curve breaks up into the individual particles. A graphic display resulting from a run using a line of particles is shown in Figure 7.

The particle display facility permits the production of motion pictures similar to those produced by photographing a tracer experiment. Since they are massless and occupy no space, computational tracer particles have an advantage over experimental

Figure 6 Display computed using particles from a source



ones in that their presence does not influence the experiment. Particles thus provide a tool that permits detailed comparisons of simulations with experiments.

The particle plot data storage shown in Figure 8 consists of two parts: (1) a FORTRAN labeled common area that contains the data that define and direct the processing of the ten individual particle lists, and (2) the x and y velocity component arrays and particle position data. Note that two configurations A and B of velocity-position data storage are shown, depending whether scratch I/O storage is required. For particle plotting, the system uses either the no-scratch-I/O storage configuration (A) or the scratch-I/O storage configuration (B) plus the FORTRAN labeled common area. (Particle list computing and contouring are mutually exclusive operations. Therefore, the same area of main storage is used for both particle plot data list computing and contouring data by the process of storage overlaying.)

Since the number of particles in a given list is dynamic, the program checks storage allocation for the particle lists every time step to determine which of the two conditions—A or B—is the active one. Thus, if all particle position data is contained in main storage, we have condition A. On the other hand, if the storage required for the entire set of particle lists exceeds the available storage we have condition B. With scratch I/O, the program splits the particle position data area into two parts. The first part is

Figure 7 Display computed using a line of particles

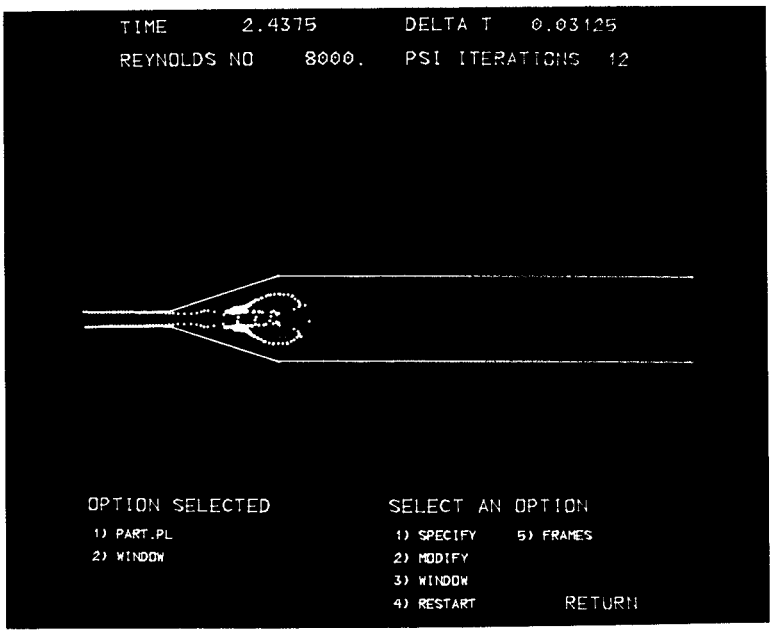
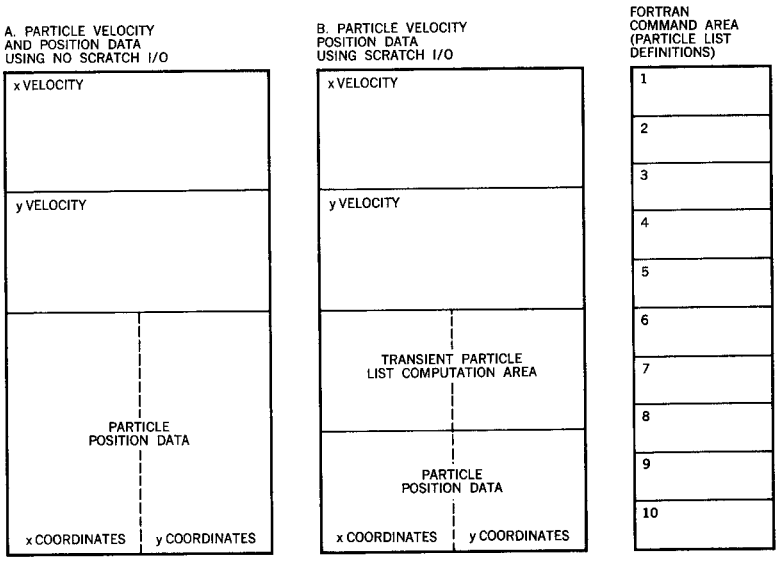


Figure 8 Particle plot data storage



used as a transient computation area for the longer particle lists which are stored on scratch I/O. During each time step the particle lists so stored are read, updated, and stored back onto scratch I/O. The other particle position data reside in main storage and are processed there. If particle position data lists con-

tinue to grow and finally exceed the storage available, the longest of these lists is added to those already on scratch I/O. In this manner, the program frees the scientist from concern for the storage requirements of the particle plot computation. All that need concern him is that he is obtaining the picture of the simulation that he requires.

The data in common that describe the particle position and control particle processing are created, maintained, and updated by a combination of routines. Interactive routines permit the scientist to define and modify the description of the particle lists and to direct their processing. If necessary, a storage allocation routine updates the storage allocation data of the particle lists. The computational routines, which update the particle positions each time step, refer to the appropriate data list stored in common to perform the computation and produce the desired display. Updating the particle positions requires the x and y velocity data for every time step of the simulation. These data are prepared prior to the running of the graphic analysis program and are read, as needed, by the graphic analysis program.

The particle plot may be animated simularly to the contour maps as shown by the source and line particle plots in Figures 6 and 7. An appreciable amount of real time is required to reach middle or late simulation times in the production of a movie. Therefore, the particle plot portion of the graphic analysis program imposes the same recovery conditions as the actual simulation. This protection is provided by writing the state of the particle plot computation into a sequential data set every time step. In the event of a program restart or of the scientist interactively asking to begin the particle plot from the beginning, this data set is read by the graphic analysis routine. When the end of this data set is encountered, the program restores itself to the state described in the last complete record, positions the velocity tape to the proper simulated time, and then resumes the computation of the plot. Once again, this all occurs without the direct intervention of the scientist. His only concern is the display on the console. This feature prevents a system failure that occurs three-fourths of the way through the production of a movie from forcing a restart from the beginning.

Concluding remarks

In designing and implementing programs for simulating the Navier-Stokes equations of incompressible viscous flow and for displaying the simulation results, the following features of the System/360 Operating System are particularly useful:

- FORTRAN variable-dimensioned arrays
- FORTRAN data set sequence numbers

- Assembler language in combination with FORTRAN
- Execution-time storage allocation
- Low-priority program execution in a multiprogramming environment
- Multi-data set volumes and multi-volume data sets
- Multiprogram sharing of data sets resident in direct-access storage

The resulting application program structure uses FORTRAN capabilities efficiently and provides a simple restart procedure. These capabilities permit ease of changing the system being simulated and/or the algorithms being used while making possible efficient computer operation. The restart capability also provides protection against computer failure.

An interactive graphic analysis program permits the scientist to study both the theoretical constructs of the simulation and the results as they would be seen by an experimenter. Programs have been designed to use the IBM 2250 both as interactive graphic device and as a film recorder.

The programming and storage structuring techniques discussed here are intended to suggest ways by which many other systems of nonlinear partial differential equations may be more effectively simulated and analyzed graphically.

REFERENCES

1. H. G. Kolsky, "Problem formulation using APL," *IBM Systems Journal* **8,** No. 3, 204–219 (1969).
2. J. E. Fromm, "Numerical solution of two-dimensional stall in fluid diffusers," Symposium on high speed in computing fluid dynamics, *Physics of fluids, supplement* **II,** II–113 (1969).
3. J. E. Fromm, "Numerical method for computing nonlinear, time dependent, *buoyant* circulation of air in rooms," *IBM Journal of Research and Development*, **15,** No. 3, 186–196 (May 1971).
4. G. Cottafava and G. LeMoli, "Automatic contour map," *Communications of the ACM*, **12,** No. 7, 386–391 (July 1969).
5. D. E. Schreiber, "Computer control of a camera for motion picture generation from an IBM 2250", *IBM Research Note*, RJ 666 (February 1970) may be obtained from the IBM Research Center, Yorktown Heights, New York 10598.