NASA
Technical
Paper
2179

June 1983

Parallel, Asynchronous Executive (PAX): System Concepts, Facilities, and Architecture

William H. Jones

25th Anniversary
1958-1983

NASA

NASA
Technical
Paper
2179

1983

Parallel, Asynchronous Executive (PAX): System Concepts, Facilities, and Architecture

William H. Jones
*Lewis Research Center
Cleveland, Ohio*

NASA
National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

Summary

In the mid-1970's, the author began the development of CASPER, a collection of fluid-flow simulation routines. As development proceeded, it became apparent that CASPER could be worked on by virtually identical programs at the same time. A large calculation could be divided into segments that segregated inputs from outputs, and logical data-base records could be arranged into physical mass-storage records that could be independently read and written. Because of the enormous computational size of CASPER, the author decided to implement this idea as the Parallel, Asynchronous Executive (PAX).

The following features have been accomplished in the current implementation of PAX:

(1) PAX splits one segment of a calculation into fully asynchronous, parallel tasks.

(2) PAX manages any number of parallel processors.

(3) PAX manages any serial aspects of the problem, including those necessary to resolve parallel-processing conflicts.

(4) PAX provides facilities for error and fault reporting and recovery.

(5) PAX and its parallel processors can be stopped, changed, and restarted without loss of position in a computation. Thus programming errors can be repaired without lossing results calculated before the problem occurred.

6. PAX provides communications facilities for interaction with machine operators.

This report details the fundamental concepts, facilities, and architecture of PAX. PAX manages the execution of CASPER (Combined Aerodynamic Structural Dynamic Problem Emulation Routines), a program to simulate airflow through arbitrary flow fields. CASPER is not discussed in this report except to provide examples of parallel-processing techniques. The current implementation of PAX is exploratory and experimental. PAX is a vehicle for pointing the way to fully developed parallel, asynchronous processing systems.

Introduction

Historically, computing machines have executed a logically unified task in a step-by-step fashion. As computer size and speed increased, variously complex software structures (operating systems) allowed machines to work on many problems in a quasi-parallel manner; however, these problems were logically unrelated in that, as far as the machine was concerned, the output of one problem did not affect the outputs or inputs of another problem. Each logically unified problem still had to be approached in a step-by-step manner, regardless of whether that serial relationship was actually required by the problem itself.

This serial structure of computing organization is in sharp contrast to human organizational structure, which is parallel and asynchronous. Many average workers can be organized to form a formidable work force to produce a product that would take one person thousands of years. For many valid reasons, rather than change organizational strategy, a new (faster) implementation of an existing computer architecture usually has been produced to increase performance to meet new demands (witness the progress of IBM 360, 370, 370/3033, 370/3081).

In the past few years an extension of serial computing has appeared in the form of vector processors (as offered by Control Data Corporation and Cray Research Corporation), but these still have not broken from the fundamentally serial approach to logically unified problems. Certainly these machines have great merit. Although serial organization constrains machine architecture, it offers the utmost in algorithmic flexibility. The problem on the horizon for even these vector processors is the fact that, sooner or later, technology will reach a limit beyond which the serial organization cannot proceed. The vector processors acknowledge this limit by processing vector commands in parallel.

PAX attempts to organize a highly parallel, asynchronous computing environment by using the human experience as a model. This approach is not without its difficulties. Chief among these is the fact that the management system must have a much greater knowledge of the problem to be managed than has been required in the past. Simply knowing a memory requirement, a mass-storage requirement, and a set of connections to some undefined (in the system's terms) user is not adequate to organize many machines to work in parallel on a common problem. PAX is an attempt to deal with this organizational problem in a realistic manner. Two fundamental facts guided initial PAX design: (1) any parallel, asynchronous processor system would be subject to random failures of its processing components and (2) all problems generate some procedural sequences that must be serialized. Thus the initial design of PAX went beyond simple parallel processing to management of real parallel machines and to features appropriate to a real parallel problem.

PAX is an entry into the well-populated field of highly parallel computing. Haynes, Lau, Siewiorek, and Mizell in a recent survey article (ref. 1) identify six classes of highly parallel computing machines: (1) special-purpose functional units, (2) associative processors, (3) array processors, (4) data-flow processors, (5) functional programming-language processors, and (6) multiple general-purpose processors. PAX is designed as a management system for the sixth class of highly parallel computers. Haynes et al. go on to identify an "extra hard" class of scientific problems (usually involving nonlinear, three-dimensional partial differential equations) and report that there ". . . is a consensus among the cognoscenti that the best approach to a first attempt at extra-hard scientific problems is a network of hundreds or thousands of fairly general-purpose machines." It is precisely this massive accumulation of general-purpose machines, each doing similar (yet not identical) computations that PAX is designed to manage.

In exchange for the increased complexity of PAX, the user obtains a computational resource that can increase, without practical bound, to meet the requirements of very large computational tasks. A worker is added to PAX simply by increasing the size of the appropriate tables within PAX. Furthermore, workers are, for the purposes of PAX, interchangeable: the work done by one worker can be done by any other worker. Thus, should a worker fail, PAX is able to allocate a replacement worker and continue with the problem.

PAX, as implemented on the Lewis Research Center's UNIVAC 1100/42 computer, has succeeded in demonstrating these capabilities. A logically unified problem (that of airflow through realistic, time-varying flow fields) has been split by the author into a sequence of procedures to be executed asynchronously in parallel. Serial synchronization, where needed, is available. Also, a considerable level of tolerance to random faults in the parallel-processing activities has been demonstrated.

This report presents a technical overview of PAX in an effort to describe what PAX is and what it does in many situations of importance during parallel processing. Technical philosophies and choices are presented without the exhaustive detail of a technical manual.

PAX Overview

Because PAX is a large program (well over 50 000 lines of Fortran) that deals with many complicated concepts, this section gives a "big picture" of PAX and its concepts. The purpose of PAX is to apply many computers simultaneously to a single problem. The problem is broken up by the user for PAX into a series of procedures that follow each other in a step-by-step manner just as in normal computers. However, each procedure is broken up by PAX into pieces that are divided among the available worker processors. A worker proceeds at its own pace through its assigned work and reports to PAX when the work is complete. PAX then assigns it more work.

Usually a procedure contains only one computation. This computation is a specific algorithm performed over a large range of index values. It is this range of values that PAX manages and distributes (in pieces) to various worker computers. For instance, a procedure might be to perform an algorithm over the range 1 to 1 million. PAX breaks that computation into pieces for distribution to workers. One worker may be told to do the algorithm for the range 1 to 100, while the next worker is told to do the algorithm over the range 101 to 200. When the first worker reports that it has completed the range 1 to 100, PAX marks that work as completed and then gives that worker more work from the uncompleted portion of the computation, say the range 201 to 300. PAX continues to distribute work to workers in this fashion until the entire range 1 to 1 million is completed. Then PAX moves on to the next procedure in the problem. Appendix A gives several examples of algorithms that can be executed in this parallel manner under PAX management.

In most problems the workers must share data both as inputs to the computations and as the results of the computation. Something must be done to assure that individual workers do not conflict with each other in their access to shared data. PAX does this by placing restrictions on the use of data by the algorithms and then by careful distribution of the work to be done. In most parallel-processing algorithms the delivery of work by PAX to a worker carries with it the implicit authority to read any necessary inputs and to write to shared storage any generated outputs. No further authorizations are required for a worker to proceed at its own pace on its assigned work.

In some cases a worker may recognize that an output must be made to shared storage that is not allowed by the work it is given (see third example in appendix A). In this event the worker sends a message to PAX indicating the nature of the conflict. PAX then reviews the work that is in progress and the work to be done and schedules the new work so that the necessary output occurs without conflict.

PAX is tolerant of faults. When a worker is given a piece of work to do, PAX estimates the completion time of that work. If the worker does not report back to PAX that the work is done by the time that PAX expects completion, PAX assumes that the worker has crashed. PAX then invokes a user-specified method to recover from the loss of that worker. In most cases the work that was lost need only be given out to some other worker; however, PAX has the ability to discard all work done on the particular procedure in progress at the time of the fault, execute one or more procedures to recover from the possible effects of the fault, and then retry the procedure that experienced the fault.

Fundamental Concepts

Basic Units of Work

Three terms and their interrelationships must be mastered before any understanding of PAX can begin. These terms are algorithm, execution vector, and task.

An algorithm is simply a formula or method for performing work. For instance, the quadratic equation is an algorithm that defines the method for computing the solution of any second-order polynomial equation of one variable. It is very important to see that the quadratic equation provides only the method of solution, not the specifics of the work to be done. A person who understands the quadratic equation still has no work to do because he has no specific job to which to apply it (i.e., no data or parameters).

The execution vector is the counterpart to the algorithm. It is an ordered n-tuple that specifies the particulars of the job to be done but does not supply the method. In the quadratic equation example the execution vector is the polynomial coefficients of the particular equation to be solved. Again, a person with only an execution vector has no work to do because he has no method to apply to his vector.

The fundamental descriptor of work is the task. A task is the combination of an algorithm with an execution vector. This combination provides the worker with a method and a job to which to apply it. No unit of work smaller than this fundamental combination is defined by PAX. In PAX many individual tasks (e.g., many quadratic solution jobs) may be merged into one large task description, which is subsequently referred to as a task.

Nothing more than that discussed immediately above is implied by the words algorithm, execution vector, and task. These words simply define method, specification, and work.

Task Splitting and Associated Algorithmic Constraints

In PAX most tasks describe a large amount of work by describing exactly one algorithm (always) and many execution vectors. PAX splits one such large task into two or more smaller tasks. Each resulting task describes the same algorithm but uses only a subset of the execution vectors. The union of these subsets will always equal the original set of execution vectors so that work will be conserved. What has been achieved is that two (or more) distinct pieces of work now exist where only one had existed before. This ability to split a task allows PAX to hand out large jobs in piecemeal fashion to workers as they become available. (In future implementations of PAX each worker's assignment could be tailored to the specific characteristics of that worker if any distinctions between workers exist.)

The present rules that PAX uses to split tasks provide a fundamental constraint on the structure of the work that can be described: the result (or results) of the work must not depend in any way on the order in which the work is done. If the execution vectors in the quadratic equation example consist of a great many polynomial coefficient groups, the quadratic solutions obtained will not be affected by which polynomial coefficient group is solved first and which is solved last. Although this constraint would appear to be severe, in fact many algorithms of interest are not restricted by it (e.g., most vector operations such as element-by-element addition, subtraction, multiplication, division, and square root). This constraint also implies that no independent output of the work may be an input to the work since no guarantee exists as to the order in which the work is to be done. For example, no quadratic equation root result from one execution vector would be allowed as an element in another execution vector in the task description. However, outputs of the work may be inputs to the work if they occur within the same task at the time of execution. This condition must be checked for by the worker at the time of execution.

The parallel-processing nature of PAX arises from the fact that PAX will split off a task for execution any time a worker processor reports that it is idle. If there is only one worker, only serial processing of work occurs; however, if there are two or more workers, PAX will deliver work to them whenever they are idle. This allows two or more workers to be working on individual pieces of the whole problem at any time.

An additional restriction applies to the work to be done if two or more workers are sharing data (regardless of where the shared storage is located). The storage areas to be written as a result of any two independent execution vectors must not overlap. This restriction is necessary to assure that the final result is not dependent on the order in which the work described by the execution vectors is done. Note that this overlap is considered at the independently writable level. If the outputs do not occupy the same storage, but one cannot be written without writing the other, they overlap for the purposes of this restriction. No constraint is placed on read access to shared storage.

To summarize, the following constraints are imposed on any computation that is to be performed in parallel under PAX:

(1) The computation must consist of exactly one algorithm and a collection of execution vectors.

(2) The result of the computation must not depend on the order of computation (i.e., the order of execution vector delivery).

(3) No output of the computation may be used as an imput to a later stage of the computation unless it is determined (by the worker at the time that the input is required) that the output has already been produced by the same task that is to use it as an input.

(4) If output storage areas are to be shared, the storage areas to be written by any two independent execution vectors must not overlap.

Granularity of Tasks

Some tasks may only be split at specific points in the collection of execution vectors that define the range of work to be done. If this is true, the task is said to be granular in nature since it is composed of groups, or granules, containing several execution vectors that cannot legitimately be separated. (Actually, all tasks are granular; however, the usual granule is exactly one execution vector.) PAX allows the user to specify a granularity for each algorithm that the user defines to PAX. PAX assumes that all granules for a particular algorithm are equal in size and allows the user to specify for each manipulated dimension of the execution vector both the granule size and the starting position.

This recognition of task granularity allows a slight modification of the previously stated rule concerning the overlap of shared output storage areas for independent execution vectors. When task granularity is used, it is necessary only that the output areas shared by any two granules of work not overlap, since PAX will guarantee that execution vectors from the same work granule will never be delivered to two independent workers. Since the worker assigned a particular granule of work will always work with the most recent shared storage information (including any new outputs that the worker has made), one output from the granule cannot accidentally destroy another output from the same granule.

Types of Work

Most work managed by PAX is computational, the results being numbers that are meaningful to the user. Two types of computational work are recognized by PAX: main computation work and conflict resolution work. In the quadratic equation example the roots of each such equation are the meaningful result of the main computational work.

The management of a parallel-processing system requires the definition of a different kind of work to perform management services. These services are for the maintenance of the user's computational environment and the control of the system components by PAX. The control of user-transparent, shared-data access routines and the connection and disconnection of individual machines from the PAX system are examples of this service work.

PAX provides five distinct types of management service work: worker startup, worker initialization (precomputation), worker cleanup (postcomputation), worker hold (unexpected cessation of computation), and worker termination. Because each of these types of work relates to the worker rather than to the computation, each management task created by PAX is identified as being for a particular worker.

The user does not have to concern himself with the creation of management tasks, but only with informing PAX about the management tasks appropriate to a particular calculation. PAX will create the management tasks for each computation at the time that the computation is begun and will create one of each specified management task for each worker that is active at that time.

Description of Larger Quantities of Work

The ultimate description of work is the task; however, a single task description is seldom adequate to define an entire problem. As shown in figure 1, PAX groups tasks into collections called procedures. These procedures are typically made up of one main computational task (defining a large amount of work) and one or more management tasks. The various tasks are sequenced to assure proper operation of management functions. This sequencing assures, for instance, that a worker is initialized for the particular computation before the worker is actually given any computational work. Work proceeds asynchronously in parallel within the procedure. As each worker completes work and becomes idle, PAX delivers the next appropriate task to the worker for execution.

Problems are made up of a sequence of procedures executed in a procedure-by-procedure manner. PAX allows work to be done on only one procedure at a time. In this sense problems are still solved in a serial manner

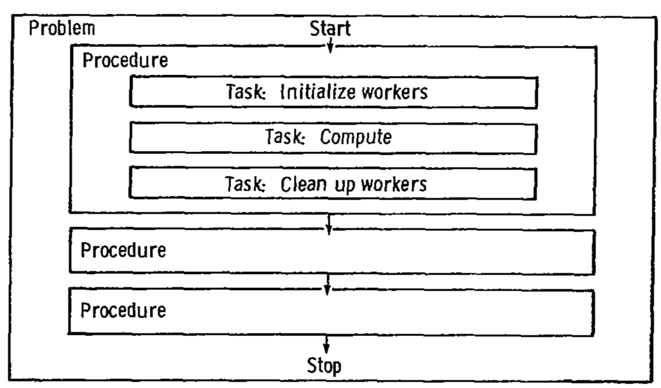

Figure 1. - PAX work description structure.

just as with conventional computers; however, executing the tasks within a procedure in parallel permits much more work to be done in less time. The user customizes each step by defining for PAX the algorithm and the execution vectors to be processed.

Resolution of Conflicts

In dealing with real-world problems the need occasionally arises for a worker to generate an output that is not allowed by the parallel-processing restrictions. Appendix A contains an example of a linked-list-processing algorithm that generates such output conflicts. PAX calls this circumstance a conflict and provides workers with a service for its resolution. When PAX's internal tables are built, the user describes the nature of each conflict that might arise and gives it the necessary details as to acceptable resolution procedures.

When a worker encounters a conflict, it transmits a message to PAX indicating the conflicted work to be done (i.e., an algorithm and an execution vector). PAX uses this information to create a computational task containing the work whose execution might conflict with other tasks already in the system. This task, designated a conflicted task, is scheduled specially to assure that its execution will not interfere with the execution of other tasks.

This scheduling occurs by the method selected by the user from several options available in PAX. The most common selection is one in which the conflicted task is executed only after the completion of the main computational task that contains the point of conflict. PAX extracts the point of conflict from the supplied execution vector and constructs the conflicted task. PAX then inserts the conflicted task into a queue associated with the main computational task that contains the point of conflict. The queue head is actually in the description of the main computational task. Tasks in this queue cannot be released for execution until the main computational task is complete. At task completion PAX checks to see whether any tasks are enqueued in the conflict queue of the task description and, if such tasks are encountered, dequeues them and releases them for execution. Once conflicted tasks are released for execution, they can run in parallel in the same manner as other tasks.

Two conflicted tasks can conflict not only with a main computational task, but also with each other. PAX offers a user-selectable solution to this problem by serializing the execution of conflicted tasks that specify the same point of conflict. When the main computational task is located, PAX will see whether a conflicted task with the same point of conflict is already in the conflict queue of the main task. If so, PAX will queue the new conflicted task onto the completion of the last such conflicted task instead of onto the main task. Each succeeding conflicted task with that conflict point is queued onto the previous one. Thus conflicted tasks with the same point of conflict are released individually for execution upon the completion of the previous task with that conflict point.

Consider as an example of conflict resolution the manipulation in parallel of a large number of linked lists for the purpose of removing elements that are linked into the wrong list and inserting those elements into the right list (where "right" and "wrong" are not important to this example). The execution vectors for the work would be the collection of list identification numbers. After task building, PAX begins handing out work to each idle worker, giving it one or more specific linked lists to process. Each worker receives the implicit authority to manipulate (unlink, link, etc.) each linked list that it is to process; however, it does not receive authority to manipulate other lists since another worker may be manipulating those lists at the same time. Eventually a worker encounters an element that does not belong in the list it is currently processing, and it removes the element from that list. The worker may check to see whether the element belongs in a list that it is allowed to manipulate (by virtue of the list being a part of the assigned task of the worker) and, if so, the worker inserts the task in the correct list. However, if the worker is not allowed to manipulate the correct linked list, a conflict has occurred. The worker sends PAX a message defining the conflict by identifying the algorithm to be performed (linked-list element insertion) and the execution vector specifying the work (linked-list number and element number).

PAX responds to this message by creating a conflicted-task description. Based on its own internal tables (as filled in by the user), PAX determines that the linked-list insertion must occur after the main processing of the target linked list is complete. PAX then locates the task that includes the main processing of the target linked list and checks to see whether a linked-list insertion for the same target list is already queued onto the task. If so, PAX enqueues the newly created insertion task onto the previous insertion task for that particular list; otherwise, PAX enqueues the new task directly onto the main list-processing task. When the main task completes, the insertion task will no longer conflict with it. PAX detects the enqueued insertion task, dequeues it, and releases it for execution. The third example in appendix A explores this linked-list manipulation in more detail.

Worker/Procedure Synchronization

During actual parallel operations PAX is usually unaware of the exact state of the procedure under computation. Specifically the location of the most recent valid copy of shared data is usually unknown to PAX since workers may buffer shared data in their own local memory areas. This is in full accord with the design of PAX; however, at certain times (e.g., the release of a managed worker from further use or the release of a conflicted task for execution) PAX must know the explicit state of the procedure to assure that all necessary components of the procedure and its algorithmic results are properly retained and protected. Thus PAX currently defines a procedure and a worker to be "synchronized" when

(1) The assigned worker has all appropriate background information for executing all pertinent computational tasks

(2) The assigned worker knows the actual location of the most recent valid copy of all required input to any pertinent task

(3) PAX knows the actual location of the most recent valid copy of all generated output of all tasks assigned to the worker

If these conditions are not met, a worker could either proceed on a task with incorrect input data or be detached from PAX while in possession of the only valid copy of output data. Clearly such conditions cannot be accepted. Therefore only when a worker is synchronized with a procedure according to the preceding conditions may that worker begin an assignment or be detached after completing an assignment. Furthermore, if such an illegal transition does occur, PAX detects it and institutes appropriate fault recovery mechanisms to restore the problem to an uncorrupted state.

Exceptional Conditions: Faults and Errors

One of the most difficult problems in computing is responding to the unexpected. In conventional systems exceptional events frequently invoke a response that appears catastrophic from the user's point of view. Often the response is for the entire system to cease operation. The PAX design recognized that unexpected events, such as the failure of an individual worker, would be likely and that a catastrophic response would be unacceptable because of both the anticipated cost of the system and the computations to be performed on it. Thus PAX design includes facilities for the user to specify responses that permit the recovery of his computational product from the most likely failures of the system. Two reporting mechanisms are implemented in PAX: the error and the fault. The error mechanism is for use by the user and is to report algorithmic anomalies that only the user can anticipate and detect. The fault mechanism is used by PAX for reporting unexpected events in the operation of PAX components and reflects those things that are within the (automated) understanding of the management program.

PAX defines an error as an exceptional event that occurs because of the combination of algorithm, execution vector, and input data. An error may require remedial action in a procedure-wide context, possibly including recovery through remedial computation. Thus all PAX computational management facilities are available for servicing errors. Furthermore, because error recovery may have ramifications across the entire procedure, explicit knowledge concerning the appropriate error recovery action must be supplied to PAX for each possible error. The explicit error handling instructions obviate the need for a specific error state. The user must separately identify to PAX any changes in state (e.g., from executable to nonexecutable) that may be associated with a particular error.

The user may include in his code some specific checks on the progress or validity of his computation (e.g., for convergence difficulties or unexpected results). When defining a procedure to the PAX system the user must specify each of the possible errors and the desired response from PAX to each one. Then if an error is detected, the corresponding response instructs PAX to halt, to retry, or to take other appropriate action in an orderly manner.

PAX and its workers report difficulties through the fault mechanism. Faults are exceptional events related to the internal operations of PAX and its workers. Faults are independent of the actual algorithm, execution vector, and input data being executed. The uncontrolled termination of a worker is the most important of all faults recognized by PAX. Because PAX design calls for complete recovery from such faults, PAX requires extensive information from the user (much as for errors) to define acceptable recovery mechanisms for each procedure should a fault occur during the execution of that procedure. The current implementation of PAX detects and recovers from worker-failure faults, and similar methods could be used to recover from other faults possible in actual parallel, asynchronous machines.

Facilities

PAX offers a number of facilities for the control of overall problem computation, for the management of serial and parallel procedural computation, and for the interaction of parallel processors and procedures with their management. Because of the potential cost of terminating computations after obtaining only intermediate results, an extensive facility for suspending operation and making necessary corrections without loss of computational position is also provided.

PAX Control Language

The fundamental facility for computational control is the PAX control stream. This stream of PAX control codes is constructed by the PAX Control Language Assembler, PCLASM. A sample of this language is provided in listing 1. This language is structured like an assembly language. PAX fetches control codes from the stream produced by this language and executes the procedures identified by those codes.

As an example, refer to listing 1, page 3, lines 39 to 41. On line 39 the mnemonic TEDM has been translated to the hexadecimal code 000000010, which tells PAX to enter its dispatching mode. When PAX enters the dispatching mode, it requires more information to identify and specify the parallel procedure to be dispatched to the workers. This information is provided by the mnemonic DVCOR on line 39 as well as as by the mnemonics on lines 40 and 41. DVCOR is translated to a (default) value of hexadecimal 000000006, which corresponds in PAX's internal tables to the algorithm defined in listing 2 (and discussed at greater length in appendix A). Line 40 identifies a single required argument (to be appended to internally generated execution vector components) through an addressing mode code (DASSM, hexadecimal 000000003) and addressing data (VCORF, hexadecimal 000000085, derived by adding hexadecimal control section offset 000000013 from page 2, line 24, and the hexadecimal base address of 000000072 for control section 0003 from page 13). Finally line 41 terminates argument processing with the control code hexadecimal 000000000.

The listing reveals a higher level on which PAX can be viewed. If the reader were not aware of all of the parallel-processing capabilities of PAX, he might deduce that PAX was a simple, step-by-step computer with a very high-level instruction set (e.g., instructions that solve the Navier-Stokes equations, as on page 5, line 6, of listing 1). This is a key observation to understanding the bigger PAX picture because the user can define, in effect, a superinstruction set (i.e., procedures) and use it in a simple step-by-step solution.

The control-code stream facility draws added importance from the fact that it is an integral part of PAX's fault tolerance capabilities. PAX understands the control-code stream structure and is able dynamically to create and insert control-code sequences of its own for error and fault recovery and for certain system management procedures. The error and fault recovery control-code streams for each procedure must be installed in PAX by the user. Such sequences must define, at a minimum, the operations necessary to recover from the unexpected, procedure-asynchronous termination of a worker. Should such an event occur, PAX uses the supplied codes to alter its control code stream and provides the necessary linkages back to the original procedure that experienced the error or fault.

PAX Commands

PAX may receive commands to modify or report its operating state asynchronously with respect to computational operations. PAX commands serve an entirely different purpose from that of the PAX control language. The PAX control language defines the problem, which is independent of the time of execution or of checkpoint and restart occurrences. The PAX commands have no influence at all on the problem. They are concerned solely with events such as checkpointing, stopping, and restarting. Although a variety of different functions are (or might be) served by this facility, its principal use is to direct PAX to an orderly halt. These commands are currently entered through the UNIVAC systems console; however, this is not an architectural constraint of PAX.

Although a considerable number of systems console commands are currently honored by PAX, the following examples should give the reader the general flavor of the facility:

(1) PAX may be ordered to bring parallel computing operations to a close at any time by issuing a STOP console command. This directs PAX to cease the dispatching of further computational work and to perform a complete problem checkpoint-and-exit process when the work that is currently under way completes.

(2) PAX may be ordered to adjust the average running time of tasks split for parallel execution by issuing the

CONFIGURE TASK.TARGET.TIME time.value.pairs

console command. This command directs PAX to change to the indicated value the desired execution time value maintained internally by PAX. When PAX splits off a task for execution, this target execution time is used in conjunction with running-time history tables for the algorithm to estimate how much of the parent task should be split off to make a task of reasonable duration.

(3) The wall clock running time of PAX can be specified by issuing the

SET.RUNTIME time.value.pairs

console command. This command directs PAX to set an internal timer that operates based on wall clock (rather than program execution clock) time. When the time expires, PAX will internally issue a STOP command.

The command facility does not require that command execution proceed immediately to a logical conclusion at the time of initial command execution. A command may suspend itself pending the occurrence of one or more enabling events (e.g., a timer timeout, the return of all workers to idle, or the receipt of a countervailing command). This capability is necessary since the PAX parallel-processing facilities are needed to perform an orderly shutdown of workers. In such a shutdown sequence the change of state inhibits PAX from dispatching any further computational work but allows it to process the completions of outstanding work and to manage the synchronization of workers with the procedure so that those workers can be detached from the problem.

Since command interpretation may be suspended, a command priority structure is provided. This facility allows the PAX system builder to resolve potential conflicts that might occur in interleaved interpretation of commands.

PAX-Worker Interaction Facility

PAX and its workers interact on a dynamic basis by exchanging messages through a shared data area. Currently PAX transmits only one type of message to direct an individual worker to execute a task.

The workers may transmit the following messages to PAX:

(1) The worker is ready to begin task execution.

(2) The worker has successfully completed a task that it was directed to perform.

(3) The worker has encountered an error condition while executing its task.

(4) The worker needs more time to complete its assigned task.

(5) The worker has identified a condition requiring operations outside the limit of its authority and thus requests that PAX manage a task identified in the message to effect these operations.

(6) The worker has identified a change-of-task state. Currently the only defined transition is to a nonexecuting condition.

(7) The worker is on the verge of unconditionally ceasing operation.

A worker is under no constraint in regard to the messages that it can send at any time. Thus PAX is prepared to handle even inconvenient message sequences such as the transmission of a processor termination message in response to a PAX message to perform a computational task.

Error and Event Logging

As might be expected, the debugging of parallel, asynchronous operations can be very challenging. PAX provides an error and event logging facility for the purpose of tracking and diagnosing PAX operational experience. Each error that is detected, whether by PAX or by a worker, is noted and logged. Also, PAX notes and logs a number of significant events and changes of state that occur within its own boundaries. Information defining the precise geneology of each such error or event may, optionally, be recorded in the log entries for enhanced diagnostic use.

Error and Fault Recovery

PAX provides extensive error and fault recovery mechanisms. The entire computational management facilities of PAX are available for this purpose so that parallel, asynchronous computational procedures can be used to recover from errors and faults. Invoking such recovery procedures is optional for errors; however, PAX must be provided with appropriate information for handling PAX system faults. The most likely of these faults is the uncontrolled termination of a worker. Fault recovery options range from simple reassignment of the worker's task to rejection of all computational results from the entire procedure followed by a recovery sequence (of other procedures) and subsequent reexecution of the procedure during which the fault occurred. When each procedure is defined by the user to PAX, information regarding the desired error and fault recovery options must be provided. For example, this information might include a complete computational sequence, potentially involving parallel computations, to reconstruct lost relationships in shared data. Under these circumstances PAX would dynamically insert the supplied control language codes into its own control stream and begin executing them. The end of the recovery-code sequence is made by PAX to restart the computational procedure in which the error or fault occurred. This recovery mechanism can be extended to any practical depth should additional errors or faults be encountered during a recovery sequence.

The error and fault detection and recovery mechanisms keep track of the number of times errors and faults have occurred both in particular tasks and in the procedure. Should errors recur and exceed a preset numerical limit, PAX will bring to an orderly halt all work on the problem and await the user's intervention. PAX does not provide any new solutions to the problem of detecting errors, particularly the infinite loop problem. PAX's error counting mechanisms are intended to limit the spread of such problems rather than to diagnose and correct them; however, future versions of PAX may extend the logic to measure and compare worker productivity in order to detect infinite loops as they execute.

The most probable fault in a real parallel machine is the unexpected failure of a managed processor. As the number of processors increases, the probability of encountering such a failure during the operation of a problem rises, presumably in a linear manner. Because of the high cost anticipated of operating such a machine, it is essential that the PAX design not respond to such events by discarding the computational product produced up to the fault point. Simple checkpointing of previous computational results is a possible alternative, but experience gained in implementing a real parallel problem suggests that such checkpointing requires more time and resources than do recovery methods based on the true needs of individual procedures.

Checkpoint and Restart Facility

PAX offers its own checkpoint and restart facility because a number of independent but logically unified processes may be executing under PAX at any time. The checkpoint sequence occurs whenever PAX is ordered to halt. Such an order may be delivered to PAX from the UNIVAC systems console, from the PAX control language stream, or from within PAX itself.

The checkpoint and restart facility separates problem-specific information (i.e., information that describes the current state of the problem work to be done) from code and data relating to the management and operation of PAX and its workers. All PAX starts begin by loading the problem-specific data from a known, permanent place. Data relating solely to PAX's internal operations and arrangement are not loaded from any checkpoint file but are, instead, accepted as supplied in PAX's own program load image.

This selective reloading of data during the PAX start sequence allows PAX to be highly tolerant of alterations, particularly to its own code and that of its workers. In this way bugs can be corrected without loss of position in a current problem. Additionally careful adjustments to the current problem state or the data base supporting such a problem can be made while PAX is halted without loss of position in the problem.

Architecture

The following discussion details architectural points of PAX as it is simulated on the Lewis Research Center's UNIVAC 1100/42 system. Although the current implementation is not intended for a real parallel, asynchronous machine system, most of the organizational aspects will still apply in a real system.

The current PAX implementation is constrained by the fact that PAX has no authority regarding the allocation of resources within its host environment. In particular, worker components can be placed temporarily in a nonexecutable state by UNIVAC's EXEC VIII operating system without the knowledge of PAX. This situation causes difficulties in that PAX misinterprets the absence of activity from the worker to be an unscheduled termination rather than a temporary suspension of that worker.

Labor-Management Architecture

The principal architectural division in PAX is the labor-management division. The management function (i.e., the definition, direction, interaction, and management of a problem) is contained within the formal boundary of PAX (fig. 2). All parallel, asynchronous computation is performed by the workers. PAX and its workers are connected by a communications facility through which messages can be passed to direct the actions of the workers and to report the results of such action and the status of the workers.

The architecture also defines an access path for PAX and all of the workers to a shared source of data. This shared source of data is optional since some meaningful parallel-processing problems do not require shared data. These problems are usually not input data intensive.

Some serial computation is performed within the formal boundaries of PAX. This architecture simplifies internal PAX design and is appropriate when PAX is a single-user system. When multiuser architecture is approached, this concept may well be revised since PAX would be likely to have more pressing management duties that would be given precedence over the execution of serial tasks for a particular user.

PAX Management Architecture

PAX has six internal components (fig. 3):

(1) The shared executive-data area (EXDA) is the internal binding among the other five components of PAX. All data defining the current operating state of PAX and the current state of the computational problem under consideration are contained in the EXDA. Also, all internal communications between PAX components are routed through the EXDA.

(2) The overall manager (OM) provides all basic management decisions and directions.

(3) The external listener (EL) waits for messages from workers or other software entities that have access to the (PAX) interprocessor communications path. When such messages are received, the EL performs some error checking and message transformation and queues an appropriate message to the OM.

(4) The anticoma activity (AC) serves as a timer for PAX. It periodically scans the expected completion times of any outstanding work in the PAX system and notifies the OM of any overdue events. This activity prevents the

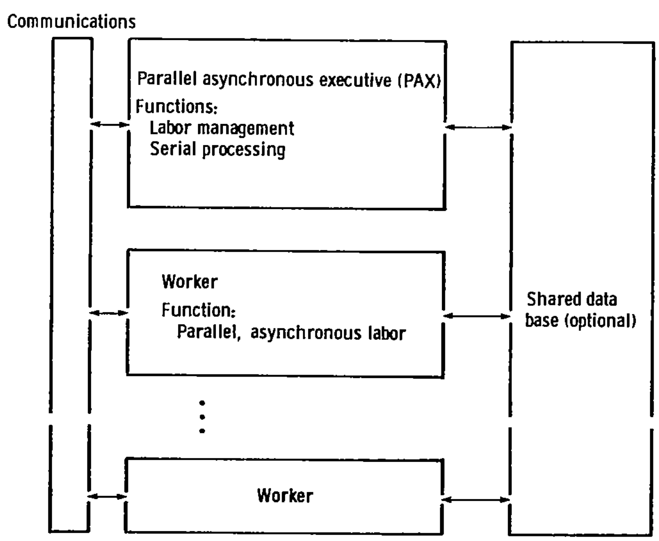

Figure 2. - PAX labor-management architecture.

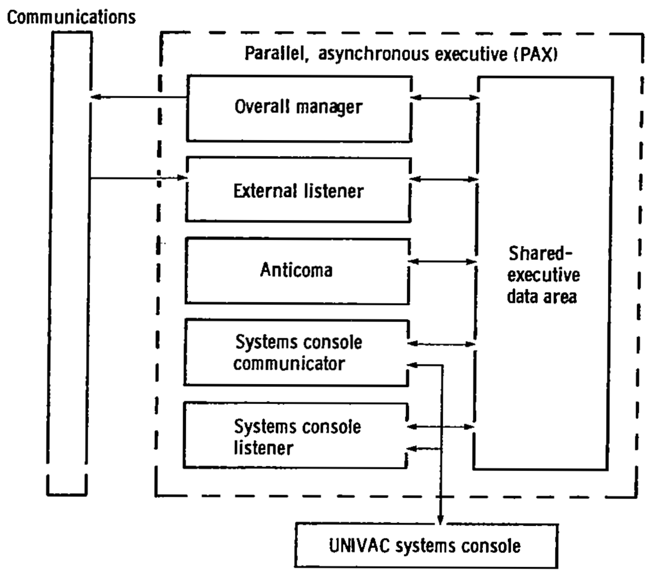

Figure 3. - PAX management architecture.

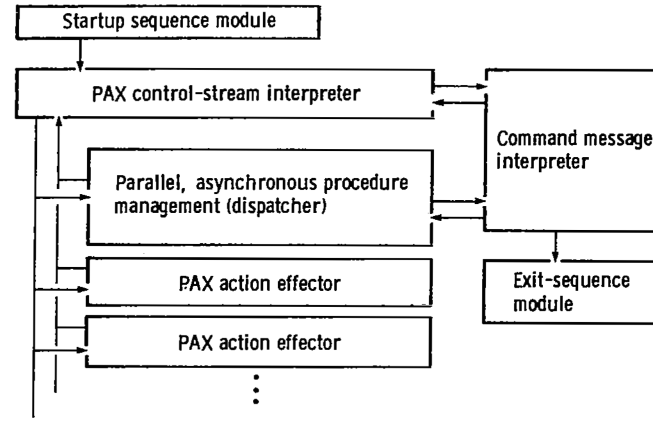

Figure 4. - PAX overall manager architecture.

OM from drifting off into a comatose state in the event that all of the workers fail (e.g., an unexpected infinite loop occurs in a parallel procedure).

(5) The Systems Console Communicator (CC) provides an error checking and message translating intermediary between the OM and the UNIVAC systems console. Full bidirectional conversations initiated by either party may be carried on between the OM and the UNIVAC systems console.

(6) The Systems Console Listener (CL) waits for an indication from the UNIVAC systems console that it desires a conversation with the OM. In this event the CL so informs the OM, which then responds through the CC.

Overall Manager Architecture

The internal arrangement of the OM is depicted in figure 4. After the PAX startup sequence has completed, control passes to the PAX control-stream interpreter. This interpreter fetches the codes produced by the PAX Control Language Assembler (or dynamically created by PAX itself) and directs control to an appropriate PAX action effector. A specific action effector is dedicated to each PAX control code and is responsible for carrying out the desired action. Between control-code fetches, the control stream interpreter checks to see whether any PAX command messages are waiting. If such a message is waiting, control is diverted to the PAX Command Message Interpreter (CMI) to process the command. Normally, control returns then to the control-stream interpreter; however, on the appropriate command, control may pass to the exit sequence module from which a normal exit occurs.

From a control-stream context PAX can be viewed as a virtual machine. The control codes supplied in the stream designate actions to be performed by the PAX virtual machine, each action being completed before the next is begun. Some actions performed by the PAX virtual machine are procedures that are split into segments that operate in parallel; however, in the control-stream sense, they still appear as single actions designated by a single code. Thus a parallel, asynchronous procedure does not differ from any other action when considered from the control-stream perspective; however, internally the parallel, asynchronous procedure management action effector (also referred to as the dispatcher) is very different from other action effectors. The principal difference is that it checks for the presence of command messages and, if such a message is present, transfers control to the CMI. Upon completion (or suspension) of message interpretation, control transfers back to the dispatcher. The other principal difference is that the dispatcher's actions consist not of computation but of message generation, receipt, and processing.

Parallel, Asynchronous Procedure Management Architecture

Figure 5 depicts the general organization of the dispatcher portion of the OM. Upon transfer of control to the dispatcher an initialization sequence is performed (1) to establish the status of each authorized worker and (2) to construct the necessary internal task descriptions to effect the requested parallel procedure.

Once initialization is complete, a specific process of handling messages and dispatching work is begun. The priority of dispatcher attention is as follows:

(1) Any waiting command message is interpreted by a temporary transfer of control to the CMI.

(2) Any messages received from workers are handled by an internal segment of the dispatcher.

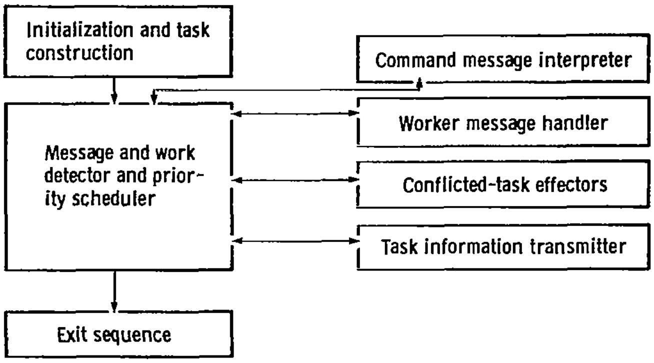

Figure 5. - Parallel, asynchronous procedure management architecture (dispatcher).

(3) Any read-to-run conflicted tasks that, instead of being distributed to workers, are to be executed by PAX are executed in an internal segment of the dispatcher.

(4) If ready-to-execute parallel tasks and idle workers exist, appropriate task execution messages are made up and transmitted to the workers by an internal segment of the dispatcher.

If none of these conditions exist, the dispatcher issues an activity suspension request on behalf of the OM and awaits the arrival of either a command or a parallel processor message.

The dispatcher action effector also offers an alternative initialization sequence, which allows reentry of a suspended parallel procedure. This initialization skips the problem-related task-building operations and, instead, simply accepts the task descriptions already in the various PAX task queues. Parallel processor management functions and maintenance-task building proceed normally in this situation. This architectural feature allows PAX to suspend parallel operations in midprocedure and to resume those operations at a later time. This ability is necessary to satisfy checkpoint/stop requests (on command or on internal error) in a timely manner.

As noted in item 4 in the list of priorities, the dispatcher is responsible for matching waiting tasks to available workers and transmitting appropriate messages to such workers to effect the tasks. To perform this action, the dispatcher splits such tasks (if possible) into tasks of manageable size. The dispatcher maintains tables in the EXDA for use in establishing the number of execution vectors that will lead to a task of reasonable duration.

The response of the dispatcher to errors and faults arising from executing tasks is important to the overall success of PAX. The following options are available to the dispatcher, one of which must be selected by the user (currently, at PAX build time) for each dispatchable task:

(1) PAX may be ordered to checkpoint and stop immediately.

(2) The error may be noted and ignored. Faults (e.g., the unconditional termination of a processor that is unsynchronized with the problem) may not be ignored.

(3) The task generating the error or fault may be placed in the waiting task queue for reexecution by the next available appropriate worker.

(4) The entire procedure generating the error or fault may be reexecuted.

(5) The procedure generating the error or fault may be discarded in the most expeditious manner possible. Then a user-specified series of procedures may be inserted into the PAX control stream and executed in order to perform such remedial actions as are necessary to return the problem to a known state. Upon successful completion of the reconstruction, control will transfer to the faulting procedure, which will be freshly initialized and executed.

PAX maintains statistics on the occurrence of errors (on a task basis) and faults (on a processor basis) and does not allow limitless repetition of errors or faults. Repeated errors from a particular task will eventually force a checkpoint and stop of the problem. Repeated faults from a particular processor will cause PAX to remove that processor from use and deliver it to an architecturally defined (but not currently implemented) maintenance facility. If PAX removes such a processor from use, it will attempt to obtain a replacement and, in any event, will continue on with the problem with whatever resources remain. If all parallel processor resources are exhausted, PAX will checkpoint and stop the problem and itself.

Worker Architecture

The architecture of a PAX worker is shown in figure 6. (Note that "worker" is used here in a conceptual sense

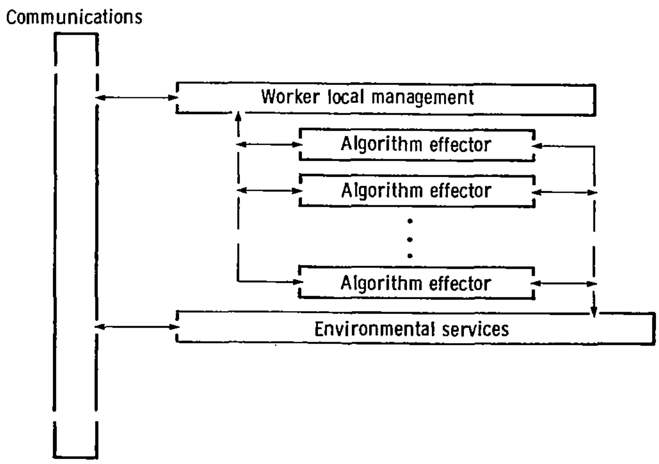

Figure 6. - Worker architecture.

and, for PAX's purposes, may mean one of many processes on an individual worker computer.) A worker is controlled by a simple management program that receives and transmits messages and transfers control to algorithm effectors. The algorithm effectors periodically transfer control to a progress estimator (an environmental service) that may transmit a request for more execution time to PAX if necessary. Several other services are available to algorithm effectors for the transmission of other requests to PAX.

The worker cycle is simply this:

(1) The worker receives a message to execute a task.

(2) The worker executes the appropriate algorithm as specified by the supplied execution vector (or vectors).

(3) Various PAX facility requests (for conflicted tasks, etc.) are transmitted to PAX as appropriate.

(4) A task completion message is transmitted to PAX on completion of algorithm execution.

No ability to query the worker during task execution is defined within PAX architecture. This relieves PAX of the burden of periodically querying a (potentially) very large number of workers and simplifies worker design and implementation; however, it also means that fault detection must become a passive process since PAX cannot query a supposedly busy worker to determine its progress or its health. This architecture could be changed in future implementations. Current experience shows that an algorithm with an infinite loop can easily consume all available PAX system resources through the passive fault detection mechanism. The mechanism is as follows:

(1) After a reasonable period of time, PAX declares the worker executing the infinite loop to have faulted.

(2) PAX institutes the appropriate recovery procedures, including the addition of a replacement worker. Eventually, the task containing the infinite loop is assigned to another worker.

(3) While the worker that was originally assigned the task containing the infinite loop continues to work diligently at its assigned task, a second worker attempts to execute the infinite loop and is eventually faulted by PAX.

(4) Steps 1 to 3 repeat until all workers are executing the infinite loop and PAX is halted for lack of worker resources.

As can be seen, the addition of some sort of asynchronous query facility is highly desirable.

Concluding Remarks

A software operating system (PAX) has been developed to demonstrate the feasibility of applying many independent processors to a single, logically unified problem. Results indicate that a real parallel, asynchronous processing system can be defined, implemented, and brought to bear on large computational problems. This system will allow the man-month rule to apply to a wide range of computational problems that fall within the restrictions set forth in this report. Thus a problem (operating under this system) that could be solved in 2 months by 20 computers might be solved in 2 days by 600 computers. This man-month rule may be followed without practical engineering limit.

PAX has achieved the following:

(1) Applied several computing processes simultaneously to a single, logically unified problem (CASPER)

(2) Resolved most parallel-processor conflicts by careful work assignment

(3) Resolved by means of worker requests to PAX any conflicts not resolved by work assignment

(4) Provided fault isolation and recovery mechanisms to meet the problems of an actual parallel, asynchronous processing machine

As with all such research efforts, much work remains to be done (as delineated in appendix B). The limitations of the reported work are the result of imperfect vision during the design phase and do not represent long-term imperfections of the overall concept. The reported work is a solid base of learning from which a second generation of parallel, asynchronous process management can be designed and implemented for a truly parallel, asynchronous machine.

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio, February 28, 1983

Reference

1. Haynes, L. S.; et al.: A Survey of Highly Parallel Computing. Computer, vol. 15, no. 1, Jan. 1982, pp. 9-24.

Appendix A
Parallel-Process Examples

PAX was designed and implemented in response to the needs of CASPER (Combined Aerodynamic Structural Dynamic Problem Emulation Routines), a method for simulating the unsteady viscous flow of air through real, time-varying flow fields. CASPER simulates such airflow by creating a vast population of Lagrangian aerodynamic elements. It applies various algorithms to known properties of an aeroelement to calculate other properties for it. For instance, the velocity and position of an aeroelement and its nearest neighbor elements are used to establish the velocity field gradients for the aeroelement. Then (in a subsequent computational procedure) these velocity gradients can be used with the Navier-Stokes equation to produce aeroelement accelerations. Although this report does not offer a detailed exploration of the mathematics and methods of CASPER, it provides some examples of parallel processing as applied to CASPER.

Simple Parallel Process

In CASPER, the volume of each aeroelement is estimated on the basis of its proximity to each of its nearest neighbor aeroelements. This estimate is not necessarily accurate in an absolute sense, but it is consistent on an element-to-element basis. During individual volume estimation a running total of all such individual volumes is maintained. By comparing the final total of individual aeroelement volumes to the actual volume known to be occupied by all aeroelements, a multiplicative correction factor can be obtained and applied to each aeroelement.

Listing 2 illustrates how the volume correction factor can be applied in a simple parallel, asynchronous process. The subroutine VCOR multiplies each aeroelement volume by the correction factor and places the result in a scratch location associated with the aeroelement. The estimated volume of each aeroelement is obtained from a shared data area through a call to the Fortran function V (line 24 of listing 2). Write access to each aeroelement's scratch location is through the Fortran subroutine STAESC (line 25 of listing 2). The correction factor is supplied to VCOR as the subroutine argument VL. The algorithm's execution vector is the aeroelement identification (the DO-LOOP index of line 22) and the volume correction factor VL.

PAX delivers many individual execution vectors to each worker executing this subroutine by supplying a range of aeroelement identifications (IL to IH, supplied in the argument list) and a single correction factor VL, shared by all aeroelements. This arrangement is typical of execution vector manipulations by PAX. Many execution vectors contain components that do not vary from task to task and thus are ignored in work scheduling. PAX distributes work according to the parts of an execution vector that distinguish a specific piece of work from all other pieces of work. These components of the execution vector are manipulated as ranges of values rather than as individual values.

This example is simple, but it illustrates the advisability of input and output segregation in algorithm design for parallel processing. In VCOR the corrected volume result is placed in a scratch location for later (post-parallel-procedure) use in a subsequent aeroelement volume update procedure. The alternative would have been to make in-place correction of each aeroelement's volume. The selected approach has the distinct advantage that, should an unsynchronized worker failure occur during this procedure, the shared data base can be recovered merely by reexecuting the parallel task (or tasks) placed under suspicion by that failure since the input data can safely be assumed to be uncorrupted. If the algorithm had stored the corrected volume result back into the shared volume location for the aeroelement, such a failure would have left PAX uncertain as to the state (corrected or uncorrected) of each suspect volume. In such an event recovery procedures would have to include the reestimation of aeroelement volumes from other uncorrupted data.

Parallel Process with a Conditional Algorithm

Listing 3 illustrates a parallel process that requires conditional branches within the algorithm. Subroutine MOVEL moves aeroelements through space by integrating velocity and acceleration, subject to the constraint that no physical boundary shall be violated. The conditional branch occurs when a boundary is violated. In this event the algorithm must locate the point of violation and provide an alteration of course at that point. Not all aeroelements will require such an alteration, nor will the same boundaries affect each aeroelement (whether or not a violation occurs).

CASPER supplies this algorithm with an initial position (line 192, function reference X) and velocity (line 193, function reference U) for each aeroelement, as well as an acceleration (line 194, function reference A). The acceleration is previously calculated with the Navier-Stokes equation and is presumed to be constant for the time period over which the positions are to be calculated. CASPER describes real shapes as a concatenation of truncated functions F of space and time. A boundary is the locus of all points such that F is zero. The volume contained by such a boundary is the locus of all points in space-time such that F is less than zero. To reduce computational load, CASPER identifies zones in space (in this example through the subroutine call TSTZN at line 334) for which particular subsets (identified through function references IPZBL and ZBL, lines 338, 340, and 343) of the concatenation of functions F apply. Thus the need to check positions in space-time against all functions in the concatenation is eliminated.

The key conditional branch occurs at line 208 of the listing. The internal subroutine YNM has just returned in the variable SVMIN the smallest surface function value for the appropriate subset of the functions F at the current position of the aeroelement in space-time. If the value of SVMIN is zero or negative, a boundary violation is occurring at that point in space-time and corrective action (beginning at line 250 of listing 3) must be taken. This corrective action consists of (1) identifying the point in space-time just short of boundary violation for use in the next normal boundary violation test (lines 206 to 208) and (2) setting a flag to indicate that a boundary bounce operation must occur if the normal boundary violation test shows no violation. The subroutine inspects the boundary bounce flag at line 209 of listing 3 and, if so directed, applies an angle-of-incidence-equals-angle-of-reflection rule to the aeroelement's path by adjusting the aeroelement's velocity vector (lines 210 to 230, especially line 224, listing 3). The angle-of-incidence-equals-angle-of-reflection rule is given as a first approximation to aeroelement behavior, but almost any other rule could easily be inserted in this code.

It is important to note the highly conditional and (from the algorithmic design viewpoint) unpredictable nature of this parallel process. Each aeroelement is checked against only a subset of the boundary functions, and the subset may change in midflight for any particular aeroelement. An aeroelement may or may not violate one or more of the boundary functions, and a course modification code must be applied only if such a violation occurs. Parallel processing is ideal for handling these conditional clauses because the algorithm is executed independently for each aeroelement by a traditional serial machine in which conditional branches do not carry any particular penalty. The power of parallel processing arises from the fact that this algorithm can be split by aeroelement identification (ID) range (i.e., worker N sets aeroelement ID's running from ILn to IHn while worker M sets aeroelement ID's running from ILm to IHm, etc.) into many tasks to run on many individual machines. Such splitting is possible because the inputs (aeroelement initial position, initial velocity, and acceleration) are segregated from the outputs (aeroelement final position and final velocity, which are both placed in aeroelement scratch locations) and because shared outputs (aeroelement scratch locations) are mapped on a one-to-one basis by the execution vector (aeroelement ID's).

Parallel-Process-Generated, Shared-Access Conflicts

Listing 4 illustrates a parallel process that generates shared-data-access conflicts. CASPER maintains a linked list for each flow zone of all of the aeroelements that are actually resident in that flow zone. As aeroelements move through space, they may move to another flow zone. Thus CASPER must periodically search through each flow zone list to assure that it contains only aeroelements that are actually resident in that flow zone. The purposes of the subroutine RESO2 are (1) to search through the linked list of each flow zone in the range IZL to IZH for aeroelements that do not reside in that flow zone, (2) to remove each offending aeroelement from that list, and (3) to link each such aeroelement into the list of the proper flow zone. The need for PAX conflict resolution services arises from the fact that PAX grants authority to the worker to manipulate lists only in the assigned range IZL to IZH. Although this authority is sufficient to allow an individual worker to remove an offending aeroelement from a list it is searching, it does not necessarily permit the worker to place that aeroelement in the correct list since that list may lie outside the range IZL to IZH.

To link an offending aeroelement into its correct list, the subroutine first checks to see whether the targeted list is within its range of authority (lines 165 and 166). If so, relinking proceeds without communication with PAX; otherwise the aeroelement is linked into a local list for later transmission to PAX in a conflicted-task request. To reduce computational load, these local lists are maintained by target list number so that PAX will not have to perform any further sorting. Also, these local lists are held until either (1) the parallel process comes to an end and must report completion to PAX or (2) no more local list room is available and a new list must be accommodated. Lines 184 to 187 are associated with the former condition, with line 186 invoking the tether (local list) flush subroutine TETHF.

The tether flush routine (listing 5) illustrates the conflicted-task request procedure. The target list number (flow zone ID in variable J, lines 31 and 35) and first aeroelement ID in the local list (variable I, lines 29 and 34) are passed to the PAX conflicted-task request routine, REQSAF, on a stack that also contains appropriate argument control codes. The requester's ID (parameter OURID) and request number (literal argument to REQSAF) are provided in the actual call on line 47 to the request subroutine. REQSAF, a worker environmental service (fig. 6), provides the interface to the PAX/parallel processor communications facility by constructing and transmitting the appropriate message to PAX. A shared-data-base flush of local buffers must precede the call to the request routine, to assure that PAX will be able to access the most recent information placed by the executing process in various aeroelement linkage slots. Also, listing 4 and 5 do not show that storage locations associated with the first aeroelement in each local list contain the ID number of the last aeroelement and the number of aeroelements in the local list. This information is needed to execute the conflicted task.

The example of listing 4 illustrates the need for the PAX parallel-process fault recovery features. Consider what would happen if that parallel process should unexpectedly terminate (e.g., by a hardware failure) while sorting through lists as directed. In this event some offending aeroelements might remain linked in local lists with no reference to them from any of the shared lists. Alternatively, if the termination occurred during the unlinking or relinking (lines 152 to 154 and lines 167 to 177, respectively) of an aeroelement, the integrity of the shared list would be compromised. Clearly such difficulties cannot be corrected by simply rerunning the process on another processor.

In response to this, PAX offers its extensive recovery capabilities. In this case the choice was to reconstruct the shared linked lists to assure list integrity and completeness, by discarding the work of the existing parallel procedure and instituting a new parallel procedure. The reconstruction procedure links every aeroelement into some legal shared linked list without regard to the correctness of the selected list. This reestablishes the integrity of the shared data structure so that the parallel sorting procedure will produce correct results when it is subsequently reexecuted. In this way the computational product managed by PAX can be preserved despite the otherwise catastrophic failure of one or more of PAX's managed components.

Appendix B
Suggestions for Further Work

As with most research projects, more work remains to be done. This initial exploration has suggested a number of possible improvements to current PAX design that would facilitate its use for a real parallel, asynchronous processing machine. These improvements—adjustments to existing PAX software strategies and desirable selections for PAX hardware environments—are discussed in this appendix.

Software Improvements

Initial PAX design did not account for parallel shared-data storage (i.e., the storing of logically related data across many mass storage units), nor did it provide for recovery from mass-storage-unit failures. Since future implementations will undoubtedly require such parallel storage, fault recovery schemes must be defined for the failure of individual mass-storage units. Recovery procedures for mass-storage-unit failure would be specified by the user in a manner similar to that for processor unit failure. It would be desirable not to burden the user with the problem of fielding shared-data-access failures. Thus one (or more) layers of shared-data-access services, including the ability to identify and report to PAX such data access failures, must be provided in the PAX system environment.

Intelligent shared-data-base controllers might be desirable to field requests from workers for data access. These controllers could add two valuable design features. First, they could handle data-base-unit failure as mentioned in the preceding paragraph. Second, they could provide a dynamic redirection facility to the shared data base to ease the local buffer flushing loads that may be encountered in an improved system. This feature might work by having each data requester inform the controller if the data access is to include data modification rights. If this is the case, the shared-data-base controller could redirect subsequent requests for the particular data directly to the controller for the local buffer of the most recent (potentially) modifying requester. Thus the last processor to modify the data would transmit that data directly to the new requester, saving the intermediate transmission to the shared-data-base controller. Care must be taken to account for the fact that the new requester may also be a modifying requester. Also, it is possible that in some cases a data request message might not represent a sufficiently smaller transmission load than the requested data itself. If so, a shared-data-base controller might well be a needless complication.

As mentioned in the main body of this report, an asynchronous worker status facility would be useful to avoid long latencies by PAX in assessing the health of a particular worker. Since PAX would presumably be implemented as a superexecutive over existing operating systems, it should not be difficult to provide a mechanism for the local machine operating system to report the operating statistics for a particular process. The consumption by a particular process of system resources (memory, processor, and input/output) should be a reasonable first measure in determining process health. The local operating system could also report any process state transitions (e.g., from "competing for resources" to "blocked for lack of local resources") to PAX in order to eliminate unnecessary health queries and erroneous health determinations by PAX.

The definition of a worker "personality" may be advisable to allow PAX to manage nonhomogeneous parallel processors (or, more easily, a family of computers with identical architecture but differing in computational speed). This ability would be especially useful when massive parallel-processing facilities are not affordable on a full-time basis. An organization with occasional need for such supercomputing may be able to get it by using the computing power that it normally applies to other needs, such as shop management, accounting, business computation, and office automation. Although computers currently in place may not be entirely appropriate for management by PAX, a family of computers might be selected that would serve well both as PAX workers and as computers for various other needs.

Finally PAX capabilities were limited unnecessarily in this version by the decision to make PAX a single-problem environment. The next PAX design should allow more than one parallel problem to be managed and executed concurrently in order to increase and even out the utilization of the entire conglomerated machine. Although a single parallel problem could keep each parallel processor busy if several logical workers are assigned to it, periods of severe inactivity may be expected as the problem goes through changes of state, either in an internal sense (e.g., extensive serial operations for crucial problem-management decisions or for fault recovery) or in an external sense (e.g., being checkpointed). Thus having several parallel problem streams in progress would be desirable to fill in the gaps.

Certain advantages would be available in exchange for the increased complexity of the multiproblem architecture. The health and characteristics of processes in one problem may provide information useful in determining the health of processes in another problem. For instance, if a process in problem A is overdue for completion when a process in problem B running on the same physical machine has completed in record time, PAX could conclude either (1) that the A process is healthy but has been squeezed out of its share of the machine resources by the B process or (2) that the B process demonstrates that the physical machine is healthy but the A process either is looping or has crashed.

Beyond these conceptual adjustments to PAX, a number of practical concerns should be considered in future designs. These include the management, maintenance, and alternative utilization of the large number of machines that would be associated with a real PAX implementation. A facility for PAX to turn a suspect machine and appropriate symptom messages over to a diagnostic and maintenance complex could be valuable because of the large number of machines that might be used by PAX. Furthermore it might be financially desirable for PAX to be able to release an operator-selected machine from parallel-processing duties for use in other operations (e.g., to operate a test facility or to provide business processing services during normal business hours). Another useful feature might be a dynamically specified limit on the level of parallel-processing activity for a particular machine, so that machines that are not fully utilized for some other necessary activity such as word processing may simultaneously participate in parallel-processing problems.

Hardware Improvements

It is the author's view that PAX will require much less hardware development than most other supercomputer schemes. Indeed a principal goal of any PAX implementation should be to keep hardware components straightforward, reliable, and inexpensive and thus avoid the difficulties of ultra-high-performance electronics usually associated with supercomputers. Off-the-shelf computer components appropriate for a PAX implementation are now available in quantity at relatively low cost. The author believes that an entirely satisfactory PAX implementation could be produced with off-the-shelf components currently in production by any of several manufacturers.

A thoughtful review of the concepts outlined in this report should convince the reader that the most difficult hardware problem will be communications. In particular, for shared-data-intensive problems the communications link between the workers and the mass-storage units will be the pace-setting path, since all data to be used must filter through the data-access communications path. Thus the performance of the communications link must be matched to the performance of the mass-storage units, with due consideration given to the relative shared-data intensity of the problems to be solved.

Communications hardware is available off the shelf that approximates the performance of some midrange mass-storage units (1 million to 10 million bits/sec). Higher performance communications options are available; however, such hardware may leave the developer spending more for communications units than for the mass-storage and processing units that are being linked together. Some manufacturers are beginning to offer communications hardware using fiber-optic technology that may considerably improve this situation and allow the effective use of high-performance disks in shared-data-intensive problems.

Careful PAX implementation can render the resulting software product relatively insensitive to future improvements and upgrades in communications technology. A natural dividing line in PAX design occurs between PAX and its communications services. Thus future improvements in communications technology can be incorporated into the hardware with minimal software difficulty.

Aside from communications technology the communications speed problem can also be approached from the context of communications topology. Each candidate topology offers a trade-off between communications equipment cost and communications speed. This subject has been treated in great detail elsewhere and need not be explored here. It is sufficient to note that, again, careful design can make PAX insensitive to communications topology so that PAX implementations can be tailored to meet the requirements of particular parallel problems. With the topological tailoring approach, useful PAX systems should be configurable with off-the-shelf hardware out to economic limits determined by the trade-off between performance and cost.

The selection of a computing unit for a PAX implementation is less critical than the definition of communications methods; however, implementation will be easier if certain features are provided. First, the candidate machine should have a large address space, at least 2^{32} bytes. The existing PAX software is large and will certainly expand in any new implementation. Furthermore a great deal of information must be maintained on a dynamic basis to define the current state of a parallel problem. The amount of this information will grow as more worker processors are added to a PAX implementation since separate information must be maintained about each parallel process that is in execution. Additionally, certain PAX management schemes may retain information beyond the minimum necessary for parallel-process management (e.g., the exact history associated with each task of a parallel procedure). All of this could combine to increase the size of PAX significantly. Thus any candidate machine must facilitate the use of such large amounts of information.

The accessing of large amounts of data by workers and the distribution of that data across many physical storage units also dictate that the selected computing unit provide some means of translating a user data reference by index number (e.g., by a reference in the manner of a Fortran array) into the necessary information to locate and retrieve that data from its shared-storage location. The author is unaware of any machine that offers such a feature as a standard part of its operation; however, a number of machines provide user-writable control stores in their processors. With such a feature a machine instruction might be devised (along with appropriate data structures) to facilitate such a translation of information. In particular, machines that implement a virtual addressing feature and offer a writable control store would be highly desirable since presumably they would have the hardware necessary to ease the translation from an index group through a logical address to a physical or mass-storage location. This feature becomes more important as a problem becomes more shared-data intensive. The author's experience with the aerodynamics computations suggests that the address translation feature is very important.

Another key point in selecting a PAX worker machine is the longevity of its architecture. The development of PAX software for a real system will be a large project. It would be unfortunate if, as PAX reached practical application, the selected machine disappeared from the marketplace because its architecture was out of date. It would also be undesirable if PAX were forced into unending rewrites to use features of an expanding architecture. Therefore one should select an architecture that is not expected to grow, having started out with all of the appropriate features to make a good, flexible, fully integrated computer system. Only the capabilities of the machines designed to the architecture should grow, for example, in terms of either increased speed or decreased physical size. Architectural stability will allow PAX to use the latest technology without extensive software changes.

Final considerations here in selecting a computing unit are its reliability and maintainability. PAX design recognizes the inevitability of worker failures, especially within a large community of machines. Although PAX can accommodate these failures without catastrophic results, too many such failures would set a premature limit on the expansion size of the system when it spent more time accommodating failures than computing useful results. Furthermore worker downtime would be minimized if most machine problems could be identified automatically by some maintenance complex associated with PAX. The computing unit should thus have some capabilities for self-diagnosis and remote diagnosis. These features are available to varying degrees on some machines on the market today. Although this diagnosis feature is not required by PAX design, it strongly affects the practicality of maintaining a parallel-processing machine.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978            PAGE       1


;+
; PAX-CASPER CONTROL LANGUAGE
;
; F100 DUCT WORK AIRFLOW
;
;       AUTHOR  WILLIAM HENRY JONES
;       X01-00  19 FEB 81
;-
```

Listing 1. - PAX control language stream.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978           PAGE      2

000C                                           .PSECT $DATA, D, RW, LCL, REL, CON
000D                                   ;+
000E                                   ; WORKING CONTROL DATA
000F                                   ;-
0010  000000000  000000001             CTRL1:  .WORD   1         ; RELOCATION SUBROUTINES TO NORMAL MODE
0011  000000001  000000004             CTRL2:  .WORD   4         ; SORT EXHAUST AND UN-USED TO INLET WITH XREF
0012  000000002  000000005             CNTR1:  .WORD   5         ; RELOCATION COUNTER
0013  000000003  000000005             CNTR2:  .WORD   5         ; RELOCATION LOOP COUNT
0014  000000004  000000003             FZIDL:  .WORD   3         ; FOLLOWING ID LIST CAN ALSO BE A 3 WORD STRING
0015  000000005  000000003             IDI:    .WORD   3         ; INLET ZONE ID
0016  000000006  000000004             IDE:    .WORD   4         ; EXHAUST ZONE ID
0017  000000007  000000005             IDU:    .WORD   5         ; UNUSED ZONE ID
0018  000000008  000010000             ISIZE:  .WORD   65536     ; NUMBER OF AEROELEMENTS
0019  000000009  000000052             NZN:    .WORD   82        ; NUMBER OF FLOW ZONES
001A  00000000A  39E8DB8B7             GDAST:  .FLT    %D0.0001 ; TIME INCREMENT
001B  00000000B  C61724748             MGDAST: .FLT    -%D0.0001 ; MINUS TIME INCREMENT
001C  00000000C  40C000000             CURTIM: .FLT    1.0       ; CURRENT TIME
001D  00000000D  00000000A             NINC:   .WORD   10        ; NUMBER OF TIME SUB-INCREMENTS
001E  00000000E  000000000             PVHRBP: .WORD   0         ; P-V HISTORY POINTER
001F  00000000F  40C000000             PAV:    .FLT    1.0       ; PREVIOUS SOLID ANGLE AVERAGE
0020  000000010  000000000             OMEGA:  .FLT    0.0       ; SOLID ANGLE ACCUMULATOR
0021  000000011  000000000             VOLA:   .FLT    0.0       ; AEROELEMENT VOLUME ACCUMULATOR
0022  000000012  49D460000             VREST:  .FLT    345600.0 ; VOLUME OF PROBLEM LESS INLET AND EXHAUST
0023  000000013  40C000000             VCORF:  .FLT    1.0       ; VOLUME ESTIMATE CORRECTION FACTOR
0024  000000014  000000000             PHLOW:  .WORD   0         ; SPECIAL RANGE LOW LIMIT
0025  000000015  000000000             PHHIGH: .WORD   0         ; SPECIAL RANGE HIGH LIMIT
0026  000000016  000040000             IBDZ:   .WORD   262144    ; IBDZN HIGH LIMIT
0027  000000017  436AAE147             GASCON: .FLT    53.34     ; GAS CONSTANT FOR AIR
```

Listing 1. - Continued.

```
002B                                                            .PSECT  $RECYL, I, RW, LCL, REL, CON
002C                                                    ;+
002D                                                    ; THIS CODE DOES ELEMENT RELOCATION (IF CNTR1 IS ZERO) AND RECYCLING.
002E                                                    ;-
002F  000000000  000000001  00000000F                   RECYL:  TIEA    EEXHST            ; ALWAYS GIVE USER A SHOT AT IT
0030  000000002  00000000E                                      TEEA                      ;
0031  000000003  000000006  000000001  000000074                TTST    MDFR,CNTR1        ; RELOCATE AEROELEMENTS TO INLET ?
0032  000000006  00000000D  000000001  0000000A0                TBNE    MDFR,RECYL1       ; NO
0033  000000009  000000001  00000000B                           TIEA    EREL2             ; YES, DO SO NOW
0034  00000000B  000000002  000000072                           TENL    CTRL1             ;
0035  00000000D  000000013  00000000D                           TEMI    XFZIDL            ;
0036  00000000F  00000000E                                      TEEA                      ;
0037  000000010  000000010  000000003                   RECYL1: TEDM    DFNBZN            ; FIND NEAREST NEIGHBORS FOR ALL, REGARDLESS
0038  000000012  000000000                                              DASEA             ;
0039  000000013  000000001  000000001                           TIEA    EMIGR1            ; BUMP PRESSURE-VOLUME HISTORY RING BUFFER
003A  000000015  000000002  000000080                           TENL    PVHRBP            ;  POINTER
003B  000000017  00000000E                                      TEEA                      ;
003C  000000018  000000010  000000009                           TEDM    DMIGR2            ; REVISE EACH ELEMENT'S POINTER WITH RESULT
003D  00000001A  000000003  000000080                                   DASSM,PVHRBP      ;
003E  00000001C  000000000                                              DASEA             ;
003F  00000001D  00000001E  000000002  00000000C                TCLRF   MDDF,XOMEGA       ; SOLID ANGLE ACCUMULATOR
0040  000000020  000000010  000000004                           TEDM    DVOL              ; ESTIMATE AEROELEMENT VOLUMES AND
0041  000000022  000000003  000000081                                   DASSM,PAV         ;  ACCUMULATE SOLID ANGLES
0042  000000024  000000005  00000000C                                   DASSI,XOMEGA      ;
0043  000000026  000000000                                              DASEA             ;
0044  000000027  000000010  000000001                           TEDM    DVSUM             ; SUM UP AEROELEMENT VOLUME ESTIMATES
0045  000000029  000000005  00000000B                                   DASSI,XVOLA       ;
0046  00000002B  000000000                                              DASEA             ;
0047  00000002C  000000001  00000000C                           TIEA    EREL3             ; COMPUTE AEROELEMENT VOLUME ESTIMATE
0048  00000002E  000000002  000000072                           TENL    CTRL1             ;  CORRECTION FACTOR AND AVERAGE SOLID
0049  000000030  000000002  000000084                           TENL    VREST             ;  ANGLE
004A  000000032  000000012  00000000B                           TESI    XVOLA             ;
004B  000000034  000000002  000000085                           TENL    VCORF             ;
004C  000000036  000000002  000000081                           TENL    PAV               ;
004D  000000038  000000012  00000000C                           TESI    XOMEGA            ;
004E  00000003A  000000012  00000000E                           TESI    XISIZE            ;
004F  00000003C  000000013  00000000D                           TEMI    XFZIDL            ;
0050  00000003E  00000000E                                      TEEA                      ;
0051  00000003F  000000010  000000006                           TEDM    DVCOR             ; CORRECT ALL VOLUME ESTIMATES
0052  000000041  000000003  000000085                                   DASSM,VCORF       ;
0053  000000043  000000000                                              DASEA             ;
0054  000000044  000000010  000000007                           TEDM    DVCORA            ; RESULT COPY-BACK
0055  000000046  000000000                                              DASEA             ;
0056  000000047  000000006  000000001  000000074                TTST    MDFR,CNTR1        ; ASSIGN NEW INLET MASSES ?
0057  00000004A  00000000D  000000001  0000000E4                TBNE    MDFR,RECYL2       ; NO
0058  00000004D  000000001  00000000D                           TIEA    EREL4             ; YES
0059  00000004F  000000002  000000072                           TENL    CTRL1             ;
005A  000000051  000000013  00000000D                           TEMI    XFZIDL            ;
005B  000000053  00000000E                                      TEEA                      ;
005C  000000054  000000010  00000000A                   RECYL2: TEDM    DRHOPR            ; COMPUTE PRESSURES AND DENSITIES FOR ALL
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978          PAGE      4

0050  000000056  000000000                                          DASEA          ;
005E  000000057  000000006  000000001  000000074          TTST      MDFR,CNTR1     ; PLAIN-JANE HISTORY FOR INLET ?
005F  00000005A  00000000D  000000001  0000000FF          TBNE      MDFR,RECYL3    ; NO
0060  00000005D  000000001  00000000E                     TIEA      ESUPHR         ; YES, GET INLET ZONE RANGE LIMITS
0061  00000005F  000000012  00000000F                     TESI      XIDI           ;
0062  000000061  000000002  000000086                     TENL      PHLOW          ;
0063  000000063  000000002  000000087                     TENL      PHHIGH         ;
0064  000000065  00000000E                                TEEA                     ;
0065  000000066  000000010  000000008                     TEDM      DPRSHI         ; SET THE HISTORY FOR EACH INLET AEROELEMENT
0066  000000068  000000003  000000086                               DASSM,PHLOW    ;
0067  00000006A  000000003  000000087                               DASSM,PHHIGH   ;
006B  00000006C  000000005  00000000F                               DASSI,XIDI     ;
0069  00000006E  000000000                                          DASEA          ;
006A  00000006F  000000010  000000005                RECYL3: TEDM   DPWRC          ; POWER OF COMPRESSION FOR ALL
006B  000000071  000000003  00000007C                               DASSM,GDAST    ;
006C  000000073  000000003  00000007D                               DASSM,MGDAST   ;
006D  000000075  000000000                                          DASEA          ;
006E  000000076  000000010  00000000B  000000000          TEDM      DINTF,DASEA    ; INTERPOLATION MATRICIES FOR ALL
006F  000000079  000000005  000000001  000000112          TJMP      MDFR,STOKES    ; JUMP TO NEXT SECTION
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81     19 JUL 1982  10:29:33.978          PAGE     5

0073                                                             .PSECT  $STOKE, I, RW, LCL, REL, CON
0074                                                     ;+
0075                                                     ; THIS SECTION CALCULATES AEROELEMENT ACCELERATIONS VIA THE COMPLETE
0076                                                     ; NAVIER-STOKES EQUATION.
0077                                                     ;-
0078  000000000  000000010  00000000C  000000000  STOKES: TEDM     DSTOK1,DASEA     ;
0079  000000003  000000010  00000000D  000000000          TEDM     DSTOK2,DASEA     ;
007A  000000006  000000005  000000001  000000121          TJMP     MDFR,WORK        ; GO ON TO WORK FLOW
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978            PAGE      6

007E                                                                 .PSECT  $WORK, I, RW, LCL, REL, CON
007F                                                         ;+
0080                                                         ; THIS SECTION CALCULATES THE INTERELEMENT FLOW OF WORK.
0081                                                         ;-
0082  000000000  000000010  00000000E  000000000  WORK:   TEDM    DWRKA,DASEA     ; INITIALIZE THE DATA BASE
0083  000000003  000000010  00000000F  000000000          TEDM    DWRKD,DASEA     ; POWER OF DISTORTION - PHASE 1
0084  000000006  000000010  000000010  000000000          TEDM    DWRKE,DASEA     ; POWER OF DISTORTION - PHASE 2
0085  000000009  000000010  000000011                     TEDM    DWRKF           ; HEAT TRANSFER BETWEEN RECIPROCATING
0086  00000000B  000000003  00000007C                             DASSM,GDAST     ;  NEAREST NEIGHBORS
0087  00000000D  000000000                                        DASEA           ;
0088  00000000E  000000010  000000012                     TEDM    DWRKG           ; ACCUMULATE ALL HEAT TRANSFER CONTRIBUTIONS
0089  000000010  000000003  00000007C                             DASSM,GDAST     ;  AND ADJUST AEROELEMENT TEMPERATURES.
008A  000000012  000000000                                        DASEA           ;
008B  000000013  000000005  000000001  00000013D          TJMP    MDFR,OUTPUT     ;
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978           PAGE      7


008F                                                                   .PSECT  $OUTPU, I, RW, LCL, REL, CON
0090                                                            ;+
0091                                                            ; DATA OUTPUT
0092                                                            ;-
0093   000000000   000000005   000000001   000000146   OUTPUT: TJMP    MDFR,MOVE        ; NO OUTPUT AT THIS TIME
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978            PAGE       8

0097                                                                .PSECT  MOVEL, I, RW, LCL, REL, CON
0098                                                        ;+
0099                                                        ; THIS SECTION MOVES THE AEROELEMENTS BASED UPON THE CALCULATED
009A                                                        ; ACCELERATIONS.
009B                                                        ;-
009C  000000000  000000010  000000013                       MOVE:   TEDM    DMOVEL          ; DO ELEMENT MOTION
009D  000000002  000000003  00000007E                                       DASSM,CURTIM    ;
009E  000000004  000000003  00000007C                                       DASSM,GDAST     ;
009F  000000006  000000003  00000007F                                       DASSM,NINC      ;
00A0  000000008  000000000                                                  DASEA           ;
00A1  000000009  000000010  000000014  000000000                    TEDM    DMOVL2,DASEA    ; RESULT COPY-BACK
00A2  00000000C  00000001D  000000001  00000007C                    TADDF   MDFR,GDAST      ; BUMP CURRENT TIME
00A3  00000000F  000000001  00000007E                                       MDFR,CURTIM     ;
00A4  000000011  000000005  000000001  000000160                    TJMP    MDFR,SORT       ;
```

Listing 1. - Continued.

```
00A8                                                            .PSECT  $SORT, I, RW, LCL, REL, CON
00A9                                                    ;+
00AA                                                    ; THIS SECTION DECREMENTS THE AEROELEMENT RELOCATION COUNTER FOR THE
00AB                                                    ; NEXT PASS.  IF THE RESULT IS ZERO, CTRL2 IS SET TO 4 TO CAUSE GENERATION
00AC                                                    ; OF THE FLOW ZONE RESIDENT AEROELEMENT CROSS REFERENCE INFORMATION.
00AD                                                    ;-
00AE 000000000 000000019 000000000 000000004  SORT:     TMOV    MIMD,4          ; ASSUME A RELOCATION PASS
00AF 000000003 000000001 000000073                              MDFR,CTRL2      ;
00B0 000000005 000000016 000000001 000000074            TDEC    MDFR,CNTR1      ; SHALL WE RELOCATE ?
00B1 000000008 00000000A 000000001 000000178            TBEQ    MDFR,SORT2      ; YES
00B2 00000000B 000000008 000000001 000000173            TBGT    MDFR,SORT1      ; NO, STILL IN WAIT LOOP
00B3 00000000E 000000019 000000001 000000075            TMOV    MDFR,CNTR2      ; NO, RE-INITIALIZE COUNTER TO BEGIN
00B4 000000011 000000001 000000074                              MDFR,CNTR1      ;  ANOTHER WAIT LOOP
00B5 000000013 000000019 000000000 000000001  SORT1:    TMOV    MIMD,1          ; TURN OFF CROSS-REFERENCE REQUEST
00B6 000000016 000000001 000000073                              MDFR,CTRL2      ;
00B7 000000018 000000001 000000003            SORT2:    TIEA    ERES07          ; ZAP CROSS-REFERENCE CONTROLS, REGARDLESS
00B8 00000001A 000000013 00000000D                      TEMI    XFZIDL          ;
00B9 00000001C 00000000E                                TEEA                    ;
00BA 00000001D 000000010 000000015 000000000            TEDM    DRES01,DASEA    ; INHIBIT REDUNDANT SORT CHECKS
00BB 000000020 000000010 000000016                      TEDM    DRES02          ; DO THE SORT
00BC 000000022 000000003 000000073                              DASSM,CTRL2     ;
00BD 000000024 000000006 00000000D                              DASMI,XFZIDL    ;
00BE 000000026 000000000                                        DASEA           ;
00BF 000000027 000000001 000000005                      TIEA    EDBBF           ; FLUSH CROSS-REFERENCE RESULTS
00C0 000000029 00000000E                                TEEA                    ;
00C1 00000002A 000000005 000000001 000000090            TJMP    MDFR,RECYL      ; LOOP BACK
```

Listing 1. - Continued.

```
00C5                                        .PSECT  .$ABS.
00C6                                ;+
00C7                                ; INITIALIZE INDIRECT POINTERS
00C8                                ;-
00C9               00000000B        . =     XVOLA            ;
00CA  00000000B    000000083                .WORD   VOLA     ;
00CB               00000000C        . =     XOMEGA           ;
00CC  00000000C    000000082                .WORD   OMEGA    ;
00CD               00000000D        . =     XFZIDL           ;
00CE  00000000D    000000076                .WORD   FZIDL    ;
00CF               00000000E        . =     XISIZE           ;
00D0  00000000E    00000007A                .WORD   ISIZE    ;
00D1               00000000F        . =     XIDI             ;
00D2  00000000F    000000077                .WORD   IDI      ;
00D3               000000010        . =     XIDE             ;
00D4  000000010    000000078                .WORD   IDE      ;
00D5               000000011        . =     XIDU             ;
00D6  000000011    000000079                .WORD   IDU      ;
00D7               000000012        . =     XBDZNS           ;
00D8  000000012    000000088                .WORD   IBDZ     ;
00D9               000000013        . =     XGASCO           ;
00DA  000000013    000000089                .WORD   GASCON   ;
00DB               000000065        . =     101              ;
00DC  000000065    00000007E                .WORD   CURTIM   ;
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978            PAGE     11


00E0                                            ;+
00E1                                            ; END OF PROGRAM
00E2                                            ;-
00E3            000000090                               .END    RECYL   ;
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81       19 JUL 1982  10:29:33.978            PAGE     12

                     ******** SYMBOL TABLE ********

0003 CNTR1   000000074 R        0003 CNTR2   000000075 R        0000 CON     ********* R        0003 CTRL1   000000072 R
0003 CTRL2   000000073 R        0003 CURTIM  00000007E R        0000 D       ********* R        0000 DASEA   000000000 RD
0000 DASMI   000000006 RD       0000 DASSI   000000005 RD       0000 DASSM   000000003 RD       0000 DFNBZN  000000003 RD
0000 DINTF   00000000B RD       0000 DMIGR2  000000009 RD       0000 DMOVEL  000000013 RD       0000 DMOVL2  000000014 RD
0000 DPRSHI  000000008 RD       0000 DPWRC   000000005 RD       0000 DRES01  000000015 RD       0000 DRES02  000000016 RD
0000 DRHOPR  00000000A RD       0000 DSTOK1  00000000C RD       0000 DSTOK2  00000000D RD       0000 DVCOR   000000006 RD
0000 DVCORA  000000007 RD       0000 DVOL    000000004 RD       0000 DVSUM   000000001 RD       0000 DWRKA   00000000E RD
0000 DWRKD   00000000F RD       0000 DWRKE   000000010 RD       0000 DWRKF   000000011 RD       0000 DWRKG   000000012 RD
0000 EDBBF   000000005 RD       0000 EEXHST  00000000F RD       0000 EMIGR1  000000001 RD       0000 EREL2   00000000B RD
0000 EREL3   00000000C RD       0000 EREL4   00000000D RD       0000 ERES07  000000003 RD       0000 ESUPHR  00000000E RD
0003 FZIDL   000000076 R        0003 GASCON  000000089 R        0003 GDAST   00000007C R        0000 I       ********* R
0003 IBDZ    000000088 R        0003 IDE     000000078 R        0003 IDI     000000077 R        0003 IDU     000000079 R
0003 ISIZE   00000007A R        0000 LCL     ********* R        0000 MDDF    000000002 RD       0000 MDFR    000000001 RD
0003 MGDAST  00000007D R        0000 MIMD    000000000 RD       0008 MOVE    000000146 R        0000 MOVEL   ********* R
0003 NINC    00000007F R        0003 NZN     00000007B R        0003 OMEGA   000000082 R        0007 OUTPUT  00000013D R
0003 PAV     000000081 R        0003 PHHIGH  000000087 R        0003 PHLOW   000000086 R        0003 PVHRBF  000000080 R
0004 RECYL   000000090 R        0004 RECYL1  0000000A0 R        0004 RECYL2  0000000E4 R        0004 RECYL3  0000000FF R
0000 REL     ********* R        0000 RW      ********* R        0009 SORT    000000160 R        0009 SORT1   000000173 R
0009 SORT2   000000178 R        0005 STOKES  000000112 R        0003 VCORF   000000085 R        0003 VOLA    000000083 R
0003 VREST   000000084 R        0006 WORK    000000121 R        0000 XBDZNS  000000012 RD       0000 XFZIDL  00000000D RD
0000 XGASCO  000000013 RD       0000 XIDE    000000010 RD       0000 XIDI    00000000F RD       0000 XIDU    000000011 RD
0000 XISIZE  00000000E RD       0000 XOMEGA  00000000C RD       0000 XVOLA   00000000B RD       0000 $DATA   ********* R
0000 $OUTPU  ********* R        0000 $RECYL  ********* R        0000 $SORT   ********* R        0000 $STOKE  ********* R
0000 $WORK   ********* R        0000         ********* R        0000 .$ABS.  ********* R
```

Listing 1. - Continued.

```
PAX CONTROL CODE ASSEMBLER -- X01.00A  10 AUG 81      19 JUL 1982  10:29:33.978            PAGE     13


                  ******** PROGRAM SECTION TABLE ********

0001 .$ABS.  000000000  000000066  (       0.        102.)   D    RW    LCL   ABS   CON
0002 .$REL.  00000006C  000000000  (     108.          0.)   I    RW    LCL   REL   CON
0003 $DATA   000000072  000000018  (     114.         24.)   D    RW    LCL   REL   CON
0004 $RECYL  000000090  00000007C  (     144.        124.)   I    RW    LCL   REL   CON
0005 $STOKE  000000112  000000009  (     274.          9.)   I    RW    LCL   REL   CON
0006 $WORK   000000121  000000016  (     289.         22.)   I    RW    LCL   REL   CON
0007 $OUTPU  00000013D  000000003  (     317.          3.)   I    RW    LCL   REL   CON
0008 MOVEL   000000146  000000014  (     326.         20.)   I    RW    LCL   REL   CON
0009 $SORT   000000160  00000002D  (     352.         45.)   I    RW    LCL   REL   CON



ERROR REPORTS FOR 19 JUL 1982 AT 10:34:33.344

*** NO ERRORS TO REPORT ***
```

Listing 1. - Concluded.

```
 @FOR,MS CASPER1.VCORD
FOR   4R1   E -01/13/83-14:03:10 (1,)
>@EOF

   SUBROUTINE VCOR      ENTRY POINT 000051

   STORAGE USED: CODE(1) 000070; DATA(0) 000015; BLANK COMMON(2) 000000

   EXTERNAL REFERENCES (BLOCK, NAME)

    0003   CHKLH
    0004   CHKTIM
    0005   V
    0006   STAESC
    0007   NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000017 116G      0001   000040 20L       0000 I 000004 I         0000 I 000001 ID        0000 I 000002 IE
    0000   000005 INJP$     0000 I 000000 IS        0000 R 000003 R         0005 R 000000 V


00101       1*          SUBROUTINE VCOR (IL,IH,VL)                                         13200010    000000
00101       2*      C+                                                                     13200020    000000
00101       3*      C                                                                      13200030    000000
00101       4*      C   VCOR         ****** A SUBROUTINE FOR CASPER ******                 13200040    000000
00101       5*      C                AUTHOR      WILLIAM HENRY JONES                       13200050    000000
00101       6*      C                V01-00      02 FEB 79                                 13200060    000000
00101       7*      C                V01-00A     08 FEB 80      SEPARATES INPUTS AND OUTPUT            000000
00101       8*      C                                                                      13200070    000000
```

Listing 2. - Simple parallel computation.

```
00101      9*    C                                                            13200080    000000
00101     10*    C      DESCRIPTION ******                                    13200090    000000
00101     11*    C                                                            13200100    000000
00101     12*    C      APPLIES A SUPPLIED MULTIPLICATIVE CORRECTION TO THE VOLUME 13200110  000000
00101     13*    C      ESTIMATES OF ALL AEROELEMENTS IN THE RANGE 'IL' TO 'IH'.   13200120  000000
00101     14*    C                                                            13200130    000000
00101     15*    C-                                                           13200140    000000
00103     16*           INTEGER IL,IH,IS,ID,IE                                13200150    000000
00104     17*           REAL VL,R                                             13200160    000000
00105     18*           DATA ID/132/                                          13200170    000000
00107     19*           DATA IE/1/                                            13200180    000000
00111     20*           CALL CHKLH (IL,IH,IS,ID,IE)      @ CHECK AEROELEMENT RANGE  13200190  000000
00112     21*           IF (IS) 17,20,17                 @ VALID RANGE ?      13200200    000006
00115     22*     17    DO 19 I=IL,IH,IS                 @ YES, APPLY CORRECTION    13200210  000010
00120     23*           CALL CHKTIM (IL,IH,I)            @ KEEP AN EYE ON THE TIME            000017
00121     24*           R=VL*V(I)                        @                    13200220    000024
00122     25*    19     CALL STAESC (I,R)                @ RESULT TO SCRATCH SLOT               000031
00124     26*     20    RETURN                           @                    13200240    000040
00125     27*           END                                                   13200250    000067
END FOR
>
```

Listing 2. - Concluded.

```
@FOR,MS CASPER9.MOVELD
FOR   4R1   E -01/13/83-14:04:09 (6,)
>@EOF

   SUBROUTINE MOVEL     ENTRY POINT 000556

   STORAGE USED: CODE(1) 001015; DATA(0) 000172; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003   NZNC   000001
    0004   IDUC   000001

   EXTERNAL REFERENCES (BLOCK, NAME)

    0005   FZ
    0006   STAT
    0007   ZBL
    0010   IPZBL
    0011   IPLZN
    0012   LZN
    0013   CHKLH
    0014   CHKTIM
    0015   STIAES
    0016   X
    0017   U
    0020   A
    0021   STS
    0022   GRDBD
    0023   SURFVE
    0024   STSTAT
    0025   ERROR2
    0026   TSTZN
```

Listing 3. - Parallel computation with coordinated algorithm.

```
 0027   TSTBDT
 0030   SQRT
 0031   NERR3$

STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

 0001   000147 101L      0001   000263 106L      0001   000200 107L      0001   000035 136G      0001   000055 144G
 0001   000137 165G      0001   000202 206G      0001   000212 215G      0001   000501 219L      0001   000535 221L
 0001   000227 223G      0001   000242 232G      0001   000251 237G      0001   000301 253G      0001   000027 2796L
 0001   000666 29L       0001   000351 301G      0001   000361 310G      0001   000417 331G      0001   000431 341G
 0001   000310 3500L     0001   000335 3570L     0001   000346 3600L     0001   000410 3710L     0001   000305 373G
 0001   000425 3760L     0001   000436 3790L     0001   000440 3800L     0001   000454 3840L     0001   000470 3880L
 0001   000747 41L       0001   000603 411G      0001   000751 42L       0001   000706 436G      0001   000757 45L
 0020 R 000000 A         0000 R 000001 AL        0000 R 000066 B         0000 R 000067 C         0005 I 000000 FZ
 0000 R 000024 GB        0000 I 000057 I         0000 I 000104 ID        0000 I 000063 IDB       0000 I 000076 IDMIN
 0004 I 000000 IDU       0000 I 000075 IE3F      0000 I 000056 IFZ       0000   000134 INJP$     0000   000126 INJP$
 0000   000121 INJP$     0011 I 000000 IPLZN     0010 I 000000 IPZBL     0000 I 000000 IS        0000 I 000106 IT
 0000 I 000055 I1        0000 I 000072 J         0000 I 000065 K         0012 I 000000 LZN       0000 I 000077 N
 0000 I 000100 N         0000 I 000101 NA        0000 I 000102 NB        0000 I 000103 NC        0000 I 000070 NNN
 0003 I 000000 NZN       0006 I 000000 STAT      0000 R 000105 SV        0000 R 000064 SVMIN     0000 R 000060 T
 0000 R 000073 TA        0000 R 000054 TE        0000 R 000052 TINC      0000 R 000053 TINCSQ    0000 R 000062 TLEFT
 0000 R 000071 TN        0000 R 000074 TQ        0000 R 000061 TS        0017 R 000000 U         0000 R 000043 UA
 0000 R 000027 UL        0000 R 000037 UN        0000 R 000033 UQ        0000 R 000047 VBO       0016 R 000000 X
 0000 R 000020 XA        0000 R 000004 XL        0000 R 000014 XN        0000 R 000010 XQ        0007 R 000000 ZBL


00101       1*                                                                                        000000
00101       2*                                                                                        000000
00101       3*                                                                                        000000
00101       4*                                                                                        000000
00101       5*                                                                                        000000
00101       6*                                                                                        000000
00101       7*                                                                                        000000
```

Listing 3. - Continued.

```
00101         SUBROUTINE MOVEL (IL,IH,CURTIM,GDAST,MINC)                                    901A0010   000000
00101      C                                                                                90100030   000000
00101      C                                                                                90100040   000000
00101      C  MOVEL       ****** A SUBROUTINE FOR CASPER ******                                 90100050   000000
00101      C              AUTHOR      WILLIAM HENRY JONES                                   90100060   000000
00101      C              V02-00      14 APR 77                                             90100070   000000
00101      C              V02-01      22 JUN 77                                             90110071   000000
00101      C              V02-02      26 JUL 77                                             90120072   000000
00101      C              V02-03      22 SEP 77                                             90130073   000000
00101      C              V02-04      22 SEP 77                                             90140074   000000
00101      C              V02-05      26 SEP 77                                             90150075   000000
00101      C              V02-06      01 JUN 78                                             90160076   000000
00101      C              V02-07      16 JUN 78                                             90170077   000000
00101      C              V02-08      29 AUG 78                                             90180078   000000
00101      C              V02-08A     13 FEB 79                                             901A0079   000000
00101      C              V02-08B     09 MAR 79                                             901B0080   000000
00101      C              V02-08C     13 FEB 80      INPUT/OUTPUT SEGREGATION                          000000
00101      C              V02-08D     15 SEP 80      FUNCTION TYPE STATEMENTS                          000000
00101      C              V02-08E     28 SEP 81      MOVING BOUNDARIES                      V02-08E    000000
00101      C              V02-08F     06 JAN 83      BAD POSITION INTEGRATION               V02-08F    000000
00101      C                                                                                90100090   000000
00101      C  ARGUMENTS IN CASPER 'CACHE' MEMORY ******                                         90180094   000000
00101      C                                                                                90180096   000000
00101      C  ARGUMENT    TYPE        DIMENSION      DESCRIPTION                            90100100   000000
00101      C  ---------   ---------   ---------      -------------------                    90180110   000000
00101      C  X           REAL        1 TO ISIZE     AEROELEMENT POSITION                   90100120   000000
00101      C                          1 TO 3           COORDINATES                          90100130   000000
00101      C                                                                                90100140   000000
00101      C  U           REAL        1 TO ISIZE     AEROELEMENT VELOCITIES                 90100150   000000
00101      C                          1 TO 3                                                90100160   000000
00101      C                                                                                90100170   000000
00101      C  A           REAL        1 TO ISIZE     AEROELEMENT ACCELERATIONS              90100180   000000
00101      C                          1 TO 3                                                90100190   000000
00101      C                                                                                90100200   000000
00101      C  FZ          INTEGER     1 TO ISIZE     AEROELEMENT FLOW ZONE                  90100210   000000
00101      C                                           NUMBERS (BY AEROELEMENT)             90100220   000000
```

Listing 3. - Continued.

```
00101   C                                                                   90100230   000000
00101   C   STAT       INTEGER    1 TO ISIZE    AEROELEMENT STATUS LIST     90100240   000000
00101   C                                                                   90100360   000000
00101   C   ZBL        INTEGER    1 TO ZBLSZ    BOUNDARY LIST BY FLOW ZONES90100370    000000
00101   C                                                                   90100380   000000
00101   C   IPZBL      INTEGER    1 TO NZN      ZBL CONTROL PARAMETERS LIST90100390    000000
00101   C                         1 TO 2          (X,1) = STARTING POINT    90100400   000000
00101   C                                         (X,2) = STRING LENGTH     90100410   000000
00101   C                                                                   90100420   000000
00101   C                                                                   90100560   000000
00101   C   ARGUMENTS PASSED IN SUBROUTINE CALL ******                      90180562   000000
00101   C                                                                   90180564   000000
00101   C   ARGUMENT   TYPE       DIMENSION     DESCRIPTION                 90180566   000000
00101   C   ---------  ---------  ---------     ------------------          90180568   000000
00101   C   IL         INTEGER    SCALAR        AEROELEMENT ID LOW LIMIT    901A0569   000000
00101   C                                                                   901A0570   000000
00101   C   IH         INTEGER    SCALAR        AEROELEMENT ID HIGH LIMIT   901A0571   000000
00101   C                                                                   901A0572   000000
00101   C   CURTIM     REAL       SCALAR        CURRENT OPENING TIME        901A0573   000000
00101   C                                                                   901A0574   000000
00101   C   GDAST      REAL       SCALAR        BASIC TIME INCREMENT        901A0575   000000
00101   C                                                                   90180576   000000
00101   C   NINC       INTEGER    SCALAR        NUMBER OF TIME SUB-         90180578   000000
00101   C                                        INCREMENTS                 90180580   000000
00101   C                                                                   90180582   000000
00101   C                                                                   90180584   000000
00101   C   RESULT LOCATIONS ******                                                    000000
00101   C                                                                              000000
00101   C   LOCATION   CONTENTS                                                        000000
00101   C   ---------  ------------------------                                        000000
00101   C   AESCRA     FINAL FLOW ZONE ID OF AEROELEMENT                               000000
00101   C                                                                              000000
00101   C   S(1)       X(1)                                                            000000
00101   C   S(2)       X(2)                                                            000000
00101   C   S(3)       X(3)                                                            000000
00101   C   S(4)       U(1)                                                            000000
```

Listing 3. - Continued.

```
00101   C     S(5)        U(2)                                                        000000
00101   C     S(6)        U(3)                                                        000000
00101   C                                                                             000000
00101   C                                                                             000000
00101   C     DESCRIPTION ******                                            90180586  000000
00101   C                                                                   90180588  000000
00101   C     MOVEL IS A SUBROUTINE WHICH, GIVEN THE POSITION, VELOCITY, AND  90100590  000000
00101   C     ACCELERATION OF INDIVIDUAL AEROELEMENTS AS WELL AS A DEFINITION 90100600  000000
00101   C     OF THE BOUNDARIES AND FLOW ZONES OF THE AIRFLOW VOLUME, WILL    90100610  000000
00101   C     REPOSITION THOSE AEROELEMENTS THAT ARE NOT RESTRICTED TO OTHER  90100620  000000
00101   C     PRESET LAWS OF MOTION (E.G. - BOUNDARY ELEMENTS FIXED IN SPACE) 90100630  000000
00101   C     ACCORDING TO THE CLASSIC INTEGRATION OF CONSTANTLY ACCELERATING 90100640  000000
00101   C     MOTION.                                                         90100650  000000
00101   C                                                                     90100660  000000
00101   C     DURING SUCH RELOCATION EACH APPROPRIATE BOUNDARY IS CHECKED FOR 90100670  000000
00101   C     POTENTIAL VIOLATIONS BY THE AEROELEMENT.  IF SUCH A VIOLATION IS 90100680  000000
00101   C     DETECTED THE POINT OF VIOLATION IS FOUND AND THE AEROELEMENT IS 90100690  000000
00101   C     ELASTICALLY BOUNCED OFF THE BOUNDARY AT THAT LOCATION.  TO ENHANCE90100700  000000
00101   C     BOUNDARY VIOLATION DETECTION A SUB-INCREMENTAL TIME STEP IS     90100710  000000
00101   C     SPECIFYABLE BY THE INTEGER ARGUMENT NINC.  THIS WILL DIVIDE THE 90100720  000000
00101   C     PARABOLIC MOTION FROM X @ T TO X @ T+GDAST INTO NINC EQUAL STEPS 90100730  000000
00101   C     AND CHECK FOR BOUNDARY VIOLATIONS AT EACH OF THE INTERMEDIATE   90100740  000000
00101   C     POSITIONS, THUS LOWERING THE PROBABILITY OF AEROELEMENTS 'PASSING 90100750  000000
00101   C     THROUGH' THIN BOUNDARIES SUCH AS LEADING AND TRAILING EDGES OF  90100760  000000
00101   C     AIRFOILS.                                                       90100770  000000
00101   C                                                                     V02-08E   000000
00101   C     THE BOUNDARY BOUNCING PROCESS IS A SIMPLE REFLECTION ALGORITHM, V02-08E   000000
00101   C     I.E., ANGLE OF INCIDENCE EQUALS ANGLE OF REFLECTION.  TO DO THIS, V02-08E   000000
00101   C     THE VELOCITY VECTOR FOR THE AEROELEMENT IS ADJUSTED AT THE TIME V02-08E   000000
00101   C     OF BOUNCE TO GIVE THE APPROPRIATE INITIAL DIRECTION.  THE       V02-08E   000000
00101   C     ACCELERATION OF THE AEROELEMENT IS NOT ADJUSTED.  TO ACCOUNT    V02-08E   000000
00101   C     FOR SITUATIONS WHERE THE AEROELEMENT IS NOT MOVING AND IS HIT   V02-08E   000000
00101   C     BY A MOVING BOUNDARY, THE AEROELEMENT VELOCITY IS FIRST CONVERTED V02-08E   000000
00101   C     TO A VELOCITY RELATIVE TO THE BOUNDARY, ADJUSTED FOR THE BOUNCE, V02-08E   000000
00101   C     AND THEN CONVERTED BACK TO VELOCITY RELATIVE TO THE STATIONARY  V02-08E   000000
00101   C     REFERENCE FRAME.                                                V02-08E   000000
```

Listing 3. - Continued.

```
00101   C                                                                              90100780   000000
00101   C     SELECTED VARIABLES IN THE ARGUMENT LIST, NOTABLY LZN, ZBL, BDZN,          90100830   000000
00101   C     AND NEIZN, ARE PASSED WITH CONTROL PARAMETER LISTS IN DYNAMICALLY         90100840   000000
00101   C     VARIABLE ARRAY FORM.  AS NOTED IN THE ARGUMENT DESCRIPTIONS, THE          90100850   000000
00101   C     CONTROL PARAMETERS CONSIST OF A STARTING POINT LIST AND A STRING          90100860   000000
00101   C     LENGTH LIST.  THESE DYNAMICALLY VARIABLE                                  90180870   000000
00101   C     ARRAYS ARE ARRANGED SUCH THAT THE SUB-ARRAY RUNS FROM 1 TO THE            90100890   000000
00101   C     STRING LENGTH AND THE FIRST ELEMENT IS AT THE STARTING POINT PLUS         90100900   000000
00101   C     1.  THUS, FOR LZN, THE J TH ELEMENT OF THE I TH FLOW ZONE LIST            90100910   000000
00101   C     WOULD BE LZN(IPLZN(I,1)+J) AND THE LENGTH OF THE I TH FLOW ZONE           90100920   000000
00101   C     LIST WOULD BE IPLZN(I,2).                                                 90100930   000000
00101   C                                                                              90100940   000000
00101   C                                                                              90100950   000000
00101   C                                                                              90100960   000000
00101   C                                                                              90100970   000000
00101   C                                                                              90100980   000000
00101   C                                                                              90100990   000000
00101   C                                                                              90101000   000000
00101   C                                                                              90102500   000000
00101   C     REQUIRED SUBROUTINES ******                                               90182510   000000
00101   C                                                                              90182512   000000
00101   C      401        AVIRI                                                         90182514   000000
00101   C                                   402        X                                90182516   000000
00101   C                                   404        U                                90182518   000000
00101   C                                   406        A                                90182520   000000
00101   C     417        STSTAT             416        STAT                             90182522   000000
00101   C     421        STFZ               420        FZ                               90182524   000000
00101   C                                   470        FVIRI                            90182534   000000
00101   C     471        ZBL                                                            90182536   000000
00101   C     475        HVIRI              476        IPZBL                            90182538   000000
00101   C     911        MRRM               912        TSTZN                            90182540   000000
00101   C                                                                              90182542   000000
00101   C                                                                              90182544   000000
00101   C     ERRORS REPORTED ******                                                    90182546   000000
00101   C                                                                              90182548   000000
00101   C     1          NEITHER RESULT OF BOUNDARY SURFACE FINDER WAS                  90182550   000000
```

Listing 3. - Continued.

```
00101   C                SAFE (901X3870).                                 90182552   000000
00101   C      2        BOUNDARY SURFACE FINDER SETUP PUSHED 'Q' BACK     90182554   000000
00101   C                BEYOND ZERO TIME WITHOUT FINDING A SAFE POSITION 90182556   000000
00101   C                (901X3348).                                      90182558   000000
00101   C      3        AN AEROELEMENT FLOW PATH WAS FOUND THAT LEADS TO  901A2560   000000
00101   C                BOUNDARY VIOLATION WITHOUT CROSSING A LEGITIMATE 901A2565   000000
00101   C                ACTIVE BOUNDARY.                                 901A2570   000000
00101   C      4        THE BOUNDARY INTERCEPT LOCATOR FAILED TO LOCK ON  901A2575   000000
00101   C                TO AN EXISTING ACTIVE BOUNDARY.                  901A2580   000000
00101   C                                                                 901A2585   000000
00101   C-                                                                901A2590   000000
00103          INTEGER IL,IH,IS                                          901A2595   000000
00104          INTEGER FZ,STAT,EIPLZN,ST,EMOD,STP1,ELZN,DLZN                        000000
00105          COMMON /NZNC/NZN                                          90102620   000000
00106          COMMON /IDUC/IDU                                          90182630   000000
00107          REAL CURTIM,GDAST                                         90182670   000000
00110          INTEGER NINC                                              90182680   000000
00111          REAL AL(3),XL(4),XQ(4),XN(4),XA(4),GB(3)                  V02-08F    000000
00112          REAL UL(4),UQ(4),UN(4),UA(4)                              V02-08F    000000
00113          REAL VBO(3)                                               V02-08E    000000
00114          DEFINE EZBL(I)=ZBL(I)                                     90112715   000000
00115          DEFINE DZBL(I,J)=EZBL(IPZBL(I,1)+J)                       90112720   000000
00116          DEFINE EIPLZN(I,J)=IPLZN(I,J)                             90112725   000000
00117          DEFINE ST(I)=EIPLZN(FZ(I),1)                              90112730   000000
00120          DEFINE NU(I)=EIPLZN(FZ(I),2)                              90112740   000000
00121          DEFINE EMOD(I,J)=MOD(I,J)                                 90112745   000000
00122          DEFINE STP1(I)=EIPLZN(EMOD(FZ(I),NZN)+1,1)                90112750   000000
00123          DEFINE ELZN(I)=LZN(I)                                     90112755   000000
00124          DEFINE DLZN(I)=ELZN(ST(I)+NU(I))                          90112760   000000
00125          TINC=GDAST/NINC                @CALC SUB-INCREMENT        90102770   000000
00126          TINCSQ=0.5*TINC*TINC           @CALC 1/2 SQUARE SUB-INCREMENT90112780   000004
00127          TE=0.001*TINC                  @TOLERANCE OF BOUNCES IN TIME 90132785   000007
00130          CALL CHKLH (IL,IH,IS,901,5)    @ GO CHECK AEROELEMENT RANGE  901A2790   000012
00131          IF (IS) 2796,2794,2796         @ VALID RANGE ?            901A2792   000021
00134     2794 RETURN                         @ NO                       901A2794   000023
00135     2796 DO 221 I1=IL,IH,IS             @ IN RANGE AEROELEMENT LOOP 901A2796   000027
```

Listing 3. - Continued.

```
            CALL CHKTIM (IL,IH,I1)             @ KEEP AN EYE ON THE TIME                   000035
            IFZ=FZ(I1)                         @ LOCALIZE                                  000042
            CALL STIAES (I1,IFZ)               @ OUTPUT IN CASE OF A SKIP                  000046
            DO 201 I=1,3                       @ LOCALIZE                                  000055
            XL(I)=X(I1,I)                      @  ORIGINAL POSITION                        000055
            UL(I)=U(I1,I)                      @  ORIGINAL VELOCITY                        000062
            AL(I)=A(I1,I)                      @  ACCELERATION                             000067
            CALL STS (I1,I,XL(I))              @ OUTPUT POSITION AND VELOCITY              000074
            CALL STS (I1,I+3,UL(I))            @  IN CASE OF A SKIP                        000103
      201   CONTINUE                           @                                           000117
            IF (AND(STAT(I1),2**9)) 221,2820,221 @ SKIP IF SPECIAL                         000117
     2820   IF (IFZ-IDU) 2830,221,2830         @ SKIP IF IN THE DOG HOUSE                  000127
     2830   XL(4)=CURTIM                       @                                           000132
            DO 219 I=1,NINC                    @START TIME SUB-INCREMENT LOOP90102850      000137
            T=TINC                             @SET TIME SPAN                 90102860     000137
            TS=TINCSQ                          @SET 1/2 SQUARE TIME SPAN      90102870     000141
            TLEFT=0.0                          @SET TIME REMAINING SUB-INC    90102880     000143
            IDB=-1                             @SET BOUNDARY NO-VIO FLAG    , 90102890     000144
      101   CALL SPC (XL,UL,T)                 @ POSITION AT END OF SPAN      V02-08F      000147
            CALL YNM (XL)                      @ GO CHECK ALL BOUNDARIES      901A2930     000153
            IF (SVMIN) 106,106,104             @ 106 ON BOUNDARY VIOLATION    901A2940     000156
      104   IF (IDB) 219,105,105               @CHK FOR PENDING BOUNCE        90103070     000161
      105   CALL GRDBD (XL,IDB,GB)             @BOUNCE - GET GRADIENT         901A3080     000164
            CALL SURFVE (XL,IDB,VBO,$107)      @ GET SURFACE VELOCITY         V02-08E      000171
      107   DO 108 K=1,3                       @ GET VELOCITY OF AEROELEMENT V01-08E       000202
      108   UL(K)=UL(K)-VBO(K)                 @  RELATIVE TO SURFACE         V02-08E      000202
            B=0.0                              @CLR ACCUMULATOR               90103090     000205
            C=0.0                              @CLR ACCUMULATOR               90103100     000206
            DO 204 K=1,3                       @ACCUMULATE LENGTH SQUARED     90103110     000212
      204   B=B+GB(K)**2                       @  OF GRADIENT VECTOR          90103120     000212
            B=1.0/SQRT(B)                      @FAST DIVIDE LENGTH OF GRAD    90133130     000216
            DO 205 K=1,3                       @LOOP TO                       90103140     000227
            GB(K)=B*GB(K)                      @  NORMALIZE GRADIENT          90103150     000227
      205   C=C+GB(K)*UL(K)                    @  ACCUMULATE DOT PROD VELOCTY90103160      000231
            C=2.0*C                            @ADJUST CONSTANT FOR BOUNCE    90103170     000235
            DO 206 K=1,3                       @BOUNCE! VELOCITY ANGLE INCID.90103180      000242
```

Listing 3. - Continued.

```
00234    224*      206  UL(K)=UL(K)-C*GB(K)                @  EQUALS ANGLE OF REFLECTION 90103190    000242
00236    225*           DO 208 K=1,3                       @ AEROELEMENT VELOCITY RELA-  V02-08E     000251
00241    226*     208   UL(K)=UL(K)+VBO(K)                 @  TIVE TO STATIONARY FRAME   V02-08E     000251
00243    227*           T=TLEFT                            @SET TIME SPAN FOR REST OF    90103200    000254
00244    228*           TLEFT=0.0                          @CLR REMAINING TIME           90103220    000256
00245    229*           IDB=-1                             @WIPE OUT PENDING BOUNCE      90103230    000257
00246    230*           GO TO 101                          @JMP BACK - FINISH SUB-INCREMT90103240    000261
00246    231*     C+                                                                     901A3250    000261
00246    232*     C     THE FOLLOWING SECTION IS ENTERED WHEN A BOUNDARY IS VIOLATED.    901A3260    000261
00246    233*     C     IT BACKS THE PARTICLE UP ALONG ITS PATH TO LOCATE THE POINT      901A3270    000261
00246    234*     C     AT WHICH IT FIRST PENETRATES A BOUNDARY.  WHEN THIS POINT IS     901A3280    000261
00246    235*     C     LOCATED TO WITHIN TOLERANCE 'TE' THE ID OF THE BOUNDARY          901A3290    000261
00246    236*     C     ABOUT TO BE VIOLATED IS LOADED INTO 'IDB' AND THE TIME IS        901A3300    000261
00246    237*     C     SUBDIVIDED TO CAUSE A STEP JUST TO THE BOUNDARY FOLLOWED BY      901A3310    000261
00246    238*     C     A BOUNCE AND A STEP TO THE END OF THE TIME SUB-INCREMENT.        901A3320    000261
00246    239*     C                                                                      901A3330    000261
00246    240*     C     IN LOCATING THE NEAR-VIOLATION POINT ALL BOUNDARIES IN THE       901A3340    000261
00246    241*     C     MANDATORY FLOW ZONE AND IN THE AEROELEMENT'S FLOW ZONE OF        901A3350    000261
00246    242*     C     RESIDENCE ARE CHECKED TO PRODUCE A VIOLATION/NO-VIOLATION        901A3360    000261
00246    243*     C     DECISION.  THIS DECISION SHOULD BE BASED ON AT LEAST ONE         901A3370    000261
00246    244*     C     BOUNDARY EVALUATION THAT DID NOT TRUNCATE.  IF THIS IS NOT       901A3380    000261
00246    245*     C     THE CASE ERROR #3 IS REPORTED.  A PROPER CASPER PROBLEM          901A3390    000261
00246    246*     C     SETUP MAY NOT HAVE ANY AEROELEMENT FLOW PATH THAT CROSSES        901A3400    000261
00246    247*     C     FROM A NON-VIOLATION AREA TO A VIOLATION AREA WITHOUT            901A3410    000261
00246    248*     C     CROSSING A DEFINED BOUNDARY SURFACE.                             901A3420    000261
00246    249*     C-                                                                     901A3430    000261
00247    250*      106  NNN=OR(STAT(I1),2**8)              @ SET MANDATORY SIFT BIT      901A3440    000263
00250    251*           CALL STSTAT (I1,NNN)               @                             901A3450    000270
00251    252*           TN=T                               @ N IMPLIES A POINT IN        901A3460    000274
00252    253*           DO 3480 J=1,4                      @  VIOLATION                  901A3470    000301
00255    254*           UN(J)=UL(J)                        @                             V02-08F     000301
00256    255*     3480  XN(J)=XL(J)                        @                             901A3480    000302
00260    256*           TA=-T                              @ XL BACK TO ORIGINAL SPOT    901A3490    000305
00261    257*     3500  CALL SPC (XL,UL,TA)                @                             V02-08F     000310
00262    258*           CALL YNN (XL)                      @ CHECK BOUNDARIES HERE       901A3510    000314
00263    259*           IF (SVMIN) 3530,3530,3600          @ NON-VIOLATING ?             901A3520    000317
```

Listing 3. - Continued.

```
00266   3530 IF (XL(4)) 3570,3570,3540          @ NO, CAN WE BACK UP FURTHER ?901A3530   000322
00271   3540 TN=TN-TA                           @ KEEP TRACK OF TIME SPANS    901A3540   000325
00272        T=T-TA                             @                             901A3550   000330
00273        GO TO 3500                         @ GO BACK IT UP               901A3560   000333
00274   3570 CALL ERROR2 (901,2)                @ CAN'T SHAKE BOUNDARY        901A3570   000335
00275        CALL STIAES (I1,IDU)               @ THIS TURKEY GOES TO SHEOL              000340
00276        GO TO 221                          @                             901A3590   000344
00277   3600 TQ=0.0                             @ Q IMPLIES A POINT NOT IN    901A3600   000346
00300        DO 3620 J=1,4                      @  VIOLATION                  901A3610   000351
00303        UQ(J)=UL(J)                        @                             V02-08F    000351
00304   3620 XQ(J)=XL(J)                        @                             901A3620   000352
00306        IE3F=0                             @ ERROR 3 ABORT FLAG          901A3630   000355
00307        DO 3790 J=1,15                     @ BISECTION LOOP              901A3640   000361
00312        IF (ABS(TN-TQ)-TE) 3800,3660,3660  @ CLOSE ENOUGH ?              901A3650   000361
00315   3660 TA=0.5*(TN+TQ)                     @ NO, BISECT AGAIN            901A3660   000367
00316        CALL SPC (XA,UA,TA)                @ FIND THAT POINT IN SPACE    V02-08F    000373
00317        CALL YNK (XA)                      @ TEST THE BOUNDARIES         901A3680   000400
00320        IF (IDMIN) 3710,3710,3700          @ DID ALL TRUNCATE ?          901A3690   000403
00323   3700 IE3F=IDMIN                         @ NO, FLAG AN ACTIVE BOUNDARY 901B3700   000406
00324   3710 IF (SVMIN) 3760,3760,3720          @ A NON-VIOLATING POINT ?     901A3710   000410
00327   3720 TQ=TA                              @ YES, REPLACE Q POINT        901A3720   000412
00330        DO 3740 K=1,4                      @                             901A3730   000417
00333        UQ(K)=UA(K)                        @                             V01-08F    000417
00334   3740 XQ(K)=XA(K)                        @                             901A3740   000420
00336        GO TO 3790                         @                             901A3750   000423
00337   3760 TN=TA                              @ NO, REPLACE N POINT         901A3760   000425
00340        DO 3780 K=1,4                      @                             901A3770   000431
00343        UN(K)=UA(K)                        @                             V02-08F    000431
00344   3780 XN(K)=XA(K)                        @                             901A3780   000432
00346   3790 CONTINUE                           @                             901A3790   000440
00350   3800 IF (IE3F) 3810,3810,3840           @ CONSTRUCTION PROBLEM ?      901A3800   000440
00353   3810 CALL ERROR2 (901,3)                @ YES, REPORT IT              901A3810   000442
00354        CALL STIAES (I1,IDU)               @ FORGET THIS GUY                        000446
00355        GO TO 221                          @                             901A3830   000452
00356   3840 IF (IDMIN) 3850,3850,3880          @ ALGORITHM DIDN'T TRACK ?    901A3840   000454
00361   3850 CALL ERROR2 (901,4)                @ YES, VERY ODD               901A3850   000456
```

Listing 3. - Continued.

```
00362             CALL STIAES (I1,IDU)          @ INTO THE BLACK HOLE WITH HIM                000462
00363             GO TO 221                     @                               901A3870      000466
00364       3880  IDB=IDMIN                     @ BOUNCE OFF THIS BOUNDARY      901A3880      000470
00365             TLEFT=TLEFT+T-TQ              @ TIME TO GO AFTER BOUNCE       901A3890      000471
00366             T=TQ                          @ TIME TO BOUNCE                901A3920      000475
00367             GO TO 101                     @TRY SHORTER TIME SPAN          90103960      000477
00370        219  CONTINUE                      @END TIME SUB-INCREMENT LOOP    90103970      000505
00372             DO 220 I=1,3                  @ RECORD NEW POSITION AND                     000505
00375             CALL STS (I1,I,XL(I))         @  VELOCITY RESULTS IN                        000505
00376        220  CALL STS (I1,I+3,UL(I))       @  ASSIGNED SLOTS                             000514
00400             CALL STIAES (I1,IFZ)          @ RECORD FINAL FLOW ZONE                      000530
00401        221  CONTINUE                      @END ELEMENT BY ELEMENT LOOP    90104010      000536
00403             RETURN                                                        90104140      000536
00404             SUBROUTINE SPC (Y,Z,TI)       @                               V02-08F       000567
00404       C+                                                                  901A4160      000567
00404       C     LOADS VECTOR Y WITH AEROELEMENT POSITION AT TIME 'TI' RELATIVE    901A4170      000567
00404       C     TO 'XL'.  CALCULATES VELOCITY AT Y AND PLACES IT IN Z.            V02-08F       000567
00404       C-                                                                  901A4190      000567
00407             REAL Y(4),Z(3),TI             @                               V02-08F       000567
00410             DO 13 N=1,3                                                   901A4210      000567
00413             Y(N)=XL(N)+(TI*UL(N))+(0.5*TI*TI*AL(N))                       V02-08F       000603
00414        13   Z(N)=UL(N)+(TI*AL(N))                                         V02-08F       000611
00416             Y(4)=XL(4)+TI                                                 901A4230      000616
00417             RETURN                                                        901A4240      000621
00420             SUBROUTINE YNM (Y)                                            901A5000      000652
00420       C+                                                                  901A5010      000652
00420       C     1) IDENTIFIES FLOW ZONE OF SPACE-TIME POINT 'Y' AND UPDATES   901A5020      000652
00420       C        FLOW ZONE OF AEROELEMENT 'I1' IF NECESSARY.                901A5030      000652
00420       C     2) CHECKS ALL APPROPRIATE BOUNDARIES TO PRODUCE 'IDMIN'/'SVMIN'.  901A5040      000652
00420       C        DOES NOT CONSIDER FOR 'IDMIN'/'SVMIN' BOUNDARIES THAT ARE  901A5050      000652
00420       C        SAFE BY TRUNCATION.                                        901A5060      000652
00420       C     3) DISCONTINUES SEARCH IF 'SVMIN' GOES NEGATIVE.              901A5070      000652
00420       C-                                                                  901A5080      000652
00423             REAL Y(4)                                                     901A5090      000652
00423       C+                                                                                000652
00423       C     CAUTION *** THIS ROUTINE DOES NOT DETECT THE 'ZONE NOT FOUND'               000652
```

Listing 3. - Continued.

```
 0027   NERR2$
 0030   NWDU$
 0031   NIO2$
 0032   NIO1$
 0033   NERR3$

STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

 0001    000763 1L      0001    000034 107L     0001    000053 110L     0001    000057 111L     0001    000063 115L
 0001    000070 117L    0000    000026 119F     0001    000124 122L     0000    000051 123F     0001    000140 124L
 0001    000207 134L    0001    000224 139L     0001    000235 140L     0001    000243 142L     0001    000246 144L
 0001    000251 145L    0001    000264 148L     0001    000061 152G     0001    000335 175L     0001    000350 179L
 0001    000364 181L    0001    000365 182L     0001    000371 184L     0001    000377 186L     0001    000404 187L
 0001    000415 189L    0001    000760 2L       0001    000134 201G     0001    000450 205L     0001    000470 209L
 0001    000146 210G    0001    000171 224G     0001    000516 225L     0001    000522 226L     0001    000524 227L
 0001    000526 228L    0001    000212 231G     0001    000547 253L     0001    000563 276L     0001    000755 3L
 0001    000621 300L    0001    000667 308L     0001    000673 325L     0001    000702 326L     0000    000057 328F
 0001    000727 330L    0001    000540 356G     0001    000753 4L       0001    000624 402G     0001    000735 5L
 0001    000732 6L      0001    001022 995L     0000    000100 997F     0001    000774 999L     0005 I 000000 ELNK
 0006 I 000000 FZ       0000 I 000005 I         0000 I 000021 IC        0000 I 000020 ICH       0000 I 000006 IDA
 0000 I 000003 IE       0000 I 000025 IEQ       0000 I 000022 IEZ       0003 I 000003 IHEAD     0000 I 000011 IHO
 0000 I 000012 IHN      0000 I 000007 IMO       0000 I 000014 IMS       0000    000132 INJP$    0000 I 000000 IOC
 0013 I 000000 IPLZN    0000 I 000016 IPO       0000 I 000017 IP1       0000 I 000024 IP2       0000 I 000013 ISE
 0000 I 000010 IYE      0000 I 000015 IZC       0000 I 000004 IZS       0000 I 000023 J         0003 I 000002 LXTETH
 0003 I 000001 MXTETH   0003 I 000000 NTETH     0004 I 000000 NZN


00101      1*            SUBROUTINE RESO2 (IZL,IZH,IOP,IDL)                                   000003
00101      2*       C+                                                                        000003
00101      3*       C                                                                         000003
00101      4*       C    RESO2        ****** A SUBROUTINE FOR CASPER ******                   000003
00101      5*       C                 AUTHOR      WILLIAM HENRY JONES                         000003
00101      6*       C                 V01-00      19 FEB 80                                   000003
```

Listing 4. - Continued.

```
00101     C               V01-00A    15 SEP 80    FUNCTION TYPE STATEMENTS              000003
00101     C               V01-00B    04 JUN 82    DEMO AND DEBUG MSGS     1002V01-00B   000003
00101     C               V01-00C    13 JUL 82    TYPO                        V01-00C   000003
00101     C                                                                             000003
00101     C                                                                             000003
00101     C        DESCRIPTION ******                                                   000003
00101     C                                                                             000003
00101     C        THIS ROUTINE PERFORMS THE FLOW ZONE AEROELEMENT LINKAGE              000003
00101     C        PURIFICATION PASS.  THIS PROCESS SEARCHES THROUGH EACH FLOW          000003
00101     C        ZONE'S RESIDENT AEROELEMENT LINKAGE LOOKING FOR AEROELEMENTS         000003
00101     C        THAT ARE NOT RESIDENT IN THAT FLOW ZONE.  ANY SUCH AEROELEMENTS      000003
00101     C        THAT ARE FOUND ARE REMOVED FROM THAT FLOW ZONE'S LINKAGE AND         000003
00101     C        CONSIGNED TO THE LINKAGE OF THE FLOW ZONE OF WHICH THEY ARE          000003
00101     C        A RESIDENT.                                                          000003
00101     C                                                                             000003
00101     C        CONSIGNMENT TO THE FLOW ZONE FOLLOWS ONE OF TWO PROCEDURES.          000003
00101     C        IF THE DESTINATION FLOW ZONE IS IN RANGE, THE AEROELEMENT IS         000003
00101     C        IMMEDIATELY LINKED INTO THE TAIL OF THAT FLOW ZONE'S LINKAGE.        000003
00101     C        IF THE FLOW ZONE IS NOT IN RANGE, THE AEROELEMENT IS TETHERED        000003
00101     C        IN A LOCAL LINKAGE.  AT AN APPROPRIATE TIME THIS LOCAL               000003
00101     C        LINKAGE IS REPORTED TO PAX FOR LINKING INTO THE DESTINATION          000003
00101     C        LINKAGE.                                                             000003
00101     C                                                                             000003
00101     C        SOME OPTIONAL REPORTS FOR CROSS REFERENCING PURPOSES MAY BE          000003
00101     C        REQUESTED AS SPECIFIED BELOW.                                        000003
00101     C                                                                             000003
00101     C        IOP          REPORT                                                  000003
00101     C        ------       ------------------------------                          000003
00101     C        1            NO ADDITIONAL INFORMATION IS REPORTED                   000003
00101     C                                                                             000003
00101     C        2            FOR I EQUAL TO 1, 2, OR 3, IF THE FLOW ZONE OF          000003
00101     C                     RESIDENCE FOR A PARTICULAR AEROELEMENT IS THE           000003
00101     C                     SAME AS IDL(I), THEN THAT AEROELEMENT'S ID IS           000003
00101     C                     APPENDED TO A LIST THAT IS ULTIMATELY ASSOCIATED        000003
00101     C                     WITH IDL(I) IN A REPORT TO PAX.                         000003
00101     C                                                                             000003
```

Listing 4. - Continued.

```
00101   C   3             FOR I EQUAL TO 1, 2, OR 3, IF THE FLOW ZONE OF            000003
00101   C                 RESIDENCE FOR A PARTICULAR AEROELEMENT IS THE             000003
00101   C                 SAME IS IDL(I), THEN THAT AEROELEMENT'S ID IS             000003
00101   C                 APPENDED TO A LIST THAT IS ULTIMATELY ASSOCIATED          000003
00101   C                 WITH IDL(1) IN A REPORT TO PAX.                           000003
00101   C                                                                           000003
00101   C   4             THE SAME AS 3.  ADDITIONAL ACTION IS TAKEN.  THE          000003
00101   C                 FLOW ZONE OF RESIDENCE FOR THE AEROELEMENT IS             000003
00101   C                 CHANGED TO IDL(1) AND LINKAGE PURIFICATION PROCEEDS       000003
00101   C                 ON THE REVISED VALUE.                                     000003
00101   C                                                                           000003
00101   C                                                                           000003
00101   C   GENERAL DATA BASE ******                                                000003
00101   C                                                                           000003
00101   C   IPLZN( ,4)    LINKAGE HEAD POINTER                                      000003
00101   C   IPLZN( ,5)    LINKAGE COUNT                                             000003
00101   C   IPLZN( ,6)    LINKAGE TAIL POINTER                                      000003
00101   C                                                                           000003
00101   C   ELNK( ,1)     NEXT ELEMENT POINTER                                      000003
00101   C   ELNK( ,2)     FOR THIS ROUTINE ONLY, FOR HEAD ELEMENT                   000003
00101   C                  ONLY, ORIGINAL LINKAGE COUNT BEFORE                      000003
00101   C                  PURIFICATION; OTHERWISE, SCRATCH                         000003
00101   C   ELNK( ,3)     SCRATCH                                                   000003
00101   C                                                                           000003
00101   C                                                                           000003
00101   C   COMMON TETHER DATA BASE ******                                          000003
00101   C                                                                           000003
00101   C   LTETH         NUMBER OF TETHER HEADS                                    000003
00101   C   NTETH         NUMBER OF HIGHEST TETHER HEAD IN USE                      000003
00101   C   MXTETH        NUMBER OF LONGEST TETHER HEAD IN USE                      000003
00101   C   LXTETH        LENGTH OF LONGEST TETHER IN USE                           000003
00101   C   IHEAD(1, )    FLOW ZONE ID ASSOCIATED WITH TETHER                       000003
00101   C   IHEAD(2, )    POINTER TO FIRST ELEMENT IN TETHER                        000003
00101   C   IHEAD(3, )    LENGTH OF TETHER                                          000003
00101   C                                                                           000003
00101   C   ELNK( ,1)      POINTER TO NEXT ELEMENT                                  000003
```

Listing 4. - Continued.

```
00101   C     ELNK( ,2)   LENGTH OF TETHER (FIRST ELEMENT ONLY)                   000003
00101   C     ELNK( ,3)   POINTER TO LAST ELEMENT (FIRST ELEMENT ONLY)            000003
00101   C                                                                         000003
00101   C                                                                         000003
00101   C-                                                                        000003
00103         PARAMETER OURID=982                                                 000003
00104         PARAMETER IOCHX=100                                                 000003
00105         INCLUDE TETHP                                                       000003
00111         INCLUDE PGSDEF         @                              V01-00B       000003
00114         INTEGER IOC(3),IDL(3),ELNK,FZ                                       000003
00115         COMMON /NZNC/NZN                                                    000003
00116         IE=1                                                                000003
00117         CALL TETHI                                                          000005
00120         IF (IZL) 1,1,101                @ ERROR CHECK FLOW ZONE             000007
00123   101   IF (IZL-NZN) 102,102,1          @  RANGE LIMITS                     000012
00126   102   IF (IZH) 2,2,103                @                                   000015
00131   103   IF (IZH-NZN) 104,104,2          @                                   000020
00134   104   IZS=1                           @ SET FLOW ZONE STEP                000023
00135         IF (IZH-IZL) 106,107,107        @  DIRECTION                        000025
00140   106   IZS=-1                          @                                   000031
00141   107   IF (IOP) 3,3,108                @ ERROR CHECK OPTION                000034
00144   108   IF (IOP-4) 109,109,3            @  SELECTION                        000036
00147   109   GO TO (115,110,111,111),IOP     @ OPTION BRANCH                     000041
00150   110   CALL STKCHG (3,1)               @ (OP2) ASK FOR 3 STACKS            000053
00151   111   DO 114 I=1,3                    @ (OP234) INIT COUNTS LIST          000061
00154   114   IOC(I)=0                        @                                   000061
00156   115   CONTINUE                        @                                   000063
00157         CALL TOGSW (PGSDMO+PGSBUG,$117,$124) @ DEMO OR DEBUG ON ?   V01-00B 000063
00160   117   CALL TIMPR (IDA,IMO,IYE,IHO,IMN,ISE,IMS) @ YES, GET TIME    V01-00B 000070
00161         WRITE (6,119) IZL,IZH,IDA,IMO,IYE,IHO,IMN,ISE,IMS @         V01-00B 000100
00174   119   FORMAT (1H0,5X,30HCASPER9.RESO2D (DMO) -- RANGE ,I8,4H TO ,I8,13H V01-00C 000116
00174        1ACCEPTED ON ,J2,1X,A4,J4,4H AT ,2(J2,1H:),J2,1H.,J3) @      V01-00B 000116
00175         CALL TOGSW (PGSBUG,$122,$124)   @ DEBUG ON ?                V01-00B 000116
00176   122   WRITE (6,123) IOP,(IDL(I),I=1,3)    @ YES                   V01-00B 000124
00205   123   FORMAT (1H ,7X,6HIOP = ,I6,7H IDL = ,3(I8)) @               V01-00B 000140
00206   124   CONTINUE                        @                           V01-00B 000140
```

Listing 4. - Continued.

```
00207              DO 229 IZC=IZL,IZH,IZS               @                                 000140
00212              IP0=0                                @ PREVIOUS ELEMENT POINTER        000146
00213              IP1=IPLZN(IZC,4)                     @ CURRENT ELEMENT POINTER         000147
00214              IF (IP1) 228,228,129                 @ IS THERE AN ELEMENT ?           000154
00217      129     ICH=ELNK(IP1,2)                      @ YES, GET COUNT                  000156
00220              IF (ICH) 5,228,131                   @ LEGAL COUNT ?                   000163
00223      131     DO 227 IC=1,ICH                      @ YES                             000165
00226              IEZ=FZ(IP1)                          @ GET ELEMENT'S FLOW ZONE         000171
00227              GO TO (175,134,134,134),IOP          @ BRANCH BY OPTION                000175
00230      134     DO 137 I=1,3                         @ (OP234)                         000212
00233              J=I                                  @                                 000212
00234              IF (IEZ-IDL(I)) 137,139,137          @ FLOW ZONES MATCH ?              000214
00237      137     CONTINUE                             @ NO                              000222
00241              GO TO 175                            @ TOTAL MISS                      000222
00242      139     GO TO (175,144,142,140),IOP          @                                 000224
00243      140     IEZ=IDL(1)                           @ (OP4) REVISE FLOW ZONE          000235
00244              CALL STFZ (IP1,IEZ)                  @                                 000236
00245      142     J=1                                  @ (OP34) INTERPRET AS IDL(1)      000243
00246              GO TO 145                            @                                 000244
00247      144     CALL STKSET (J)                      @ (OP2) GET RIGHT STACK           000246
00250      145     IF (IOC(J)) 6,146,148                @ NEED END-OF-ARGS ?              000251
00253      146     CALL SPSHI (0)                       @ YES                             000255
00254              CALL SPSHI (0)                       @                                 000260
00255      148     CALL SPSHI (IP1)                     @ ID ON TO STACK                  000264
00256              IOC(J)=IOC(J)+1                      @ KEEP ID COUNT                   000271
00257              IF (IOC(J)-IOCMX) 175,151,151        @ ENOUGH TO REPORT ?              000274
00262      151     CALL SPSHI (-IOC(J))                 @ YES, CODE FOR LITERAL STRING    000277
00263              CALL SPSHI (IOC(J))                  @ COUNT OF IDS                    000304
00264              CALL SPSHI (IDL(J))                  @ ZONE OF ASSOCIATION             000311
00265              CALL SPSHI (2)                       @ TWO SINGLE LITERALS             000316
00266              CALL SPSHI (IOC(J)+6)                @ STACK DEPTH                     000321
00267              CALL REQSAF (OURID,2)                @                                 000327
00270              IOC(J)=0                             @ NONE ON STACK NOW               000333
00271      175     IF (IEZ-IZC) 179,176,179             @ IS IT IN THE RIGHT ZONE ?       000335
00274      176     IP0=IP1                              @ YES, STEP TO NEXT               000337
00275              IP1=ELNK(IP0,1)                      @                                 000341
```

Listing 4. - Continued.

```
00276              GO TO 227                       @                                  000346
00277        179   CALL STELNK (IP1,2,0)           @ NO, ZAP ORIGINAL COUNT SLOT      000350
00300              IP2=ELNK(IP1,1)                 @ PULL FROM LINKAGE                000354
00301              IF (IP2) 181,182,182            @                                  000361
00304        181   IP2=0                           @                                  000364
00305        182   IF (IP0) 183,184,186            @                                  000365
00310        183   IP0=0                           @                                  000367
00311        184   CALL STIPLZ (IZC,4,IP2)         @  NEW LINKAGE HEAD                000371
00312              GO TO 187                       @                                  000375
00313        186   CALL STELNK (IP0,1,IP2)         @  JOINT IN LINKAGE MIDDLE         000377
00314        187   IF (IP2) 181,188,189            @                                  000404
00317        188   CALL STIPLZ (IZC,6,IP0)         @  NEW LINKAGE TAIL                000407
00320        189   I=IPLZN(IZC,5)-1                @  ONE LESS ELEMENT                000415
00321              CALL STIPLZ (IZC,5,I)           @                                  000422
00322              IF (IEZ-IZL) 225,201,201        @ IS CORRECT ZONE IN RANGE ?       000427
00325        201   IF (IEZ-IZH) 202,202,225        @                                  000433
00330        202   I=IPLZN(IEZ,6)                  @ YES, POINT TO ITS TAIL           000437
00331              IF (I) 204,205,209              @ IS THERE AN OLD TAIL ?           000444
00334        204   I=0                             @                                  000446
00335        205   CALL STIPLZ (IEZ,4,IP1)         @ NO, START A WHOLE NEW LINK       000450
00336              CALL STIPLZ (IEZ,5,1)           @                                  000454
00337              CALL STIPLZ (IEQ,6,IP1)         @                                  000461
00340              GO TO 226                       @                                  000466
00341        209   CALL STELNK (I,1,IP1)           @ YES, ADD THIS TO TAIL            000470
00342              CALL STIPLZ (IEZ,6,IP1)         @                                  000474
00343              I=IPLZN(IEZ,5)+1                @                                  000501
00344              CALL STIPLZ (IEZ,5,I)           @                                  000507
00345              GO TO 226                       @                                  000514
00346        225   CALL TETHA (IP1,IEZ)            @ TETHER ELEMENT LOCALLY           000516
00347        226   IP1=IP2                         @ ADJUST NEXT ELEMENT POINTER      000522
00350        227   CONTINUE                        @ END OF ELEMENT LOOP              000526
00352        228   CALL CHKTIM (IZL,IZH,IZC)       @ KEEP AN EYE ON THE TIME          000526
00353        229   CONTINUE                        @ END OF ZONE LOOP                 000540
00355              DO 253 I=1,LTETH                @ FLUSH ANY RESIDUAL TETHERS       000540
00360              IF (IHEAD(2,I)) 253,253,252     @                                  000540
00363        252   CALL TETHF (I)                  @                                  000543
```

Listing 4. - Continued.

```
00364    253   CONTINUE                        @                             000551
00366          GO TO (325,300,276,276),IOP     @ BRANCH BY OPTIONS           000551
00367    276   IF (IOC(1)) 325,325,277         @ (OP34) - ANY TO REPORT ?    000563
00372    277   CALL SPSHI (-IOC(1))            @ YES, LIT. STRING CODE       000565
00373          CALL SPSHI (IOC(1))             @ COUNT OF IDS                000572
00374          CALL SPSHI (IDL(1))             @ ZONE OF ASSOCIATION         000575
00375          CALL SPSHI (2)                  @ TWO SINGLE LITERALS         000602
00376          CALL SPSHI (IOC(1)+6)           @ STACK DEPTH                 000605
00377          CALL REQSAF (OURID,2)           @ TRANSMIT                    000613
00400          GO TO 325                       @                             000617
00401    300   DO 308 J=1,3                    @ (OP2) LOOK AT EACH          000624
00404          IF (IOC(J)) 300,308,302         @ ANY TO REPORT ?             000624
00407    302   CALL STKSET (J)                 @ YES, GET RIGHT STACK        000627
00410          CALL SPSHI (-IOC(J))            @                             000632
00411          CALL SPSHI (IOC(J))             @                             000637
00412          CALL SPSHI (IDL(J))             @                             000644
00413          CALL SPSHI (2)                  @                             000651
00414          CALL SPSHI (IOC(J)+6)           @                             000654
00415          CALL REQSAF (OURID,2)           @                             000662
00416    308   IOC(J)=0                        @                             000667
00420          CALL STKOLD                     @ BACK TO ORIGINAL STACKS     000672
00421    325   CALL TOGSW (PGSDMO+PGSBUG,$326,$330) @ NEED CLOSING MESSAGE ?     V01-00B   000675
00422    326   CALL TIMPR (IDA,IMO,IYE,IHO,IMN,ISE,IMS) @ YES                    V01-00B   000702
00423          WRITE (6,328) IDA,IMO,IYE,IHO,IMN,ISE,IMS @                       V01-00B   000712
00434    328   FORMAT (1H ,5X,49HCASPER9.RESO2D (DMO) -- SUCCESSFUL COMPLETION ONV01-00B   000727
00434         1 ,J2,1X,A4,J4,4H AT ,2(J2,1H:),J2,1H.,J3) @                       V01-00B   000727
00435    330   RETURN                          @ DONE                            V01-00B   000727
00435    C+                                                                  000727
00435    C     ERROR REPORTING                                               000727
00435    C-                                                                  000727
00436    6     IE=IE+1      @ OPTION COUNT WAS NEGATIVE                      000732
00437    5     IE=IE+1      @ ELEMENT COUNT ILLEGAL FOR FLOW ZONE            000735
00440          IE=IE+1                @ ERR 4 NOT USED                       000737
00441          GO TO (3,4,3,3),IOP    @ TO OLD STACK CONFIGURATION ?         000741
00442    4     CALL STKOLD            @ YES                                  000753
00443    3     IE=IE+1      @ ILLEGAL OPTIONS SELECTED                       000755
```

Listing 4. - Continued.

```
00444    223*    2     IE=IE+1      @ FLOW ZONE HIGH LIMIT OUT OF RANGE                        000760
00445    224*    1     CONTINUE     @ FLOW ZONE LOW LIMIT OUT OF RANGE                         000763
00446    225*          CALL ERROR2 (OURID,IE)  @                                               000763
00447    226*          CALL TOGSW (PGSDMO+PGSBUG,$999,$995) @ NEED A MESSAGE ?        V01-00B  000766
00450    227*    999   CALL TIMPR (IDA,IMO,IYE,IHO,IMN,ISE,IMS) @ YES                 V01-00B  000774
00451    228*          WRITE (6,997) IE,IDA,IMO,IYE,IHO,IMN,ISE,IMS @                 V01-00B  001004
00463    229*    997   FORMAT (1H ,5X,30HCASPER9.RES02D (DMO) -- ERROR ,J3,4H ON ,J2,1X,AV01-00B  001022
00463    230*         14,J4,4H AT ,2(J2,1H:),J2,1H.,J3)    @                          V01-00B  001022
00464    231*    995   RETURN                             @ DONE BADLY                V01-00B  001022
00465    232*          END                     @                                               001103
END FOR
>
```

Listing 4. - Concluded.

```
@FOR,MS CASPER9.TETHFD
FOR   4R1   E -01/13/83-14:18:09 (3,)
>@EOF

   SUBROUTINE TETHF     ENTRY POINT 000161

   STORAGE USED: CODE(1) 000166; DATA(0) 000073; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003   TETHC  000457

   EXTERNAL REFERENCES (BLOCK, NAME)

    0004   ELNK
    0005   TOGSW
    0006   TIMPR
    0007   SPSHI
    0010   DBAF
    0011   REQSAF
    0012   ERROR2
    0013   WALKB
    0014   NWDU$
    0015   NIO2$
    0016   NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000072 108L     0000   000034 109F     0001   000124 110L     0001   000133 111L     0001   000137 112L
    0001   000147 115L     0001   000151 116L     0001   000010 95L      0000   000014 97F      0001   000035 99L
    0004 I 000000 ELNK     0000 I 000007 I        0000 I 000000 IDA      0003 I 000003 IHEAD    0000 I 000003 IHO
    0000 I 000004 IHN      0000 I 000001 IKO      0000 I 000006 IMS      0000   000065 INJP$    0000 I 000005 ISE
```

Listing 5. - Conflicted-task request routine.

```
0000 I 000002 IYE       0000 I 000010 J         0000 I 000011 K         0000 I 000012 L         0003 I 000002 LXTETH
0000 I 000013 M         0003 I 000001 MXTETH    0003 I 000000 NTETH


00101        SUBROUTINE TETHF (IP)                                                          000002
00101  C+                                                                                   000002
00101  C                                                                                    000002
00101  C     TETHF      ****** A SUBROUTINE FOR CASPER ******                               000002
00101  C                AUTHOR     WILLIAM HENRY JONES                                      000002
00101  C                V01-00     19 FEB 80                                                000002
00101  C                V01-00A    04 JUN 82     DEBUG MESSAGES         1002V01-00A         000002
00101  C                V01-00B    29 JUN 82     TYPO                   1005V01-00B         000002
00101  C     1007       V01-00C    14 JUL 82     DATA BASE FLUSH ADDED      V01-00C         000002
00101  C                                                                                    000002
00101  C                                                                                    000002
00101  C     DESCRIPTION ******                                                             000002
00101  C                                                                                    000002
00101  C     TRANSMITS THE INFORMATION OF TETHER 'IP' TO PAX AND RE-INITIALIZES             000002
00101  C     THE TETHER DATA.                                                               000002
00101  C                                                                                    000002
00101  C                                                                                    000002
00101  C-                                                                                   000002
00103        PARAMETER OURID=979              @ CASPER CATALOG ID                           000002
00104        INCLUDE TETHP                    @                                             000002
00110        INCLUDE PGSDEF                   @                         V01-00A             000002
00113        INTEGER ELNK                     @                         V01-00A             000002
00114        CALL TOGSW (PGSBUG,$95,$99)      @ DEBUG ON ?              V01-00A             000002
00115  95    CALL TIMPR (IDA,IMO,IYE,IHO,IMN,ISE,IMS) @ YES, NOTE THE TIME   V01-00A         000010
00116        WRITE (6,97) IDA,IMO,IYE,IHO,IMN,ISE,IMS @ PRINT HEADING MSG    V01-00A         000020
00127  97    FORMAT (1H0,5X,41HCASPER9.TETHFD (BUG) -- FLUSH INVOKED ON ,J2,1X,V01-00A       000035
00127       1A4,J4,4H AT ,2(J2,1H:),J2,1H.,J3)  @                       V01-00A             000035
00130  99    CONTINUE                         @                         V01-00A             000035
00131        I=IHEAD(2,IP)                    @ POINT TO ELEMENT                            000035
00132        IF (I) 111,111,102               @ IS THERE AN ELEMENT ?                       000036
```

Listing 5. - Continued.

```
102   J=IHEAD(1,IP)                         @ YES, GET FLOW ZONE ID                000040
      CALL SPSHI (0)                        @                                      000042
      CALL SPSHI (0)                        @                                      000045
      CALL SPSHI (I)                        @ ELEMENT POINTER                      000050
      CALL SPSHI (J)                        @ FLOW ZONE                            000053
      CALL SPSHI (2)                        @                                      000056
      CALL SPSHI (5)                        @                                      000061
      CALL TOGSW (PGSBUG,$108,$110)         @ DEBUG ON ?                 V01-00A   000064
108   K=ELNK(I,1)                           @ YES, GET HEAD ELEMENT'S    V01-00B   000072
      L=ELNK(I,2)                           @  INFO                      V01-00B   000076
      M=ELNK(I,3)                           @                            V01-00B   000103
      WRITE (6,109) IP,I,J,K,L,M            @                            V01-00A   000110
109   FORMAT (1H ,7X,16HFLUSHING TETHER ,I6,4X,13HHEAD ELEMENT ,I8,4X,10V01-00A   000124
     1HFLOW ZONE ,I8,/,1H ,9X,7HELNK = ,3(I12,2X)) @                     V01-00A   000124
110   CONTINUE                              @                            V01-00A   000124
      CALL DBAF                             @ ASSURE SHARING OF DATA     V01-00C   000124
      CALL REQSAF (OURID,1)                 @                                      000125
      GO TO 112                             @                                      000131
111   CALL ERROR2 (OURID,1)                 @ IS ERROR TO FLUSH NOTHING            000133
112   IHEAD(1,IP)=0                         @ ZAP HEAD                             000137
      IHEAD(2,IP)=0                         @                                      000137
      IHEAD(3,IP)=0                         @                                      000140
      CALL TOGSW (PGSBUG,$115,$116)         @ DEBUG ON ?                 V01-00A   000141
115   CALL WALKB                            @ YES, CONCLUDE WITH WALKBACK V01-00A  000147
116   CONTINUE                              @                            V01-00A   000151
      RETURN                                @                                      000151
      END                                   @                                      000165
END FOR
>
```

Listing 5. - Concluded.

1. Report No. NASA TP-2179	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle PARALLEL, ASYNCHRONOUS EXECUTIVE (PAX): SYSTEM CONCEPTS, FACILITIES, AND ARCHITECTURE		5. Report Date June 1983
		6. Performing Organization Code 505-40-6A
7. Author(s) William H. Jones		8. Performing Organization Report No. E-1584
		10. Work Unit No.
9. Performing Organization Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Paper
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The Parallel, Asynchronous Executive (PAX) is a software operating system simulation that allows many computers to work on a single problem at the same time. PAX is currently implemented on a UNIVAC 1100/42 computer system. Independent UNIVAC runstreams are used to simulate independent computers. Data are shared among independent UNIVAC runstreams through shared mass-storage files. PAX has achieved the following: (1) applied several computing processes simultaneously to a single, logically unified problem; (2) resolved most parallel processor conflicts by careful work assignment; (3) resolved by means of worker requests to PAX all conflicts not resolved by work assignment; (4) provided fault isolation and recovery mechanisms to meet the problems of an actual parallel, asynchronous processing machine. Additionally, one real-life problem has been constructed for the PAX environment. This is CASPER, a collection of aerodynamic and structural dynamic problem simulation routines. CASPER is not discussed in this report except to provide examples of parallel-processing techniques.

17. Key Words (Suggested by Author(s)) Parallel processing; Distributed processing; Fault tolerant processing; Parallel, asynchronous processing	18. Distribution Statement Unclassified - unlimited; STAR Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	59	A04

* For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1983