NASA Technical Memorandum 89837

Automating the Parallel Processing of Fluid and Structural Dynamics Calculations

Dale J. Arpasi and Gary L. Cole
Lewis Research Center
Cleveland, Ohio

Prepared for the
1987 Summer Computer Simulation Conference
sponsored by the Society for Computer Simulation
Montreal, Canada, July 27-30, 1987

NASA

AUTOMATING THE PARALLEL PROCESSING OF FLUID AND STRUCTURAL DYNAMICS CALCULATIONS

Dale J. Arpasi and Gary L. Cole
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

ABSTRACT

The NASA Lewis Research Center is actively involved in the development of expert system technology to assist users in applying parallel processing to computational fluid and structural dynamic analysis. The goal of this effort is to eliminate the necessity for the physical scientist to become a computer scientist in order to effectively use the computer as a research tool.

Programming and operating software utilities have previously been developed to solve systems of ordinary nonlinear differential equations on parallel scalar processors. Current efforts are aimed at extending these capabilities to systems of partial differential equations, that describe the complex behavior of fluids and structures within aerospace propulsion systems. This paper presents some important considerations in the redesign, in particular, the need for algorithms and software utilities that can automatically identify data flow patterns in the application program and partition and allocate calculations to the parallel processors. A library-oriented multiprocessing concept for integrating the hardware and software functions is described.

INTRODUCTION

Reductions in calculation times through parallel processing can be negated by inefficiencies in program development and execution. The complexities of data-flow detection, code allocation, data transfer control, computation, and computation synchronization, complicate the development of efficient parallel processing programs. Code development for parallel processing requires expertise in data-flow analysis and in the mechanics of the computing system in order to show advantage over serial computation and to justify the costs associated with parallel processing.

It is unrealistic to require this expertise of the general user who merely wants his program to run faster. A better approach is to provide an intelligent interface between the user and the parallel processing system. Ideally this "expert" software will automate program development, control program execution, and assist in managing and analyzing results through "natural" language instructions from the user. This approach allows the user to concentrate on the details of his own discipline, resulting in efficient and effective use of the parallel processing system. An added benefit of this approach is that it promotes technology transfer by raising the level of man-machine communication beyond the current Fortran/DOS norm.

The development of expert software to provide engineering-level access to parallel processing systems is an on-going effort at NASA Lewis. The development of first-generation software for programming and operating real-time propulsion simulations on parallel processors is documented in Arpasi 1986; Arpasi 1985a; Arpasi 1985b; Cole 1985; Cole 1984. Some related efforts, sponsored by NASA Lewis are described in Makoui and Karplus 1983; Feyock and Collins 1986. Current emphasis is on extending these software concepts and designs to produce a highly automated environment for constructing and utilizing solvers for fluid and structural dynamic analysis and for using them as elements of large system simulators. This will require the development of a knowledge base, relational algorithms, and software utilities to perform the following functions:

(1) Converting a differential equation model into a high-level, serial computational model that lends itself to parallel processing.

(2) Developing a parallel statement of the model from the serial statement.

(3) Generating executable load modules and executing them on the multiprocessing facility in such a way that they run as fast as possible using a minimum of resources.

(4) Providing extensive library and documentation utilities to insure the usefulness of programs and results.

Work has begun on the design and implementation of the programming and operation utilities (2 and 3 above). Results of these efforts will define interface requirements for and guide the development of the other utilities.

The following sections present some important considerations regarding the development of the expert programming and operational software. The importance of data-flow analysis to detecting parallelism in the computational model is described. Multidimensional calculation grids are discussed and a data-flow technique called propagation vectoring is described as a way of detecting and controlling the propagation of calculations over the grid. Finally, a concept for integrating all of the hardware and software tools into a library-oriented multiprocessing environment is presented.

DATA-FLOW CONSIDERATIONS

Understanding the data-flow in complex, multidimensional simulations is critical to producing an efficient parallel processing program. In this section, a technique for data-flow identification, propagation, and packing is discussed.

We start by defining a "code unit" to be a number of equivalence statements, which will be called "elements," bound together through causal relationships (each producing one result from any number of argu-

ments). For example, the following can be considered a code unit

$$X1(i)=f1[X4(i-1)]$$

$$X2(i)=f2[X4(i-1)]$$

$$X3(i)=f3[X2(i)] \qquad (1)$$

$$X4(i)=f4[X2(i)]$$

$$X5(i)=f5[X1(i),X3(i),X4(i)]$$

where i represents the current calculation, and i-1 the prior calculation of its elements (X1 ... X5). As shown in Fig. 1(a), the order of the calculations is determined by the argument-result relationships. That is, an element cannot be computed until the elements producing its arguments are computed. One must also take into account the fact that calculation of a code element requires a specific amount of time. This time, of course, depends upon the capabilities of the computer doing the calculation. In the figure, hypothetical calculation times are represented by the height of the unshaded area of each rectangle. Shaded areas represent a slot of time in which a calculation of an element may occur without delaying the calculation of subsequent elements. Thus the number of data flow paths may be minimized by judicious movement and combining (packing) of elements without increasing the calculation time of the code unit. Then, the resultant grouping of elements (paths) may each be assigned to a processor for parallel calculation. As shown in Fig. 1(a), the calculations for Eq. (1) can be packed into two parallel paths.

For the following discussion, the term "solver" is used to describe a multiprocessor dedicated to a specific code unit calculation. A dual-processor system, dedicated to calculating the two parallel paths in the example can be considered a solver of the code unit in the i^{th} calculation cycle. The solver accepts a value for $X_4(i-1)$, computes the values $X_1(i)...X_5(i)$ and returns these values after the calculation interval (here, the time to calculate f_2, f_4, and f_5).

This solver could be used repetitively to compute successive cycles as shown in Fig. 1(b). This approach may be viewed as a do-loop

$$\text{For } i=1 \ldots 2 \text{ do solver,} \qquad (2)$$

assuming that the appropriate data transfers take place. The calculation time for two cycles is twice the calculation time of a single cycle.

Figure 1(c) shows an alternative two-cycle solver. In this approach the elements for both cycles are treated as a single code unit. The data-flow paths can be packed into a dual-processor solver as shown in Fig. 1(c) so as to require less calculation time than the repetitive approach (i.e., $f_2(i+1)$ overlaps $f_5(i)$). It should be noted that this approach requires that the entire code unit reside on each processor. These examples demonstrate that direct parallelization of a repetitive calculation may not result in a minimum-time solution. Indeed, it is often necessary to expand loop calculations into a large set of equations in order to develop a minimum-time solver.

Once a solver has been developed, it can be replicated. These multiple identical solvers may then be used to parallelize the repetitive calculation of the solver's results (e.g., other do loops). The remainder of this section provides a discussion of this approach to parallel processing. Repetitive calculation is viewed as propagation of the solver over a multidimensional calculation grid (the dimension corresponding to the arrayness of the solver's variables). An approach to identifying this propagation is presented. Finally the use of control statements to establish the bounds and boundary values of the calculation grid is discussed.

Consider the following example of a three-dimensional calculation:

$$X(i,j,k)=F[X(i-1,j,k),X(i,j,k-1)], \qquad (3)$$

where i,j,k are the coordinates of a calculation gridpoint; X(i,j,k) is the result at the gridpoint; and F is the functional relationship between that result and the results at other gridpoints. Calculation and transfer of X(i-1,j,k) enables X(i,j,k) as does X(i,j,k-1). The calculation of X(i,j,k) is triggered when it has been enabled by each argument. The sequencing of gridpoint calculations can be defined in terms of the "propagation vectors" [1,0,0] and [0,0,1], which result from the roots of the argument coordinates:

$$i-1=0,j=0,k=0 \quad \text{and} \quad i=0,j=0,k-1=0. \qquad (4)$$

Since there are two propagation vectors for Eq. (3), each gridpoint requires two enablements to trigger its calculation. An enablement occurs when the gridpoint's coordinates matches the sum of another triggered gridpoint's coordinates and one of the propagation vectors.

For example, in the 4 by 4 by 4 calculation grid shown in Fig. 2, calculation of points (2,3,3) and (3,3,2) trigger calculation of point (3,3,3). The calculation of (3,3,3) would occur in the fifth calculation cycle of the solver. Once all gridpoints, which can be calculated at a specific time, have been identified, and their arguments from past cycle calculations become available, those gridpoints can be allocated to solvers and the new calculations begun. The calculation of the solver (3) will proceed according to the two propagation vectors until all 64 gridpoints have been calculated. The propagation vectors therefore determine the number of gridpoints that can be calculated each cycle. For the example, these are: 4,8,12,16,12,8,4 (see Fig. 2). Therefore if a processor is assigned to each gridpoint which can be calculated in parallel, 16 processors would be required to calculate the 64 gridpoints in the minimum 7 calculation cycles. Through proper use of propagation vectoring, it may be possible to set up the next cycle calculation in parallel with the current calculation cycle. Thus propagation vectors can provide a valuable runtime tool for governing the utilization of solver resources.

Control statements are used within programs, or as operating system commands, to govern calculation of a code unit. They therefore form the basis for communication with the programming and operation utilities. The primary control statements are:

DO <cu> ;

IF <boolean> THEN DO <cu1> ELSE DO <cu2> ;

WHILE <boolean> DO <cu1> ;

FOR i=1..n,j=1..m,... DO <cu> ;

FROM <cu> ;

where <cu> is a code unit. The statement (DO) specifies only a single calculation of the code unit and therefore the calculation grid is just a single gridpoint.

The conditional statement (IF) implies that the code units (cu1 and cu2) are data-flow independent and therefore may be computed in parallel - as may be the code unit which produces the boolean condition. The grid again is but a single point for each code unit.

The WHILE statement results in a one-dimensional calculation grid of indefinite length. The code units (cu1, and the one computing the boolean) are repetitively calculated as long as the boolean is true. The code units are calculated synchronously and may be calculated in parallel as long as the results of the final calculation cycle of cu1 can be aborted.

The FOR statement establishes explicit bounds for multidimensional grids. A statement which generates the calculation depicted in Fig. 2 is

FOR i=1..4, j=1..4, k=1..4 DO

$$X(i,j,k)=F[X(i-1,j,k),X(i,j,k-1)]. \qquad (5)$$

The FROM statement is used in conjunction with the other control statements to provide endpoint (boundary) values. The values of points immediately beyond the boundaries of the calculation grid are often necessary to complete calculations within the grid. (Calculation of X(1,1,1) requires X(0,1,1) and X(1,1,0) in the above example.) These initial and boundary conditions must be synchronously transferred to the calculation of the code unit each cycle, as a function of the triggered coordinates. They can be assumed to result from the calculation of another code unit. For example,

DO <cu1> FROM <cu2> (6)

implies that cu2 furnishes the initial values for the calculation of cu1. In this case, the code units are strongly linked since the propagation of cu1 over the grid determines the propagation of cu2, and the results of cu2 may be direct functions of the grid coordinates.

MULTIPROCESSING ENVIRONMENT

Expert software for programming and operating parallel processors is envisioned as part of a distributed, cooperative, multiprocessing environment for scientific computing. A preliminary concept for such an environment is shown in Fig. 3. It consists of two major computing systems: The multiprocessor facility and the library. The multiprocessor facility contains multiple parallel processors, some dedicated to the solution of particular codes (e.g., fluids solver A, and structures solver B), and others for general-purpose computing (e.g., for use in designing other dedicated solvers, or for integrating these solvers into large system analysis). The library contains documentation, predeveloped code, and databases for recall by the user. Users might interface to the multiprocessor through personal computers and user-dedicated parallel processors containing the aforementioned expert software utilities. The functions of specific utilities are further described in the following paragraphs.

Expert Software Utilities

The programming utility is used to assist the user in creating parallel processing tasks. The user specifies the calculation task, without regard for parallelism, using control and equivalence statements to describe the calculation. The task may utilize library calls to make use of established algorithms, or to integrate any of the facility's solvers into the calculation. It is possible that other expert software could automate the selection of algorithms and solvers based on a higher-level, (differential equation) statement of the task. It is important that a structure be imposed on task specification to insure that sufficient information be provided for data-flow analysis, and for documentation of the process. In the current design effort, that structure is being based on that described in Arpasi 1985b, with the addition of dimensional data-types, higher-level calculation control statements, and other enhancements.

The partitioning utility accepts the structured definition of the task along with a description of the multiprocessor facility available for its calculation. It provides any required links to the library, does syntax and semantic testing, and offers advice on program optimization. Then, using previously established characteristics of the multiprocessor facility, it breaks the high-level statements into basic operations, computes their calculation times and uses data-flow relationships (resulting from the kind of analysis described in the previous section) to establish the task elements.

The translating utility converts the task elements into executable code for the target processors. This code, linkage information, argument transfer maps, information from the partitioning utility, the source program, and all documentation, is sent to the library for runtime reference. Also included in this task packet is a descriptive database to establish high-level communication between the operation utility and the user.

The operation utility assembles tasks into parallel processing jobs. The user communicates through a structure designed to simplify job specification. Typically, this will involve the use of control statements and operational commands such as those described in Cole 1984. The parallel processing job is assembled from tasks contained in the library, or from using a control statement. The calculation of a dedicated solver is treated as a computational element. All necessary job information is submitted to the facility manager (Fig. 3). The operation utility provides for interactive communication between the user and the multiprocessor facility manager during job execution.

The multiprocessor facility manager (MFM) controls job calculation based on the control statements. The MFM functions are shown in Fig. 4. These functions are performed each calculation cycle according to rules established during job formulation. The setup for the next calculation cycle is done in parallel with the calculation and transfer of results for the current calculation cycle. This setup includes: propagation and control (enablement and triggering) to define the calculations required in the next cycle; followed by the generation of any boundary values needed, and in parallel, any partitioning and packing

of next cycle calculations which may be required. This, in turn, is followed by the development of load modules and allocation of these modules to the processors to be used in the next cycle.

Each control processor in the MFM has a number of service (e.g., calculation) processors assigned to its domain. Allocation is made from this domain to promote parallelism. Any service processor may become a control processor if it is allocated a control statement. Thus job execution proceeds on a tree-growth basis. Results are passed down the tree as specified by the data transfer maps, until the final results reach the facility manager. They are then transferred back to the user via the operation utility.

The results, an executable session history, and other appropriate documentation are sent from the operating utility to the library for future reference.

CONCLUDING REMARKS

A concept for a library-oriented multiprocessor environment for computational fluid and structural dynamics analysis has been presented. This concept depends on the development of expert software utilities that will automate the programming and operating of parallel processors, eliminating the need for the user to "think parallel." The feasibility of such utilities has previously been demonstrated for parallel processing of ordinary differential equations. Current efforts are aimed at extending those concepts and designs to the more general case of multidimensional, partial differential equations. Of course, the concepts presented are merely an outline of a general approach to developing the proposed multiprocessing environment. As specific software utilities are designed and tested, a clearer picture of the overall environment should emerge leading to the development and demonstration of a prototype system. It is recognized that the overhead time required by the expert software utilities, particularly at runtime, may offset any reduction in time gained by the parallel processing. Careful attention must be given to both software and hardware design to avoid this.

REFERENCES

1. Arpasi, D.J. and Milner, Edward J.: Partitioning and Packing Mathematical Simulation Models for Calculation on Parallel Computers. NASA TM-87170, 1986.

2. Arpasi, D.J.: RTMPL - A Structured Programming and Documentation Utility for Real-Time Multiprocessor Simulations. NASA TM-83608, 1985.

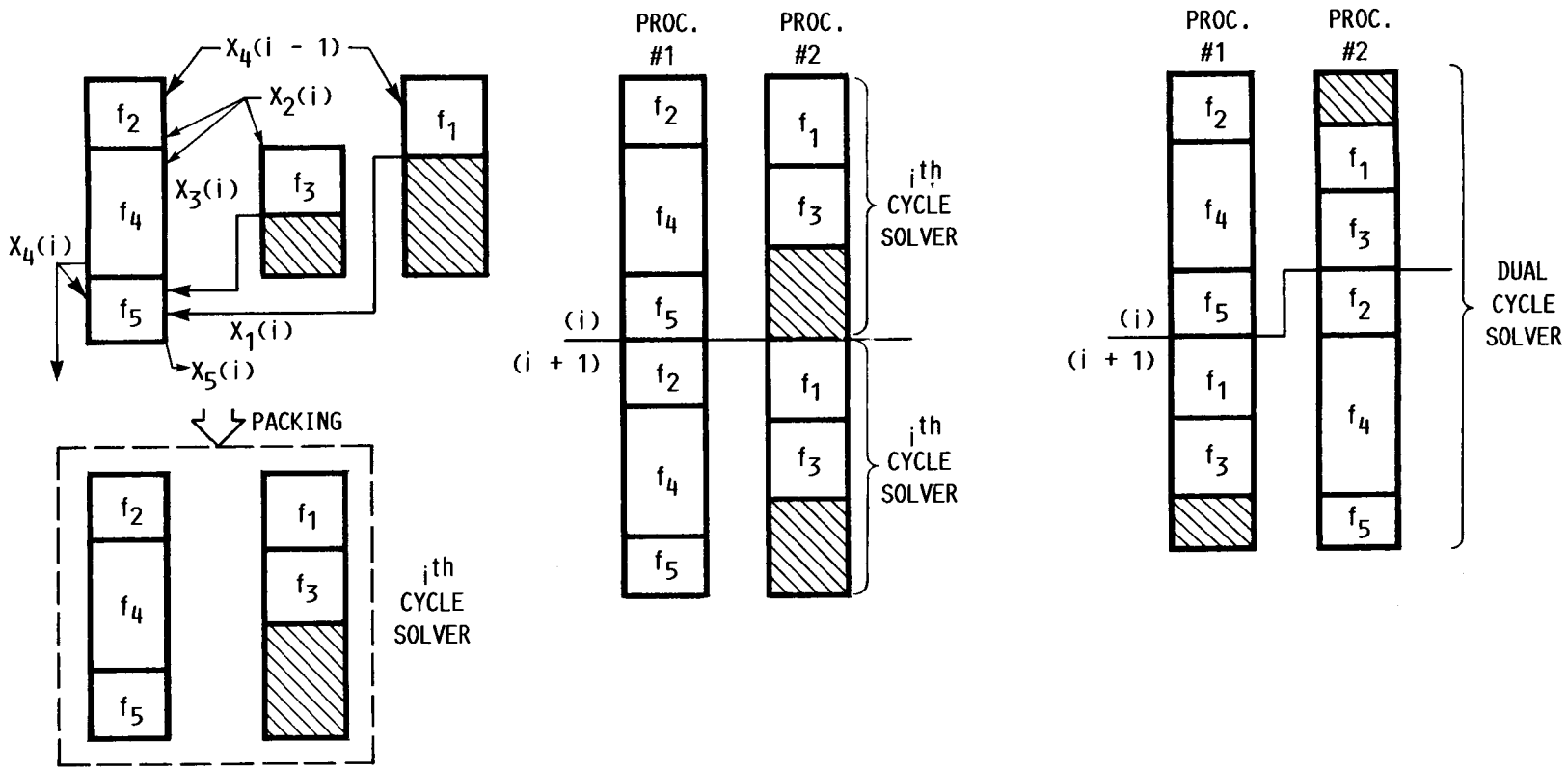
3. Arpasi, Dale.: Real-Time Multiprocessor Programming Language, (RTMPL) - Users Manual. NASA TP-2422, 1985.

4. Cole, G.L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator. NASA TM-83605, 1985.

5. Cole, G.L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator - Users Manual. NASA TP-2426, 1984.

6. Makoui, A. and Karplus, W.J.: Data Flow Methods for Dynamic System Simulation: A CSSL-IV Microcomputer Network Interface. Proceedings of the 1983 Summer Computer Simulation Conference, Vol. 1, pp. 376-382.

7. Feyock, Stefan and Collins, Robert W.: A High-Order Language for a System of Closely Coupled Processing Elements. Final Report NASA Grant NAG-3-232, 1986.



(A) DATA-FLOW AND PACKING OF i-CYCLE CALCULATIONS.

(B) DUAL CYCLE CALCULATION USING REPETITION OF A SINGLE CYCLE SOLVER.

(C) DUAL CYCLE CALCULATION USING A SOLVER FORMED FROM PACKING ALL ELEMENTS IN BOTH CYCLES.

FIGURE 1. - BASIC DATA-FLOW CONSIDERATIONS.

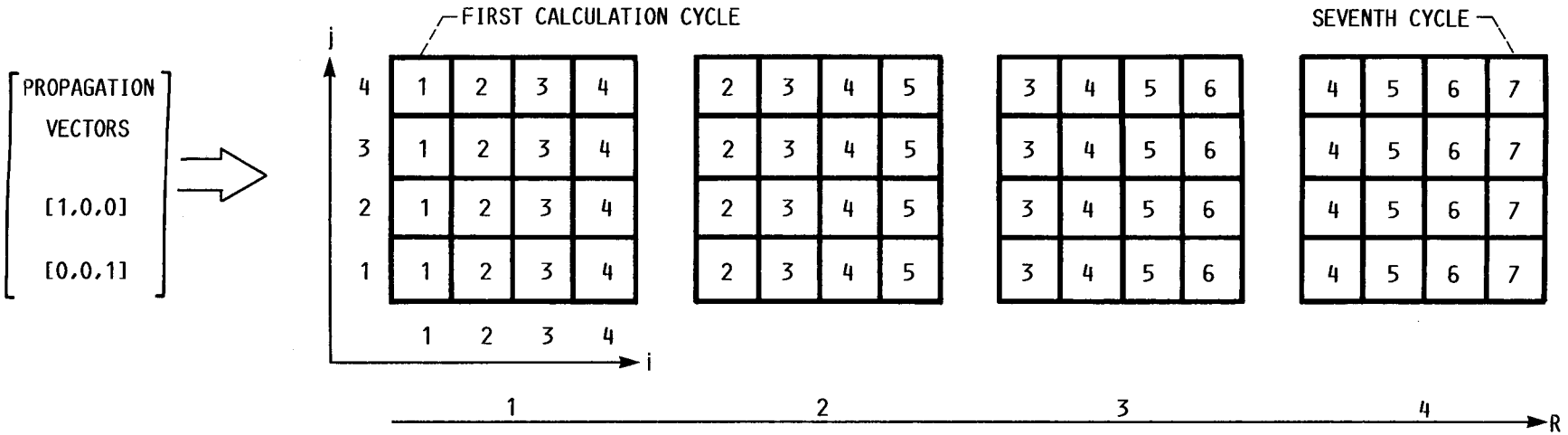


FIGURE 2. - CALCULATION CYCLE PROPAGATION.

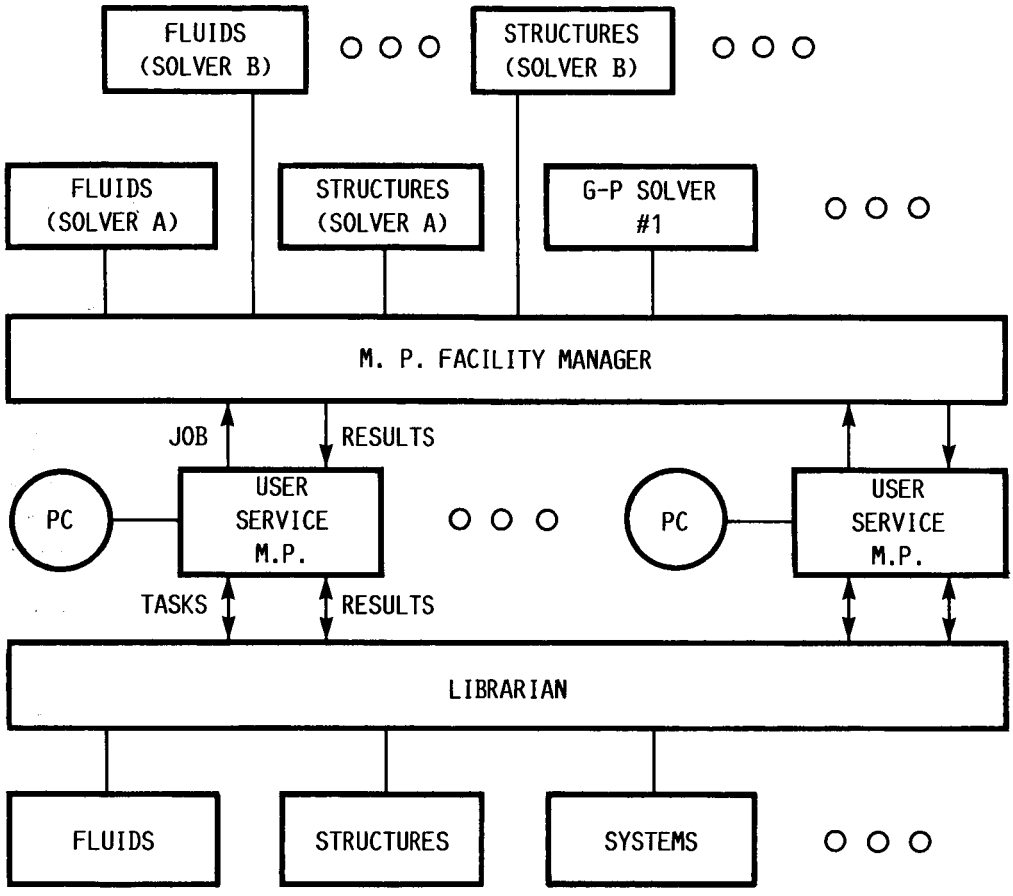


FIGURE 3. - CONCEPTUAL MULTIPROCESSING ENVIRONMENT.

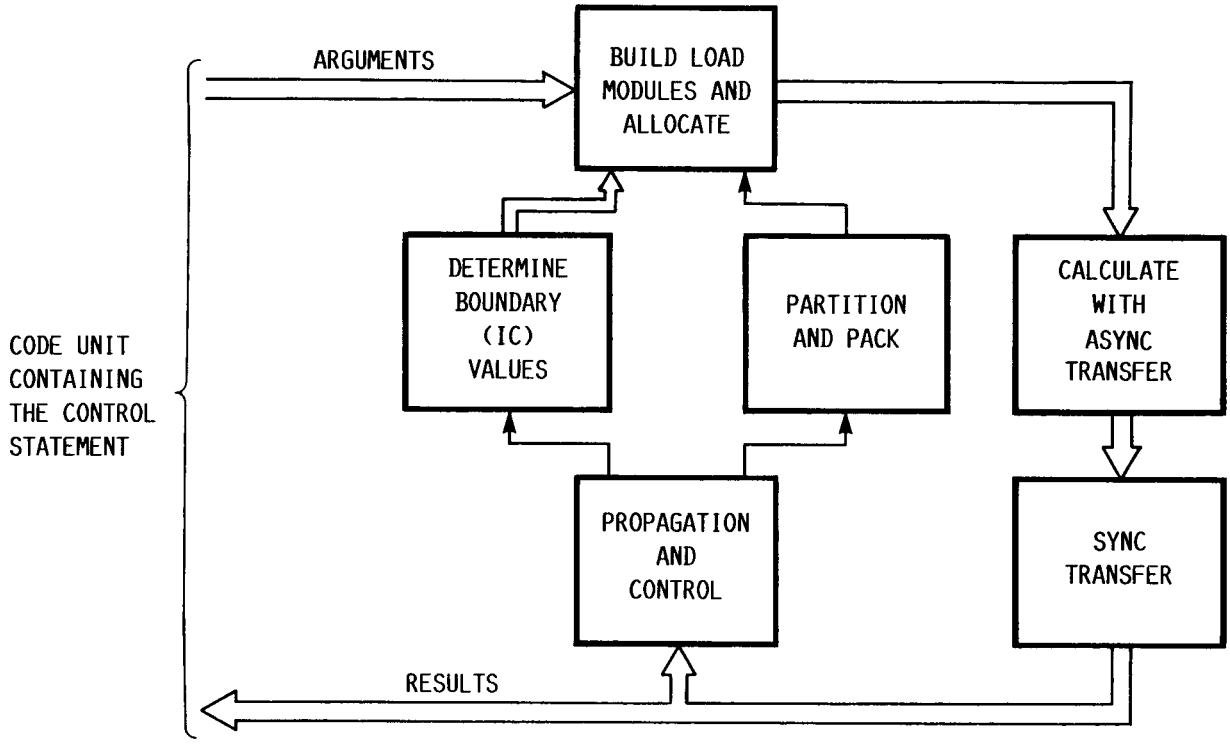


FIGURE 4. - CALCULATION CONTROL FUNCTIONS.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-89837		

4. Title and Subtitle	5. Report Date
Automating the Parallel Processing of Fluid and Structural Dynamics Calculations	6. Performing Organization Code: 505-62-21

7. Author(s)	8. Performing Organization Report No.
Dale J. Arpasi and Gary L. Cole	E-3494
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	13. Type of Report and Period Covered: Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Prepared for the 1987 Summer Computer Simulation Conference sponsored by the Society for Computer Simulation, Montreal, Canada, July 27-30, 1987.

16. Abstract

The NASA Lewis Research Center is actively involved in the development of expert system technology to assist users in applying parallel processing to computational fluid and structural dynamic analysis. The goal of this effort is to eliminate the necessity for the physical scientist to become a computer scientist in order to effectively use the computer as a research tool. Programming and operating software utilities have previously been developed to solve systems of ordinary nonlinear differential equations on parallel scalar processors. Current efforts are aimed at extending these capabilities to systems of partial differential equations, that describe the complex behavior of fluids and structures within aerospace propulsion systems. This paper presents some important considerations in the redesign, in particular, the need for algorithms and software utilities that can automatically identify data flow patterns in the application program and partition and allocate calculations to the parallel processors. A library-oriented multiprocessing concept for integrating the hardware and software functions is described.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel processing Simulation (digital)	Unclassified - unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	9	A02

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