MIT/LCS/TR-216

ANALYSIS OF THE SIMPLE CODE FOR

DATAFLOW COMPUTATION

John M. Myers

This blank page was inserted to preserve pagination.

Analysis of the SIMPLE code for dataflow computation

by

John M. Myers, Consultant

May, 1979

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

Cambridge   Massachusetts 02139

This empty page was substituted for a blank page in the original document.

Analysis of the SIMPLE code for dataflow computation

John M. Myers

ABSTRACT

We analyze a problem in hydrodynamics from the standpoint of computation on a dataflow computer that is not yet fully specified, with the objectives of helping to further specify the computer and helping to develop VAL as its source language. Lawrence Livermore Laboratory supplied the algorithm for hydrodynamics, including heat flow, as a 1749-line FORTRAN code called SIMPLE.

The algorithm viewed as 'abstract' (i.e. independent of physical arrangements in space and time for its realization) is shown to imply spatial and temporal structure that must appear in any and all implementations. Both for hardware design and program compilation it is useful to map this structure to grosser levels of description, with the grosser levels reflecting modularity of computational resources conjoined with modularity of the algorithm. Following Holt (1979) we use role diagrams to display spatio-temporal structure at different descriptive levels, so as to guide translation into VAL as well as the analysis of the time to compute.

Inter-resource communication essential to the problem is displayed, and various issues of machine design are defined. Using VAL with one set of extensions, we express the algorithm so that in principle it can be compiled for execution by a dataflow computer. Input-output functions beyond those implied by the SIMPLE code are discussed. A second set of extensions to VAL is advocated to express the conjunction of problem and resource modularity, so as to guide compilation. The dependence of time to compute on the number of processing units is shown for various aspects of the problem.

Analysis of the SIMPLE code for dataflow computation

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

This empty page was substituted for a blank page in the original document.

Analysis of the SIMPLE Code for Dataflow Computation

1. Introduction: Hydrodynamics Meets a Dataflow Computer

The equations of physics are prescriptions for calculating; from some presumed starting conditions, they generate a "future". The calculation of this "future" involves many events, each of which "consumes" items -- values of variables -- and "produces" other items. Because an item cannot be consumed before it is produced, these events are subject to <u>constraints of sequencing</u>. These constraints impose a pattern on the calculation.

Although the equations of physics constrain the calculation, they do not fully determine it. The pattern is partly determined also by the method of solution employed and by the structure of the computer. Thus the same (partial differential) equations can result in different patterns of calculation, according to the method of solution and the arrangement of computational resources. For this reason the pattern of computation for a given type of problem, say hydrodynamics, evolves as methods and computational resources evolve. Pattern, method, and resources <u>are coupled in their evolution,</u> with each selected in part to support and to draw on the others.

Over most of history the computer (human or machine) had only a sequential processing capacity, so that computation was necessarily performed one step after another. Thus methods which emphasize concurrency were not called for, and as a result are today relatively unexplored and undeveloped. Not only computers, but also numerical methods have evolved in a context that is weighted toward the sequential, and away from the concurrent.

Via such means as dataflow architecture (see Dennis, 1978), an increase in speed can be brought about by an organization of computational resources that allows concurrency of many events. This report is concerned with fitting -- or refitting -- a pattern that evolved in a sequential context

onto a dataflow computer. The report is based on a case study of an example
program written in FORTRAN for a sequential machine for the solution of a
problem of hydrodynamics, including heat flow. This program was prepared by
Lawrence Livermore Laboratory, and is named SIMPLE. The initially presented
questions were:

1.1) What is involved in translating the SIMPLE program from FORTRAN
        (suitable for a sequential computer) into a dataflow language
        (the VAL language in particular); and

1.2) Compared to a sequential computer, what speed advantage can be
        expected from a dataflow computer in the execution of the SIMPLE
        program?

        To realize the potential advantage of a dataflow computer, its
program must be free of unnecessary sequencing constraints. Sequencing
constraints come from many sources, and their necessity depends on ones
point of view. Primarily we report on the narrow view that sees sequencing
constraints as imposed by the data dependencies of the FORTRAN program.
In this view the "translation" per item 1.1 entails the removal of sequencing
only as far as possible without disrupting the data dependencies expressed in
the FORTRAN program. Such a translated program would be expected to produce
numerical results identical to the FORTRAN program, apart from round-off errors.
        But the narrow view fails to:

a) realize the potential for advances in speed, and

b) open the physics itself to new perspectives made possible by the
   power to express concurrency.

Although their resolution is outside the scope of this report, we shall define some broader issues of solution methods, machine design, and physics.

With respect to item a), the translated program will still contain unnecessary sequencing constraints, imposed by a method of solution of the equations of physics. For example, the back-substitution method (Crowley, Hendrickson and Rudy, 1978) for solving the implicit formulation of heat flow does not realize the potential of dataflow architecture, and it appears that a method could be developed that (for a dataflow computer, but not for a sequential computer) would be substantially faster. Thus in presenting our results, we shall distinguish sequencing constraints that come from the happenstance of the numerical method embodied in SIMPLE from constraints that come from less malleable sources.

Once the method of solution is considered as variable and not fixed, issues of machine design surface. If methods and machine are to be developed in concert, it might be best to tailor the machine to a certain class of methods, to the detriment of its performance with methods outside that class. If the dataflow computer is seen as a network of interconnected processors, then this issue arises with respect to the communications facility that provides processor-to-processor communication. The problems under study stem from spacially distributed fields that interact in a purely local manner. From this locality one can show that the equations can be solved on a dataflow machine using a communications network which directly links only nearest neighbors, so that a "global" communications facility is not required. Local networks are cheaper and faster than global networks; however the methods that they support have drawbacks with respect to speed, so that the question of local vs. global remains open. One way of posing the issue is through the following question:

1.3) What number N' of globally connected (i.e. fully connected)
      processors have the same cost as N locally connected processors,
      under the condition that the total memory of the two configurations
      be the same?

The idea is that the speed loss from the restriction to local connectivity
might be regained through the use of a larger network of processors. In other
words for a given investment there is a trade-off between fewer fully connected
processors and more locally connected processors. If these two contrasting
configurations are to be evaluated in their performance on a given problem,
then total system memory should be the same for each configuration.

     With respect to item b) it may be of theoretical interest to
introduce a class of dataflow computers to model what is meant by the equations
of physics.

2. The Hydrodynamic Fields

Given finite propagation velocities, the fields defined by the equations of physics can be pictured, as they were by Huygens, as networks of communicating entities, all operating concurrently. A partial differential equation represents a limit as the network becomes progressively more fine-grained. Computation is possible, however, only if the limit is not taken, or if it is "undone".

Via one or another numerical method the partial differential equations are transformed to difference equations defined on a spatial mesh of N zones, with each zone have corners at nodes, as shown in Fig. 1. In terms of the parameters defined in SIMPLE, one finds

$$N = (LMX-LMN)*(KMX-KMN) .$$
(Eq. 2.1)

SIMPLE employs a Lagrangian formulation, in which the mesh is deformable; each node is thought of as a "tagged atom", carried along in a fluid whose motion is described by the difference equations. By extending the discussion of Morse and Feshbach (1953, vol 1, p.847-8) to equations of hydrodynamics, one sees Huygen's principle works on a sufficiently small region of the mesh. For a given node, one can choose an enclosing curve through the zones that bound it, and with the result that, by interpolation, the acceleration of the node depends only on the properties of the zones that bound it. A similar argument could lead to the conclusion that the current properties of a zone depend only on past poperties of the nodes at its corners, but SIMPLE is based on a variation of this argument. Properties such as pressure and density are defined only for zones and not for nodes, and the current properties of a zone are shown to depend on their past values together with the current deformation of the zone, along with the current rate of deformation of the zone.
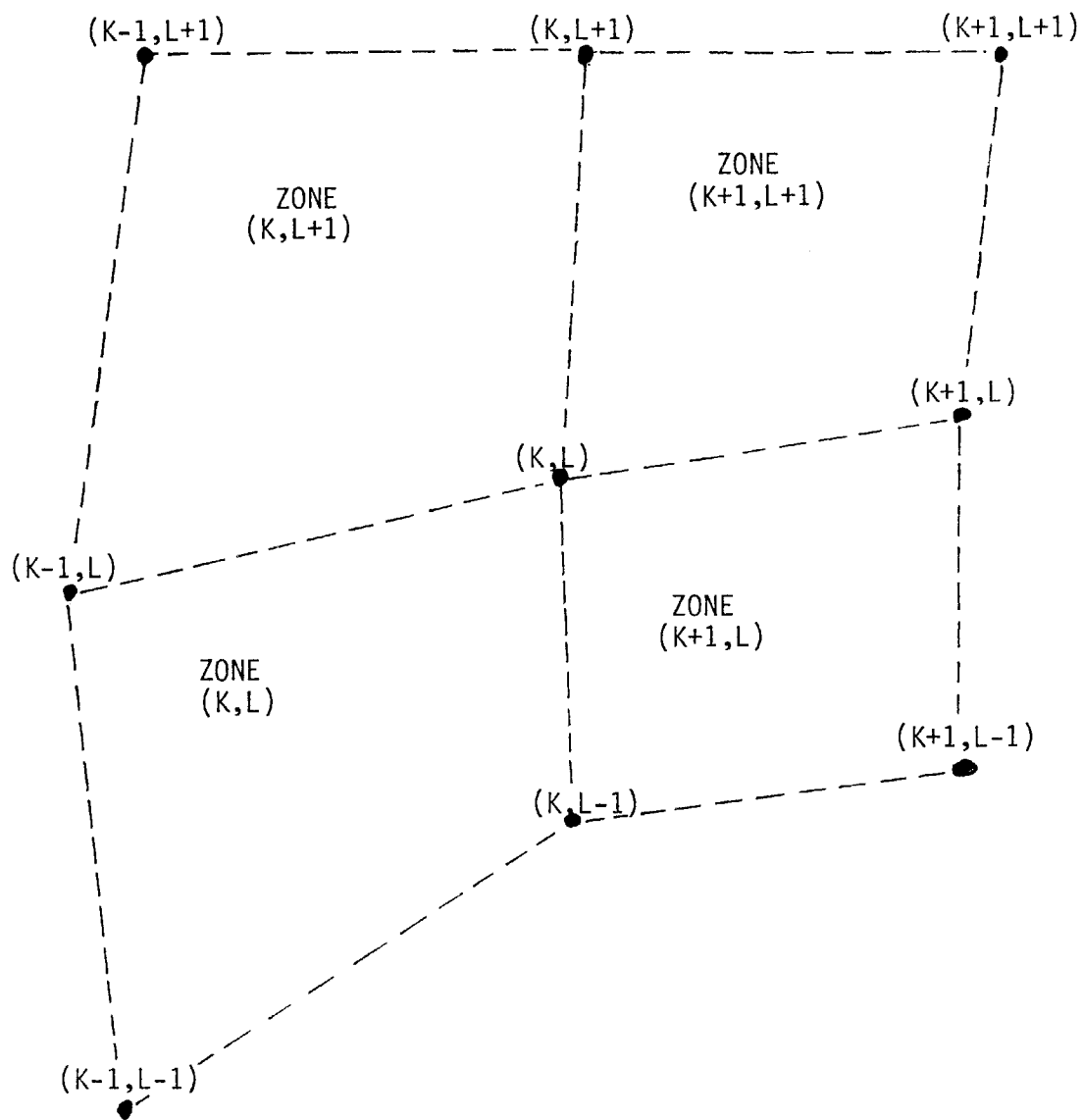
Figure 1: Nodes (shown as heavy dots) and zones (enclosed by dotted lines).

The main fields are defined by Crowley, Hendrickson, and Rudy (1978) as follows:

Field	Zonal name in FORTRAN	Definition
ε	E	energy per unit mass
p	P	pressure
q	Q	artificial viscosity
ρ	RHO	density
θ	TEMP	temperature
τ		specific volume
κ		thermal conductivity .

In addition the positions and velocities of the nodes form a field as a function of node indices k and l:

Field	Nodal name in FORTRAN	Definition
$\vec{x}$	R,Z	position as function of k,l
$\vec{u}$	U,W	velocity as function of k,l.

The field equations are

$$\frac{d\varepsilon}{dt} = - (p+q)\frac{d\tau}{dt} + \frac{1}{\rho} \nabla \cdot \kappa \nabla \theta \qquad \text{(Eq. 2.2)}$$

$$\theta = \theta(\rho, \varepsilon) \qquad \text{(Eq. 2.3)}$$

$$\kappa = \kappa(\theta) \qquad \text{(Eq. 2.4)}$$

$$q = q(\rho, \Delta\vec{u}, \varepsilon) \qquad \text{(Eq. 2.5)}$$

$$\frac{d\vec{x}}{dt} = \vec{u} \qquad \text{(Eq. 2.6)}$$

$$\rho \, \frac{d\vec{u}}{dt} = - \, \boldsymbol{\nabla}(p+q) \qquad\qquad\qquad\qquad\qquad \text{(Eq. 2.7)}$$

3. Communications and the Speed and Configuration of a Dataflow Computer

3.1. General issues

Because the least familiar aspect of a dataflow computer is its communications facility, we give a preliminary statement of issues of speed and machine design posed by the burdens that the SIMPLE problem will place on such a facility.

A computational algorithm, such as the FORTRAN program of SIMPLE, defines a flow of data values into and out of arithmetic operations. By analyzing this flow, one can produce a dataflow graph that displays not only the concurrency that is allowable within the confines of the algorithm, but also an abstract pattern of communication. For the SIMPLE problem, most of the dataflow graph can be modularized onto regions corresponding to the mesh of Fig. 1: one region for each zone, and one for each node.

To perform the computation, resources are required: physical actors must be provided to carry and transform the values that are specified by the dataflow graph. The correspondence between physical actor and role as value carrier is in part subjective, and inescapably so. There is no sure rule for the "right way" to establish the correspondence, although there are criteria by which to exclude many "wrong ways": wrong ways lead to failure (e.g. of performance or of budget). In the light of currently well developed technology, we may start by assigning a physical processor to each nodal and zonal region of the dataflow graph. If each such processor comes with attached memory, then a dataflow computer can consist of a set of processors together with a communications facility that links them.

Affordable communications facilities never offer the full measure of speed, bandwidth, freedom from blocking, and other properties that it would be "nice" to have. Compromise is necessary. The determination of an

economic configuration is outside the scope of this work, but to help prepare the ground, we consider the message patterns that are generated by the SIMPLE program. All of the communications facilities under consideration could handle all of these patterns, but different facilities will exhibit different speeds for different patterns. Thus it is helpful to find out what patterns really matter.

The burden placed by a dataflow graph on the communications facility depends on:
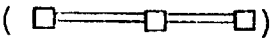
.1 the connectivity of the dataflow graph -- how "scrambled" are the needed connections;

.2 the number and accuracy of the field variables to be transmitted.

A given dataflow computer can compute a dataflow graph corresponding to a square mesh of D zones without having to time-share its hardware (as would a sequential computer). Thus D measures the largest mesh that a given dataflow computer can handle in some "fully concurrent" manner. If D is to be increased, then additional hardware must be incorporated into the dataflow computer. In many cases of interest one expects to find $N \gg D$, so that each processor will have to be time-shared among N/D regions. The burden on the communications facility will thus also be influenced by:

.3 the way in which resources are time-shared over different regions of the dataflow graph.

Item .2 affects only the size of the messages to be transmitted and will not be further considered here. Items .1 and .3 affect the "from-where-to-where" aspect of the communications burden, and we now discuss them further.

3.2. Connectivity in the face of resource sharing

By means of a role diagram, further explained in Appendix A,
Figure 2 illustrates the connectivity exhibited by the main cycle of a problem
like SIMPLE, but reduced to one space dimension and stripped of heat flow.
Figure 2 can be read as a marked graph over which tokens are moved to simulate
the occurrence of calculational activities; the top row of circles are viewed
as initially marked with tokens. A horizontally connected row of boxes
($\square\!\!=\!\!=\!\!=\!\square\!\!=\!\!=\!\square$) is a calculational activity. The inputs to an activity
arrive from above; the outputs depart below -- in other words the "flow of
time" is downward. Boxes connected by double bars ($\square\!\!=\!\!=\!\!=\!\square$) produce identical
copies of the same output value, and thus portray fanout. The figure is thought
of as wrapped around a cyclinder, with each bottom circle "wrapped up" to
coincide with the circle directly above it, so that a cycle is defined.

The diagram is to be interpreted not just as an abstract flow of
values, but as a <u>flow of values carried by physical actors</u>. Each vertical
line in Fig. 2 requires a physical resource, like a processor or a buffer,
that carries a value from one calculational activity to another. Each hor-
izonatl row likewise specifies a physical requirement -- e.g. for the
processing resources needed if the indicated values are to meet and be trans-
formed. The diagram of Fig. 2 looks similar to a dataflow graph because it
assumes no constraints due to any scarcity of resources: it assumes that
processors and communications links are provided in abundance, at least at
the level of detail portrayed. Resource constraints would change the picture;
for example, Fig. 3 shows the same values as they would flow under additional
constraints imposed by a scarcity of processors such that each processor
must handle two adjoining activities.

Figure 2: Connectivity of simplified hydrodynamics in one space dimension with one processor assigned to each nodal and zonal calculation.
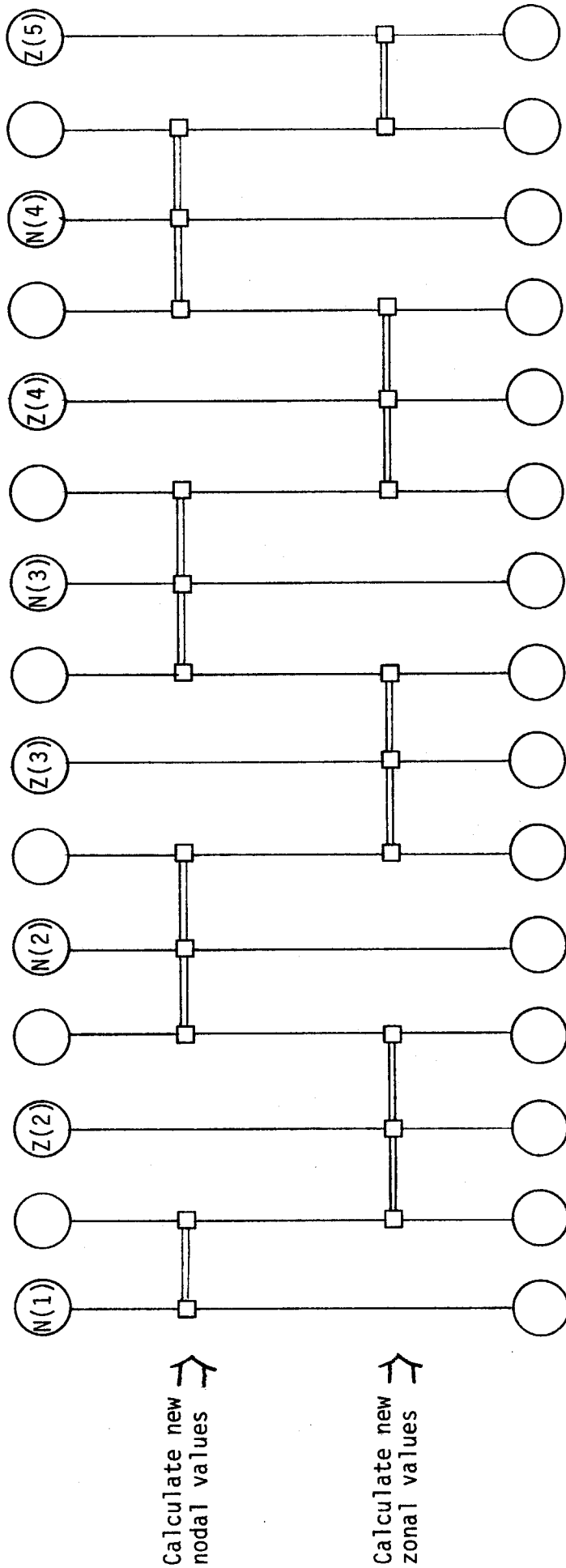
Note: N(K) = set of values for node K: {P, Q, RHO, E};

Z(K) = set of values for zone K: {X, V}.
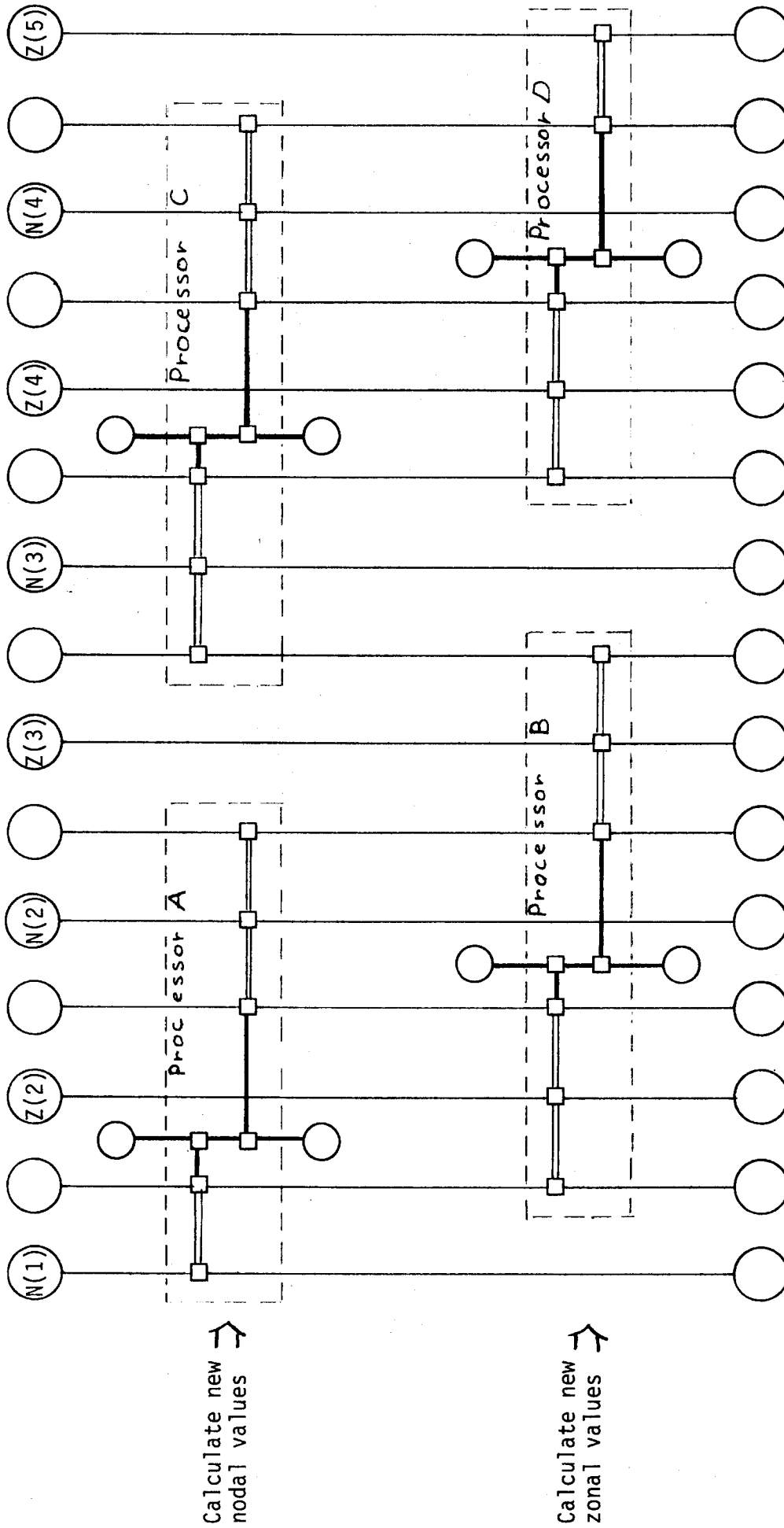
Figure 3: Constraints on concurrency (heavy lines) imposed by sharing of processors.
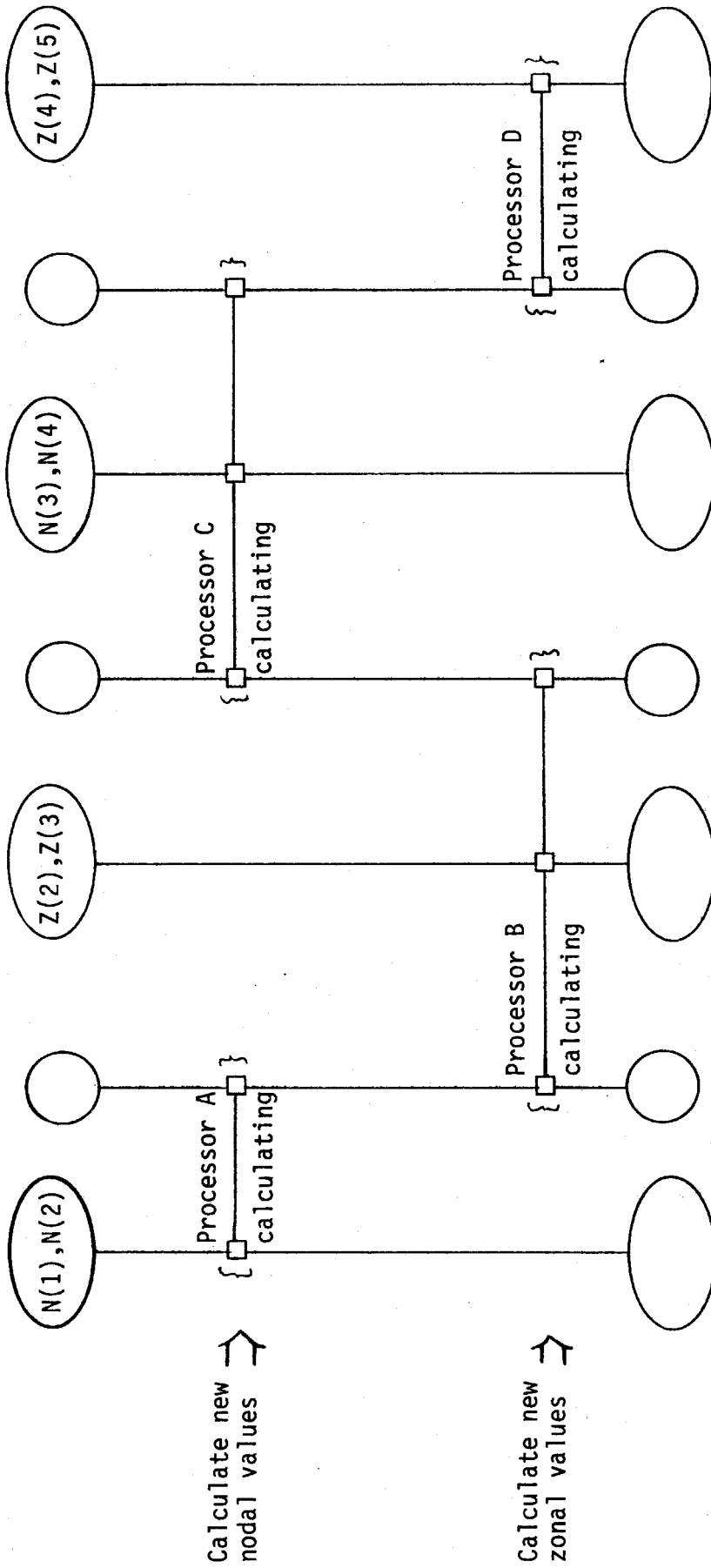
Figure 4: Grosser view of Fig. 3 highlighting connectivity between processors; (compare with Fig. 2).
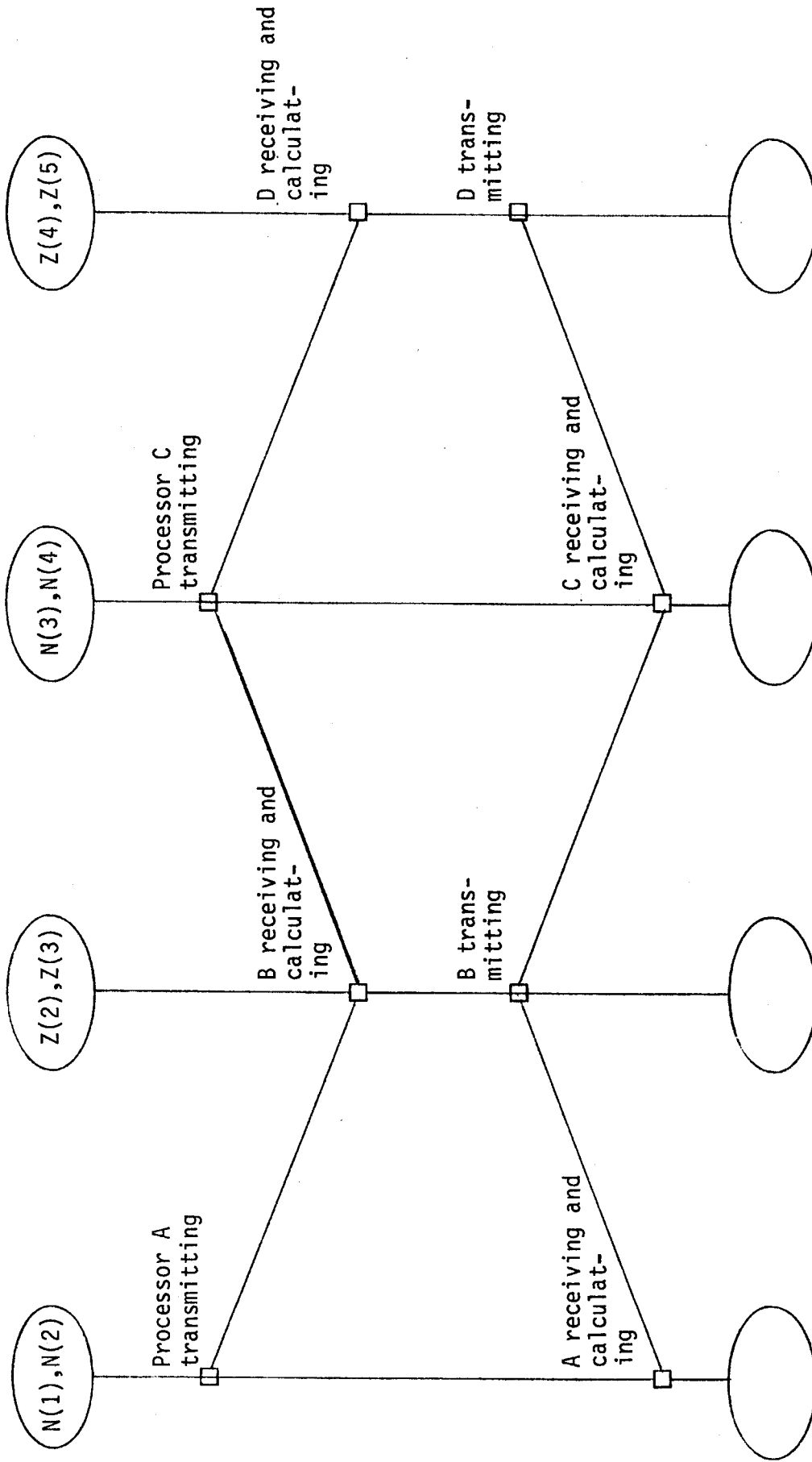
Figure 5: Alternate view of Fig. 4 using the notation of buffered communication.

(See Appendix A, Sec. A.19 for more on the notation.)

The suggested assignment of one processor to one nodal or zonal region of the dataflow graph was in some degree arbitrary. Given a small mesh and many processors, concurrency might be enhanced by assiging more than one processor to each such region. For a mesh large compared to the number of available processors, each processor would have to be assigned a larger piece of the dataflow graph. A question then arises: under this circumstance does simplicity in the connectivity of the dataflow graph imply that simplicity can be maintained in the connectivity of the processors? The answer depends on how a single processor is assigned to cover more than one region. Figure 3 illustrates the principle that such assignment can be made so that the connectivity between processors is no more complex than is the connectivity between nodal and global regions. Figure 4 highlights this connectivity among shared processors; the same connectivity can be maintained when processors are shared over larger regions of the dataflow graph. By use of the abbreviated notation described in Sec. A.19 of Appendix A, Fig. 5 shows the same connectivity as Fig. 4, but with the communications buffers (the unlabeled roles) suppressed. A slanting bar implies: a) that the lower of the activities consumes something produced by the upper activity; and b) that the two activities are linked by an intermediating resource (such as a buffer) that is not explicitly shown.

What can we learn from this example that is more generally applicable? Sharing of processors reduces the size of the communications facility required of a dataflow computer, at the cost of speed. For this example and this manner of assigning processors, the communication pattern, although becoming smaller, preserves its connectivity; be it one or many regions of dataflow graph per processor, each processor communicates only with itself and with its nearest neighbors. In the SIMPLE problem one finds somewhat more complex

more connectivity in the dataflow graph. Two points are to be noted in the assignment of processors to pieces of dataflow graph of SIMPLE.

.3. A mesh of N zones can be parcelled out to D processors in such a way that the connectivity among processors preserves any "localness" present in the connectivity among nodal and zonal regions of the dataflow graph.

.4. Other schemes of assigning processors that place additional demands on their connectivity may offer advantages in speed.

Because of item .3 we can learn what connectivity is <u>necessary</u> to D processors of a dataflow computer that is to solve a mesh of N zones, merely by studying the connectivity of the dataflow graph. Because of item .4 we must bear in mind that there will be additional questions of trade-offs between speed, cost, and the connectivity of the communications facility.

3.3. Fitting the Computation to the Minds of the Analysts: Input and Output

Programs and parameters flow into a pattern of computation, and significant features of the computation flow out. In some cases this interaction can be partitioned into a sequence of phases: input, computation, output. However, as the size of the computation increases there is progressively more need to operate interactively, so that the selectivity of what flows out can be increased along with the amount of computation.

Output from a dataflow machine is apt to involve transforming an array, or some feature (such as a contour) extracted from it, into a sequence of characters to be transmitted -- either to a person or to a storage device. Such operations are bandwidth limited and threaten to demand excessive time or buffering or both. As the scale of computation is increased, it becomes necessary to increase the selectivity of feature extraction in near proportion.

One reason that extracting features is challenging is that what is significant sometimes becomes apparent only as the computation unfolds, so that the definer of significance must interact with the computation. Further, significance varies according to the viewer. Because of this "vaporous" quality, one approach is to report out "all the data" from a computation, so that it forms a database that can later be manipulated according to taste. As the scale of computation increases, this approach becomes progressively more demanding, and may become unrealizable.

An alternative approach would be to provide a facility by which multiple viewers of the computation could each construct filters and other "feature extractors" in real time as the computation proceeds. No doubt some users would still build "databases", but they would have the opportunity (and perhaps the necessity) of building more selectively than has been the

common practice.

This approach generates requrirements to be met by dataflow hardware and software. The image is of a controllable "funnel" or "tree" that sucks up arrays of field variables as the computation proceeds, discards what is irrelevant, and issues a stream of characters that conveys the features specified by one or another analyst. The "specification of relevant features" could by supplied prior to execution, or could be supplied interactively by the analyst as the computation unfolds.

Such a scheme demands software interfaces that can accept analyst-supplied specifications of the features to be selected. Presumably the structure should accomodate multiple analysts. The hardware requirements are an extension of those already generated by the needs to sum over an array and to convert an array into an output stream for transmission over a single communications line. For example, program-controlled merging of array elements into a stream can provide efficient sorting. Just as they are needed to sum and to report out all the elements of an array, tree structures will be needed to report out selected elements of an array (such as the elements of a contour). However, one expects an advantage from more flexible control of tree connectivity and of tree, nodal and zonal processing than would be needed just to solve the field equations.

4. Modeling the Time to Compute

The prediction of execution time of SIMPLE on a dataflow computer
that is not yet fully specified is a complex task which, in this report,
can be started but not completed. For this reason we separate a general
discussion of what needs to be undertaken from a sketch of initial results.

4.1. Choosing an appropriate form of model

The question of time to compute is a question of what happens when
an abstract pattern -- the algorithm of SIMPLE -- meets a configuration of
physical resources -- communications lines, switches, buffers, processors, etc.
that compose a dataflow computer. The modeling of computation time entails
the modeling of the joining of the abstract event of the algorithm with the
physical event of the configuration. This calls for a modeling form that
straddles abstract (i.e. input-output) relations and physical circumstances.
For example, we are forced to observe that anything that is (even a value)
must be some place, such as on a communications line, in a buffer, etc.
We must learn to see something like a dataflow graph as having, in addition
to its implications for abstract values, implications concerning the resources
required to support the logical operations on values. As a foundation for
this shift in view, we turn to Holt's (1979) concept of the role played by an
actor who carries a value. The value is in the domain of mathematics and
algorithms; the actor (human or mechanical) is in the domain of space and time.

It would be advantageous to have a gross model with only a few
parameters, both to estimate the time for a dataflow computer to solve the
SIMPLE problem, and to help in configuring an implementation of a dataflow
computer. However, a believable gross model of such a complex situation
can be derived only by condensing a model that encompasses sufficient

complexity to account, for example, for the effects of pipe-lining and of communications bottlenecks. It thus appears that the modeling form should lend itself to different levels of detail.

The modeling method must encompass the concurrency exhibited by dataflow architecture. This requirement rules out models based on the concept of a system state, and directs toward models based on Petri nets.

The modeling scheme must provide for the modeling of different methods of numerical solution. For example, the implicit formulation of heat flow results in a difference equation, the solution of which is equivalent to the inversion of a certain near-diagonal matrix. The method of inversion used in SIMPLE is that of back-substitution. However, it appears possible to develop an alternative method that would impose far fewer unnecessary sequencing constraints, and would hence better realize the potential advantage of dataflow architecture.

The SIMPLE program uses a global determination of a time step that varies from one cycle to another, but is invariant over the mesh. It appears that in the computation of hydrodynamic shock, there would be a substantial advantage in providing for the local determination of time steps that would vary not only from cycle to cycle, but also from location to location over the mesh. Such methods are used in the calculation of gravitational fields and in relativistic fluid dynamics, as is discussed by Misner, Thorne and Wheeler (1970, Chap. 42). Although this extension of method is outside the scope of our present work, we require that the modeling method encompass time steps as local values derived on an even footing with other field quantities.

These requirements suggest modeling based on the concept of a Petri net. Because of its capacity to join abstract and physical operations, we choose the modeling scheme of Holt (1979) to express the essential logical and physical dependencies. For a discussion of the concepts, the reader is

referred to the cited report of Holt. As a "quick and dirty" view of "how to do it", Appendix A describes the modeling conventions.

4.2. The need for speed

Faster computers are desired to allow a finer grained mesh. Consider a given physical domain and a given duration of hydrodynamic interaction. As the mesh is made finer the number of zones, N, increases, and moreover the physical time step achievable in a cycle of computation decreases as $1/\sqrt{N}$. Therefore the time to compute increases as $N^{3/2}$. This dependence applies to a dataflow computer with $D \ll N$, just as it does to a sequential computer.

To decrease the linear dimension of the zones by a factor of 10, N must increase by a factor of 100, and to maintain a fixed time to compute, given the necessary decrease in physical time step, the speed of the computer must be raised by a factor of 1000.

One should not that the constant of proportionality that relates the allowable physical time step to $1/\sqrt{N}$ depends on the numerical method used, and that the freedom to choose an advantageous method depends on the connectivity of provided by the communications facility of the dataflow computer. Richer (e.g. more than nearest-neighbor) connectivity supports larger time steps, but then richer connectivity slows the computer and requires an investment that could otherwise buy more processors; thus there is a trade off.

4.3. The computational cycle

The SIMPLE computation consists of initialization followed by repeated execution of a main cycle. A cycle consists of computing the velocity and position of each node, and then computing the properties (such as pressure and density) of each zone. The cycle involves times in two senses: a physical time step (e.g DTNPH in SIMPLE); and a time to compute the cycle. Because the initialization is done once and the cycle is repeated many times, the (total) time of computation is nearly independent of the time to initialize the computation, and is essentially the time to compute a cycle multiplied by the number of cycles.

The computational cycle can be partitioned either in terms of the physics or in terms of the concurrency and connectivity that it presents. These two partitionings result in somewhat different pictures. The following is a compromise between the two. We view the cycle as composed of the following phases of activity:

.1. establish boundary values (by means of "ghost" nodes and zones);

.2. calculate velocity and position of interior nodes;

.3. calculate zone variables for interior zones (e.g. pressure, specific energy, artificial viscosity, density) except for temperature;

.4. calculate temperature and recalculate energy to include the effect of heat flow;

.5. calculate the time step for the next cycle;

.6. calculate totals: work done on boundary, energy lost, etc.

.7 extract needed output and bring in parameters to control
subsequent output, as discussed in Sec. 4.5.

Figure 6 schematically displays the types of connectivity, and
hence concurrency, in the flow of data prescribed by SIMPLE over a network
of processors, with one processor assigned to each node and each zone of
the dataflow graph. Additional processors are assumed to handle the "tree"
connectivity of phases 5, 6 and 7. As noted in Sec. 3, if fewer processors
are available, they can still be connected with the same connectivity, by
assigning each processor a set of contiguous zones, contiguous nodes, or
portion of the "tree". If more processors are available, then more than one
can be assigned to a given nodal or zonal region of the dataflow graph, with
the result that a higher degree of parallelism will be achieved. Some
possible assignments of this type are illustrated in Appendix B.

Phase 1: Establish boundary values via ghost nodes and zones (typical row or column).

Phase 2: Calculate velocity and position of interior nodes (typical row or column).

Phase 3: Calculate zonal values except temperature (typical row or column).

Phase 4: Calculate temperature and correct energy:

    calculate CBB and DBB (typical row or column);

Z-sweep (typical column, all columns in parallel);

R-sweep (typical row, all rows in parallel);

(continued on next page)

(continued from preceding page)

Phase 5: Calculate next time step and distribute ("tree" connectivity covers all zones):

　calculate locally,
　then take minimum;

　distribute.

Phase 6: Calculate total internal energy and energy exchange across boundary ("tree" connectivity covers all zones; see note a.)

Phase 7: Input/output:

　test values (e.g. against thresholds) and extract features (see Note b.)

　receive changes in parameters (e.g. thresholds) that control feature extraction. (See Note b and Secs. 3.3 & 4.5.)
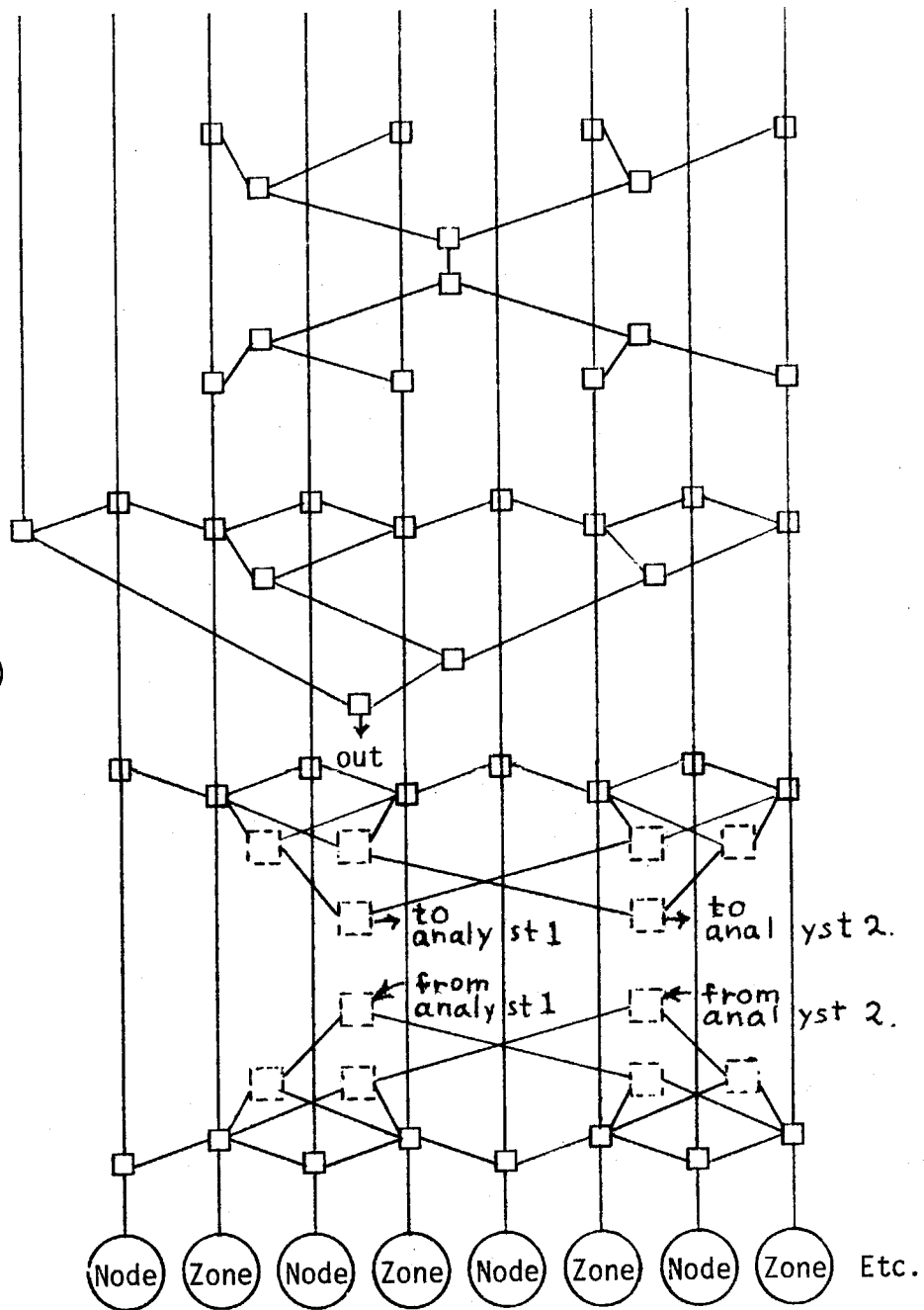
Note a: Phase 6 consists of a local calculation, like phase 3, followed by a summing operation. In SIMPLE this phase is distributed throughout the other phases; however, this distribution does not change the character of the demand placed on computational resources.

Note b: The dotted box (⌐⌐⌐) will involve sequencing (⌐⌐) or not (⌐⌐), according to whether messages are or are not concatenated.

Figure 6: Concurrency and connectivity in different phases of the cycle.

4.4. Dependence of time to compute on number of zones and number of processors

Although not attempting quantitative estimates, we discuss how the time to compute varies with the size of the mesh and the number of processors. Each phase of SIMPLE, as shown in Fig. 6, will be considered separately, as different phases exhibit different dependencies. Several areas of uncertainty confront even qualitative estimation; in particular the detailed operation of a communications facility necessary to a dataflow computer bears on the dependence. This operation has not been modeled to date; for this reason we confine our discussion to two limiting cases. The first case leans toward keeping the communications facility local; i.e. communications between nearest neighbors are stressed. The second case posits a general purpose, global communications facility without worrying about its realizability; the intent is to see what contribution to speed such a facility could make if it were available.

4.4.1. Case definitions

Case 1: connectivity restricted to nearest neighbor plus "tree". As case 1 we posit a restricted communications facility. We imagine processors connected like a two dimensional mesh, with a provision for two-way communications between each zonal processor and its neighboring nodal processors. I.e. the processors are divided into two classes, and a given direct communication is always between two members that are in different classes. Fig. 7 illustrates the connectivity. In addition, we posit additional processors and connections to perform such functions as global sums and the taking of maxima. Each zonal processor is imagined to be a twig of a tree. At nodes of the tree there are processors of a third class (the "tree" class) which can operate to

    a) accept a flow of values from twig to root, operating by program to
        select and pass on the largest value, to sum the incoming values and

pass on the sum, etc, or

b) accept a value flowing from root to twig, providing either for fanout to all zones or for selective routing to a given zone.

For simplicity we imagine that the mesh of the SIMPLE problem is roughly square, and that the D zonal processors are arranged in a square array. To use the configuration of case 1, we imagine that each zonal processor is assigned about N/D contiguous zones; i.e. each zonal processor operates on a "super"-zone of the mesh, as discussed in Sec. 3. As indicated in Sec. 3., the connectivity between super-zones (and the corresponding super-nodes) will show the same pattern as does Fig. 6. The assignment of pieces of dataflow graph to processors is static, and does not change during execution of the program.

Case 2: "general-purpose" communication. Suppose that the dataflow computer has a communications facility that is ideal in the sense that each processor can send a message to any other, with a rate of flow constrained only by the bandwidth of the processors. We define parameters as follows:

T_e = time for a processor assigned to a node or zone of the dataflow graph of SIMPLE to enter a communication into the communications facility, for forwarding to another processor; and

$T_x(D)$ = time for the communication, under the loading conditions at hand, to travel to its destination.

T_x must increase with D at least logarithmically; in practical terms it will probably grow more or less linearly.

The assignment of processors to portions of the dataflow graph can be made as in case 1, but, as will be discussed below, there is an advantage in speed if processors can be reassigned during execution. In

particular, during the Z-sweep of phase 4 it is an advantage to have each

zonal processor assigned to a column of zones of the dataflow graph; during

the R-sweep it is an advantage to have each zonal processor assigned to

a row of zones of the dataflow graph.

4.4.2. Results

Consider the SIMPLE problem for a mesh of N zones, running on a dataflow

computer capable of computing a mesh of D zones without time-sharing of

hardware. The running time will depend on the time to compute a cycle,

as discussed previously. The time to compute a cycle will be a function

of N and D. Examination of the connectivity shown in Fig. 6 for various

phases of the cycle leads to the results shown in Table 1. In Table 1

the parameters T_1 through T_7 will be different for the two cases, and indeed

depend on details of the implementation. However, they do not depend

substantially on N or D.

In order to move to a quantitative estimate, one must both estimate

the parameters T_1 through T_7 for whatever detailed cases are to be judged,

and one must also determine the degree to which pipelining could make the

total cycle time less than the sum of the times for the individual phases.

Although the values of the T-parameters may vary between case 1 and

case 2, it is to be noted that the dependence on N and D is of the same form

for the two computers, except in phase 4, where the configuration of case 2

promises a substantial advantage. This advantage could be obtained as follows.

Assume for simplicity that $N = D^2$ and that the mesh is square, so that there

is one processor for each row of zones and for each row of nodes, or alternatively,

one processor for each column of zones and for each column of nodes. For

the Z-sweep assign each processor to a column, so that one processor must

operate sequentially along its column. Because of the data dependence of the back-substitution method used, this involves no more computing time than would the assignment of one processor per zone and node. At the completion of the Z-sweep, reassign each processor to a row, in preparation for the R-sweep. In this reassignment each processor must send and receive field variables to and from all the other processors of its class. If the communications facility accepts messages as fast as the processors can stuff them in, then we find that the time to reassign is about as follows:

$$\text{Reassignment time} = D\,T_e + T_x(D) \ . \qquad\qquad \text{(Eq. 4.1)}$$

Table 1, under Phase 4, shows the comparison of dependencies achieved with this capability, versus the simpler facility offered in case 1. (Note that T_4 for case 1 is not the same as T_4 for case 2.) It is to be noted that the advantage of the more general communications facility can be realized only if the facility supports "high bandwidth" in the sense of providing for complete exchange of messages among all processors. This total exchange must actually take place to make the scheme work.

The square-root dependence shown for case 1 comes about because in a square array of processors with processing constrained to be sequential along a column (for example), then only one row of processors is in parallel; the other rows are waiting. As D is increased, the length of the row of processors grows as the square root of D.

	Phase 1: boundary value determination.	Phases 2 & 3: calculate nodes and zones except temperature.	Phase 4: calculate temperature — Case 1: communications restricted to nearest neighbor and "tree".	Phase 4 — Case 2: "general purpose, global" communications facility.	Phase 5: time step.	Phase 6: energy totals.	Phase 7: input/output.
Numerical method of SIMPLE	$T_1\sqrt{N/D}$	$(T_2+T_3)\dfrac{N}{D}$	$T_4^{(1)}N/\sqrt{D}$	$T_4^{(2)}N\left(\dfrac{1}{D}+\dfrac{T_e}{T_c}+\dfrac{T_x(D)}{D}\right)$	$T_5\dfrac{N}{D}\log_2 D$	$T_6\dfrac{N}{D}\log_2 D$ (note c)	$T_7 N$; but reducible to something approaching $T_7\log_2 N$ through increasing selectivity of feature extraction.
Change to hypothetical concurrent method for matrix inversion	=	=	(?) $T_4'\dfrac{N}{D}\log_2 D$		=	=	See Secs. 3.3 and 4.5.
Additional change to local determination of time steps	=	=		=	$T_5'\dfrac{N}{D}$	=	

Notes. a) Communication more general than "tree" + "nearest neighbor", even if available, can be effectively used only in phase 4.

b) The issues of estimating the parameters T_1 through T_6 is discussed in Sec. 4.4.

c) Phases 1 through 6 may overlap, so that, as discussed in Sec. 4.4, the cycle time may be less than their sum; in particular the results of phase 6 are not used in any loop calculation and phase 6 can thus easily be pipelined.

d) The mesh is assumed to be roughly square.

Table 1: Form of dependence of time to compute a cycle on number of zones (N) and number of processors (D).

4.5. Input, Output, and Control Over the Extraction of Features

For the first six phases of Table 1, the time to compute diminishes
as the number of processors is increased. But this is not so for phase 7:
SIMPLE requires the "wholesale" shipment of arrays to an external storage
medium. As discussed in Sec. 3.3, the time to transmit N elements over a
single transmission line has a lower bound that is proportional to N, and
moreover is independent of how many processors are brought into the dataflow
computer. Thus the generation of output threatens to consume a time that
could become excessive. This threat can be countered by providing greater
selectivity in reporting; i.e. one programs for the reporting only of
significant features, and avoids communicating "masses of raw data".

In order to avoid swamping analysts even with present computers,
Livermore Laboratory has assembled a powerful facility for computerized
extractions of significant features from masses of data. At present
the approach is to first compute a relatively "general" database, and then
to exercise selectivity in the extraction of features. In order to make
efficient use of a dataflow computer, one must shift to a greater emphasis
on selectivity in generating the output which will form displays and/or
"special purpose" databases. Without bringing selectivity into the generation
of output, the linear growth of time to report an array with the number of
zones is apt to dominate the computation. Even if it does not, the increase
in size in any "general-purpose" database is a serious drawback.

The SIMPLE code offers a small beginning in this direction in the
option in the EDIT subroutine by which one can eliminate the reporting of
nodes and zones that show less than a specified degree of motion. More is
doubtless done in other programs to provide selective reporting, but still

more must be done as the scale of computations is increased. As a specific example along these lines, an analyst could specify that the value of say pressure be reported out for any given zone <u>only</u> if the pressure had changed by more than ten percent since the last report for that zone. Thresholds (e.g. the "ten percent") might be varied during execution.

If selectivity of reporting is made to increase in near proportion to the number of zones, then input and output can be handled with a structure for which Phase 7 of Figure 6 serves as a point of departure. As discussed in Sec. 3.3, however, more trees and more flexible control over them would be of advantage. The goal of selectivity would be to keep the formation of output from overwhelming the analyst and from taking too long. Through increasing selectivity with the number of zones, one can keep the growth rate of the time to form the output from growing as fast as the number of zones; one might hope to contain it to a logarithmic dependence.

Further discussion is outside the scope of this work, but would be appropriate for a future project.

5. Translation of SIMPLE from FORTRAN into VAL

5.1. The balancing of objectives

In developing a code in any language, the following desires are balanced:

.1. Express the algorithm as clearly as possible; and

.2. Make good use of computing resources.

In producing VAL code for a dataflow computer whose hardware is not yet fully specified, it would also be desirable to illuminate constraints on concurrency, and in particular to:

.3. Organize the code so as to make clear which aspects of SIMPLE place which demands on hardware speed and connectivity; and

.4. Extend the SIMPLE problem by sketching more of the input and display functions, because these functions are essential to any actual problem of the SIMPLE type and place demands on both language and hardware not made by other phases of the problem.

In addition, since we are translating from FORTRAN, it would be desirable to:

.5. Make VAL code that can easily be compared with the given FORTRAN code.

These desires conflict in various ways, and any VAL code will reflect a balance between them. In support of items .1 and .3 we group variables into bunches (such as START) in a way that will either decrease efficiency or place extra burdens on compilation. The decrease in efficiency would take the form of sending a longer message where a shorter one would suffice;

concurrency at the level of detail shown in Fig. 6 would not be affected.

In support of items .2 and .3 we have sacrificed item .5 to the extent of introducing new variables (STRESS, GX, GV) that are tensors defined in each zone, in order to demonstrate that the connectivity demanded by SIMPLE in computing the acceleration of each node is only nearest-neighbor, in contrast to the first impression given by lines 580 throug 593 of SIMPLE (1979). Appendix B illustrates demands placed on hardware by various parts of the SIMPLE problem, as expressed in VAL.

In support of .4 we have indicated possible extensions of the VAL language that seem to be needed to help with the extraction of significant features from an array, and with input and output in general; these are:

a. the <u>stream type</u> of value for input and output;

b. the addition of <u>concatenate</u> to the list of <u>forall</u> operations,
    so that a stream can be formed quickly from a sparse array;

c. the addition of an asymmetric <u>merge</u> operation on arrays to help
    in communicating a sparce pattern of change to an array; the effect
    is that one of the two arrays to be merged supplies default values
    which are overridden by non-empty elements of the other array.

d. a form of <u>forall</u> <u>eval</u> <u>max</u> that extracts the lowest index at which
    the maximum value of an array of reals is found, in addition to
    the maximum value itself.

In support of item .5 we use the names of variables as given in the FORTRAN code except where different structures are introduced.

In connection with item .1, it is to be noted that the algorithm of

SIMPLE evolved over decades in a process that was influenced by often conflicting needs for single-step accuracy, stability, and economy; for this reason the algorithm will not be found to show a simple structure, no matter how it is displayed.

The FORTRAN code, including comments, runs some 1749 lines, and a complete translation into VAL would be of roughly the same size. Because the SIMPLE code in FORTRAN is always undergoing minor revisions, as is the VAL language, it seems beside the point to carry through details of translation that duplicate the form of translations already made. We rely on Hirshman (1978) and Woodruff (1979) to demonstrate that many FORTRAN passages can be translated efficiently into VAL; some of thes passages are referred to in what follows. Rather than duplicate their work, we present a more detailed code of the main module of the VAL program for SIMPLE, as a framework in which to view passages that deal with specific acitivities of computation. In this framework we highlight the issues that were encountered in a detailed review of the entire SIMPLE program, focusing on areas, notably input and output, that require further development of the VAL language. Our intent is both to show how the present edition of VAL is sufficient to translate most of the FORTRAN, and to show clearly certain extensions of VAL that appear necessary for a complete translation, including the extension of SIMPLE to provide for the extraction of significant features from arrays.

5.2. Samples of VAL code

5.2.1. Overall form of the VAL translation of the SIMPLE code

As discussed by Ackerman and Dennis (1979) a VAL program consists of a collection of external function modules, each of which may contain internal function modules. One internal module cannot invoke another. We present the VAL code for SIMPLE as a main external function module called SIMPLE_VAL, along with an external function JES which is a table look-up used by two functions internal to SIMPLE_VAL; in addition some external routines presumed to be in a system library are used, such as sine, cosine, and square root. The bulk of the code will be the function modules internal to SIMPLE_VAL.

Each external function module consists of:

header,

type definitions,

external function declarations (e.g. for library supplied utilities)

internal function definitions, and

body.

In the code that follows there will be gaps, indicated by comments, such as passages that can be filled in from the work of Hirshman (1978). Comments will also indicate where a possible extension of the VAL language has been invoked to overcome one or another obstacle of the type discussed in Sec. 5.1.

The program will consist of the external functions

SIMPLE_VAL

JES_VAL

SIN

COS

SQRT   %square root,

and might well be augmented by system utilities to indicate running time, etc.

Because certain features of SIMPLE_VAL are understandable only in the context of JES_VAL, we present JES_VAL first.

5.2.2. JES_VAL

The FORTRAN code of SIMPLE contains a table look-up subroutine named JES. In SIMPLE_VAL this look-up is used by two internal functions: ENERGY_HYDRO and ENERGY_HEAT. Because it is called by two internal functions, we construct the VAL translation of JES as a function external to SIMPLE_VAL.

JES operates on numbers and not arrays; it can be applied fully concurrently be each zonal processor to the elements of a given zone.

An issue in translating is that the FORTRAN version of JES uses many GOTO statements, and these statements are not supported under the more structured philosophy of VAL. Thus the JES code must be re-expressed in an IF-THEN-ELSE form. In arriving at the code displayed below, it was very helpful to first flow chart the FORTRAN CODE. Another issue is that in FORTRAN, JES is employed not by calling "JES", but by calling one or another of the entry points IES1 and IES2; these will correspond to the parameter ENTER in JES_VAL, our VAL equivalent of JES: ENTER = 1 corresponds to IES1; ENTER = 2 corresponds to IES2.

Partly because it uses a method of successive approximations, SIMPLE employs JES several times in the calculation of energy for a single zone. JES (for ENTER=2) returns energy or (for ENTER=1) pressure as a function of temperature (TARG1) and density (RARG1), by means of a table look-up. The table is organized as a two dimensional array of rectangular regions on the (temperature, density)-plane, with a region specified by a pair of integers NT and NR. The returned value is supplied by a procedure that has several steps:

- Search for and find the NT, NR for the region that contains the "point" (TARG1, RARG1);

- Per line 1353, statement 5310 of SIMPLE (1979), evaluate a function of NT and NR to obtain an integer M as index to an array of sets of coefficients -- e.g. AES[M], etc. The set of coefficients for a found M will be used to interpolate.

- Obtain the value to be returned by means of a quadratic interpolation function, using the set of coefficients AES[M], etc.

The running time of SIMPLE (at least for a sequential machine) is significantly reduced by saving NT, NR, and M as NTSV[N], NRSV[N], and MSV[N] for use as trial starting values for the search in the next invocation of JES. In the FORTRAN code NT (along with NR and M) is saved separately according to which of the two entry points (corresponding to ENTER = 1 or ENTER = 2) is invoked. Thus NT is saved in a two-element array, with one element for each possible entry point. We refer to the six saved numbers collectively as SV_REC, where SV_REC is a structure of type SV_REC_type, defined by:

type SV_REC_type = record[NT, NR, M: array[integer]] %.

The structure which we have called SV_REC saved from a given zone supplies trial values for the next invocation of JES, which may be for the same zone, or for a different, usually neighboring zone, as the sequential processor steps from zone to zone. The facilitation of the search is still likely when a shift is made to a neighboring zone, because conditions change little from a zone to its neighbors. The speed advantage accrues because the sequential processor usually last invoked JES either for the same zone

or for a neighboring zone. When the last invocation was for a far-away zone, then SV_REC is no help; this does not affect the answer produced by JES, but does extend the time to find the answer.

Now we turn to the issue of translation for a dataflow computer. Suppose, as suggested in Sec. 3, a dataflow computer has D zonal processors, each assigned to cover a "super-zone" composed of (about) N/D contiguous zones. When N >> D a given zonal processor will step sequentially from zone to zone in a "raster scan" over its N/D assigned zones, just as the sequential computer is specified by the SIMPLE code to scan all N zones. There are three options:

a. Omit the use of SV_REC, and accept a slower look-up (noting that because many look-ups will be done concurrently, the speed is not so important as it was in the FORTRAN code).

b. Create an array of SV_REC's, with one SV_REC for each zone. This option maintains the speed, but as the cost of storing a factor of N/D more SV_REC's than are really needed.

c. Cause each zonal processor to carry one SV_REC along as it steps through its N/D zones.

Option a) is easiest to implement, but is hardly an example of translating power. Option c) is both the most efficient and the most demanding, and is coded in Sec. 5.2.3, where it shows up in initializing SV prior to entering the main loop, and in Sec. 5.2.4 where it is discussed under ENERGY_HYDRO.

The VAL function module follows:

```
function JES_VAL(ENTER: integer; TARG1, RARG1: real; SV_REC: SV_REC_type

                  returns real, SV_REC_type)

type SV_REC_type = record[NT, NR, M: array[integer]]

let % The closing "in" is the           the last line of JES_VAL.

% Set up constants for table; these are provided in the FORTRAN code by

% subroutine SETUP acting via COMMON; we incorporate much of the equivalent

% of SETUP here.

IZES, ITES, IRES: array[integer] := [1: ...], ... ;

TES, RES, AES, ... , PES: array[real] := [1: ...], ... ; % End of set-up part.

EXTT1, EXTR1: real := 1;

N: integer := ENTER; % Change of name to conform to FORTRAN code

NT, NR: integer := SV_REC.NT[N], SV_REC.NR[N];

EXTT2: real := EXTT1 * TARG1;

EXTT, TARG: real, FLAG, NT1: integer :=

if TES[NT] > TARG1 then

    if NT <= ITES[N]then EXTT2 / TES[NT], TES[NT], 0, NT

    else for N1: integer := NT-1

    do if TES[N1] > TARG1 then

          if N1 > IES[N] then iter N1 := N1-1 enditer

          else EXTT2 / TES[N1], TES[N1] , 1,N1 endif

       else EXTT1, TARG1, 1, N1 endif

    endfor

    endif

else if TES[NT+1] > TARG1 then EXTT1, TARG1, 0, NT

     else if NT+2 = ITES[N+1] then EXTT2 / TES[NT+1], TES[NT+1], 0, NT

           else for N1: integer := NT-1

                do if TES[N1+1] > TARG1 then EXTT1, TARG1, 1, N1
```

```
                else if N1+2=ITES[N+1]then EXTT2 / TES[N1+1], TES[N1+1], 1, N1

                     else iter N1 := N1+1 enditer endif

              endif

           endfor

         endif

     endif

endif

EXTR2: real := EXTR1 * RARG1;

EXTR, RARG: real, FLAG2, NR1: integer:=

if FLAG=0 then

   if RES[NR] > RARG1 then

      if NR > IRES[N] then for N1: integer := NR-1

         do if RES[NR] > RARG1 then

               if NR > IRES[N] then iter N1 := N1-1 enditer

               else EXTR2 / RES[N1], RES[N1], 1, N1 endif

            else EXTR1, RARG1, 1, N1 endif

         endfor

      else EXTR2 / RES[NR], RES[NR], 0, NR endif

   else if RES[NR+1] > RARG1 then EXTR1, RARG1, 0, N1

        else if NR+2=IRES[N+1]then EXTR2 / RES[NR+1], RES[NR+1], 0, NR

           else for N1: integer := NR+1

              do if RES[N1+1] > RARG1 then EXTR1, RARG1, 1, N1

                 else if N1+3 > IRES[N+1] then EXTR2/RES[N1+1], RES[N1+1], 1, N1

                      else iter N1 := N1+1 enditer endif

                 endif

              endfor

           endif

        endif
```

```
        endif

else if RES[NR] < RARG1 then for N1: integer := NR

        do if RES[N1+1] > RARG1 then EXTR1, RARG1, 1, N1

            else if N1+3 > IRES[N+1]then EXTR2/RES[N1+1], RES[N1+1], 1, N1

                else iter N1 := N1+1 enditer endif

            endif

        endfor

    else for N1: integer := NR

        do if RES[N1] > RARG1 then

            if N1 > IRES[N]then iter N1 := N1-1 enditer

            else EXTR2/RES[N1], RES[N1], 1, N1 endif

        else EXTR1, RARG1, 1, N1 endif

        endfor

    endif

endif;

M: integer := if FLAG2=0 then SV_REC.M

        else IZES[N]+(ITES[N+1]-ITES[N]-1)*(NR1-IRES[N]+NT1-ITES[N]) endif;

SV_REC1: SV_REC_type :=

    if FLAG2=0 then SV_REC

    else SV_REC replace[NT: SV_REC.NT[N: NT1]; NR:SV_REC.NR[N:NR1];

    M: SV_REC.M[N:M] ]                endif;

FUNC: real   := AES[M] + RARG * (BES[M] + RARG * DES[M])

            + TARG * (CES[M] + RARG * (FES[M] + RARG * GES[M])

            + TARG * (EES[M] + RARG * (HES[M] + RARG * PES[M])));

FUNC1 : real := if ENTER=1 then FUNC * EXTT * EXTR

                else FUNC * EXTT endif

in %closes "let" on line 2 of JES_VAL

FUNC1, SV_REC1 endlet endfun % End of function JES_VAL
```

5.2.3. SIMPLE_VAL

SIMPLE_VAL is the main module -- i.e. the overall framework -- for the VAL code translation of SIMPLE. Because the functions internal to this module correspond to roughly 25 pages of FORTRAN code, the section of internal function definitions is abbreviated to a list of headers, and a discussion of salient features of these modules will be found in Sec. 5.2.4. The code that follows is a detailed statement of the overall structure of the VAL translation of SIMPLE.

```
% Header:
% Note presumed language extension to "stream" type for input and output.
function SIMPLE_VAL(INPUT_A:   start-type;   INPUT_B: stream
    [correction_type] returns stream[out_phys_type],
    stream[out_cycle_type] , stream[out_edit_type],
    stream out_condition_type )


%type definitions:
type vector = record[R,Z: real];
type zonal = array[array[real]];   .
type zone_tensor = array[array[record[E,W: vector]]];
type nodal = array[array[vector] ] ;
type node_scalar = array[array[real]];
type start_type = record[DTNPH, TFLR, EDDT, PØ, EØ, RHOØ, DTMIN,
    DTMAX, TMAX, CØF, C1F, GAM: real; BC: record[U, D, L, R: integer];
    LIM: record[KN, KX, LN, LX, DS: integer]; NCP: integer];
```

```
% As shorthand we shall write "STATE" and "state_type" to refer to
% a list of the variables that define the state of the computation:
% state_type = list[DTNPH, DTN, TNUP, ENCG, EDTIME, EDDT: real; NYCL:
%      integer; P, Q, RHOJ, E, S: zonal; X, V: nodal; GX: zone_tensor;
%      DTMIN, DTMAX, TMAX, CØF, C1F, GAM, EDDT, TFLR: real; NCP: integer]

  type out_phys_type = "state_type";

  type out_cycle_type = record[NYCL: integer; DTNPH, TE, ENC, SKE, HN, WN,
                                      ENCG: real; DTEN, DTC2: record[DT: real;
                                      K, L: integer]];

  type out_edit_type = "state_type";

  type out_condition_type = stream; % language extension

  type correction_type = stream;

  type lim_type = record[KN, KX, LN, LX, DS: integer]; % 4 fields correspond
% to FORTRAN code KMN, KMX, LMN, LMX; DS describes implementation for
% the implementation-dependent use of JES_VAL shown in ENERGY_HYDRO.


  type SV_REC_type = record[NT, NR, M: array[integer]];
% SV_REC discussed in Sec. 5.2.2 in connection with JES_VAL.


  type SV_type = array[array[SV_REC_type]];   % Because of our choice of
% option c) of Sec. 5.2.2, the array SV of type SV_type will have
% dimensions of LIM.DS by LIM.DS, where  LIM.DS squared is D, the
% number of zonal processors of the dataflow computer.  If option b)
% were used, then LIM.DS would not have to appear in the program, and
% the array SV would have N (number of zones in mesh) elements instead
% of D elements.
```

% external function declarations:

external JES_VAL(ENTER: integer; TARG1, RARG1: real; SV_REC:

    SV_REC_type returns real, SV_REC_type)

external sin(DUMMY: real returns real)

external cos(DUMMY: real returns real)

external sqrt(DUMMY: real returns real) % square root.

% The bodies of the internal function definitions are omitted here; the

% headers are listed for all internal functions of SIMPLE_VAL:

%   INITIALIZE(START: start_type returns "state_type")

%   EDIT(STATE returns edit_type)

%   BOUNDARY_PROJECT(P, Q, RHOJ: zonal; X: nodal; GX: zone_tensor; LIM:

%   lim_type returns zonal, zonal, zonal, zone_tensor)

%   VELOCITY(V: nodal; P, Q, RHOJ: zonal; GX: zone_tensor; DTN: real;

%   LIM: lim_type returns nodal)

%   POSITION(X,V: nodal; DTNPH: real; LIM: lim_type returns nodal)

%   HWORK(X, V: nodal; P, Q: zonal; DTNPH: real; LIM: lim_type returns real)

%   ZONE_GEOM(X, V: nodal; MASS, S: zonal; LIM: lim_type returns

%   zonal, zonal, zonal, zonal, zone_tensor, zone_tensor)

%   ENERGY_HYDRO(E, P, AJ, RHO, DVOL, MASS: zonal; GX, GV: zone_tensor;

                  SV: SV_type; DTNPH, CØF, C1F, GAM, DTMAX: real; LIM:

                  lim_type returns zonal, zonal, zonal, zonal, SV_type)

%   HYDRO_TOTAL(V: nodal; MASS, E: zonal; LIM: lim_type returns real, real, real)

%   ENERGY_HEAT(E, RHO, AJ, TEMP, MASS: zonal; X: nodal; SV: SV_type;
                DTNPH, TFLR: real; LIM: lim_type returns zonal, zonal, zonal,
                node_scalar, node_ scalar, SV_type)

%   HEAT_TOTAL(E, TEMP, MASS: zonal; CBB, DBB: node_scalar; DTNPH, HN: real;

%   LIM: lim_type returns real, real)

%   TIME_STEP(TSO, YE: zonal; X: nodal; DTNPH, DTMAX, CØF, C1F, GAM: real;

%   LIM: lim_type returns real, real, real, real)

%   PHYS_REPORT("STATE": "state_type" returns "state-type")

%   CYCLE_REPORT(YE, TSO: zonal; NYCL: integer; TNUP, DTNPH, TE, ENC,

%   SKE, HN, WN, ENCG: real; LIM: lim_type returns

%   out_cycle_type)

%   MODIFY("STATE": "state_type"; DUMMY: correction_type returns

%   "state-type")

```
% body of SIMPLE_VAL

% The gross plan of the body is

% for STATE: state_type:= INITIALIZE(first(INPUT_A));

%      OUT_PUT: stream:= null

% do if (condition) then OUT_PUT

%     else iter STATE:= main_cyle(STATE) enditer

%     endif

%     endfor

% In the detailed presentation that follows we split "STATE" into

% its fields (as given in the section of type definitions), and split

% "main_cycle" according to the phases illustrated in Figure 6:

for START: start_type:= first(INPUT_A); % read input stream

    STATE: "state_type":= INITIALIZE(START);

    OUT_PHYS: stream[out_phys_type]:= null;

    OUT_CYCLE: stream[out_cycle_type]:= null;

    OUT_EDIT: stream[out_edit_type]:= EDIT(STATE);

    CORRECTION: stream := INPUT_B;

    HN, WN: real := 0.;

    % Set up temporary variables, other than those covered in STATE,

    % needed for main cycle:

    AJ, DVOL, TEMP, TSO, YE: zonal := array_fill(LIM.KN + 1, LIM.KX,

        array_fill(LIM.LN + 1, LIM.LX, 0.));

    GV: zone_tensor := array_fill(LIM.KN + 1, LIM.KX,

        array_fill(LIM.LN + 1, LIM.LX, record[E, W: record[R, Z: 0.]]);

    CBB, DBB: nodal := array_fill(LIM.KN, LIM.KX, array_fill

        (LIM.LN, LIM.LX, record[R,Z: 0.] ));

    DTEN, DTC2, SKE, ENH, TE, ENC: real :=0.
```

```
LIM: lim_type := START.LIM;

% Set up array of SV_REC's to conform to option c) of Sec. 5.2.2.
% Let DS be the greatest integer such that DS*DS  = D, where D is the
% number of zonal processors, as discussed in Sec. 3.
SV: SV_type :=
    let DS: integer := LIM.DS % implementation-dependent parameter.
    in array_fill(1, DS, array_fill(1, DS, record NT: array_fill(1, 2, 0);
    NR: array_fill(1, 2, 0); M: array_fill(1, 2, 0); EXTR: 0.)) endlet;



do if DTNPH < DTMIN | TNUP > TMAX then
        let OUT_CONDITION: stream :=
            if DTNPH < DTMIN then "DT_STOP" || NYCL ||TNUP || DTNPH || DTMIN
            else "STOP TMAX" || NYCL || TNUP || TMAX  endif
        in OUT_PHYS, OUT_CYCLE, OUT_EDIT, OUT_CONDITION endlet
    else iter
% Phase 1 of cycle (see Fig. 6 for description of phases):
P, Q, RHOJ, GX := BOUNDARY_PROJECT (P,Q, RHOJ, X, GX, LIM);

% Phase 2 of cycle:
V := VELOCITY(V, P, Q, RHOJ, GX, DTN, LIM); % vector velocity
X := POSITION(X, V, DTNPH, LIM);  % vector position

% "WN" part of Phase 6:
WN := HWORK(X, V, P, Q, DTNPH, LIM) + WN;

% Phase 3 of cycle:
RHO, AJ, DVOL, S, GX, GV := ZONE_GEOM(X, V, MASS, S, LIM);

E, P, Q, TEMP, TSO, SV := ENERGY_HYDRO(E, P, AJ, RHO, DVOL, MASS,
                            GX, GV, SV, DTNPH, CØF, C1F, GAM, DTMAX, LIM);
```

```
% Hydro part of phase 6:

SKE, ENH, TE := HYDRO_TOTAL(V, MASS, E, LIM);


% Phase 4 of cycle:

E, RHOJ, YE, CBB, DBB, SV := ENERGY_HEAT(E, RHO, AJ, TEMP, MASS, X, SV,

                                             DTNPH, TFLR, LIM);


% Heat part of phase 6:

ENC, HN := HEAT_TOTAL(E, TEMP, MASS, CBB, DBB, DTNPH, HN, LIM);


% Phase 5 of cycle:

DTN, DTNPH, DTC2, DTEN := TIME_STEP(TSO, YE, X, DTNPH, DTMAX,

                                       CØF, C1F, GAM, LIM);


% Phase 7 of cycle (output and corrective input):

OUT_PHYS, EDTIME :=

    if TNUP < EDTIME then OUT_PHYS, EDTIME

    else OUT_PHYS || PHYS_REPORT(STATE), EDTIME + EDDT endif;

NYCL := NYCL + 1;

OUT_CYCLE :=

    if MOD(NYCL, NCP)~= 0 then OUT_CYCLE

    else OUT_CYCLE || CYCLE_REPORT(NYCL, TNUP, DTNPH, YE, TSO,

        TE, ENC, SKE, HN, WN, ENCG) % lines 766-773 of FORTRAN

    endif

STATE, CORRECTION :=

    if CORRECTION = null then STATE, CORRECTION

    else MODIFY(STATE, first(CORRECTION)), rest(CORRECTION)

    endif
```

```
% An alternative approach to output would be to extract significant
% features.  For example, we illustrate a report of pressure for only
% those elements of the array P that have changed by at least 10
% percent since they were last reported.  We assume an array P_LAST
% as an iteration variable to carry the "last reported" value of P:
P_LAST, OUT_PHYS_SELECTIVE :=
    if TNUP < EDTIME then nil %language extension for iteration variables
    else let COND: array[array[boolean]] :=
        forall K in [LIM.KN + 1, LIM.KX], L in [LIM.LN + 1, LIM.LX]
        construct    ABS((P[K,L] - P_LAST[K,L])/MAX(EPS, P_LAST[K,L]))< .1 endall
          in forall K in [LIM.KN + 1, LIM.KX], L in [LIM.LN + 1, LIM.LX]
              construct if COND then P_LAST[K,L]
                        else P[K,L] endif endall, OUT_PHYS_SELECTIVE ||
              forall K in[LIM.KN + 1, LIM.KX], L in [LIM.LN + 1, LIM.LX]
              eval concatenate %language extension
                  if COND then null
                  else record[P: P[K,L]; K: K; L: L] endif endall
        endlet
    endif
% end of example of feature extraction
enditer
endfor
endfun  % SIMPLE_VAL
```

5.2.4. Discussion of functions internal to SIMPLE_VAL

INITIALIZE includes code like the modules GENBC and GENPOS of Hirshman (1978),
along with code of the form, say for pressure,

    % P: zonal :=
    array_fill(LIM.KN + 1, LIM.KX , array_fill(LIM.LN + 1, LIM.LX, START.PØ)).

EDIT is straightforward to translate, except for one demand which it places
on the language: one needs to extract not only the maximum element of an
array (as can be done with forall eval max) but also the K,L coordinates
at which the maximum is found. Efficient support of this need requires
hardware and language attention.

BOUNDARY_PROJECT includes the module GEOMETRY of Hirshman, the filling of
P, Q, and RHOJ arrays (where RHOJ[K,L] = RHO[K,L] * AJ[K,L]), and the
calculation of GX for boundary zones. The calculation of GX for interior
zones is done in ZONE_GEOM, and is discussed in Appendix B.

VELOCITY: see Appendix B, where connectivity of the flow of data is discussed.

POSITION is like Hirshman's module HYDRO; see also Appendix B.

HWORK is essentially Hirshman's module of the same name.

ZONE_GEOM produces AJ and S like the module GENAREA of Hirshman, and also
produces GX and GV by the algorithm discussed in Appendix B.

ENERGY_HYDRO contains parts like NEWE and NEWQ of Hirshman. However,
NEWQ can be recast to use GX and GV in place of X and V, with the result
that the calculation for a given zone draws only on values of that zone;
i.e. no node-to-zone communication is required for the computation of Q when

GX and GV are made available from ZONE_GEOM.

Subroutine TEMPCAL of the FORTRAN code can be translated readily into a function module internal to ENERGY_HYDRO. Both via TEMPCAL and directly, ENERGY_HYDRO calls the external function module JES_VAL to compute pressure (from JES_VAL(1, TEMP, RHO, SV_REC)) and energy (from JES_VAL(2, TEMP, RHO, SV_REC)). The value SV_REC supplied to JES_VAL is in effect a <u>hint</u> where to start searching in a table; the value supplied does not affect the numerical results produced by JES_VAL, but it does affect the time to execute JES_VAL.

If option b) os Sec. 5.2.2 were selected, coding into VAL would be easier because there the array SV would have N elements and be of the same shape as P, RHO, etc. For that option a typical use of JES_VAL would be the production of a trial pressure P1, as in:

.1
    P1, SV: zonal :=

        forall K in [LIM.KN+1, LIM,KX], L in [LIM.LN+1, LIM.LX] construct

        JES_VAL(1, TEMP[K,L], RHO[K,L], SV[K,L]) endall; %.

Instead of using option b), we have chosen option c) as an example of the kind of demand on expressive power that occurs in tailoring an algorithm to an implementation. As discussed in Sec. 5.2.2 option c) saves storage by taking SV to be an array of only D (= number of zonal processors) elements; this can be much smaller than the N-element array used in option b). To express the N-element array P1 as a function of a D-element SV, it appears necessary to first create a <u>partitioned</u> array equivalent to P1, with a block of this partitioned array corresponding to an element of SV.

The N interior zones of the mesh constitute a two-dimensional array of (LIM.KX - LIM.KN) by (LIM.LX - LIM.LN) elements. For simplicity we assume that both of these dimensions are exactly divisible by LIM.DS,

where $D = (LIM.DS)^2$ is the number of zonal processors used, and we assume a physical configuration of a square array of LIM.DS by LIM.DS zonal processors.

Each zonal processor is to be assigned a rectangular "super-zone" of the mesh, consisting of KS by LS contiguous zones, where

.2
$$KS = (LIM.KX-LIM.KN)/LIM.DS$$

and
$$LS = (LIM.LX-LIM.LN)/LIM.DS .$$

In place of .1 one expressed an N-element P1 in terms of a D-element SV, where one elemnt of SV corresponds not to one element of P1, but rather to a block of KS by LS elements of P1. Let P_BLOCK be a partitioned array equivalent to P1; that is, while P1 is a 2-dimensional array of reals, P_BLOCK is an array of LIM.DS by LIM.DS "little" arrays, each with KS by LS real elements, so that P_BLOCK must be a 4-dimensional array of reals. Option c) demands that:

• computation proceed in each of the D blocks of P_BLOCK concurrently, and

• within a given block, computation proceed in a raster scan sequentially.

The correspondence between P1 and PBLOCK is:

.3
$$P1[K1*KS + K0,L1*LS+L0] = P\_BLOCK[K1,L1,K0,L0] .$$

In other words, K1,L1 tell which block, and K0,L0 tell which element within the block. It follows that (with the VAL convention for downward rounding of integer division) the [K,L]element of P1 is given by

.4
$$P1[K,L] = P\_BLOCK[K/KS, L/LS, MOD(K,KS), MOD(L,LS)] .$$

The VAL code for producing P1 and SV in accordance with option c) follows:

```
P1: zonal, SV: SV_type :=
let P_BLOCK: array[array[array[array[real]]]] , SV1: SV_type :=
forall K1 in [1, LIM.DS], L1 in [1, LIM.DS]
        KS: integer := (LIM.KX-LIM.KN)/LIM.DS; % Assume exactly divisible
        LS: integer := (LIM.LX-LIM.LN)/LIM.DS; % "
construct % P_BLOCK[K1,L1] is itself a 2-dimensional array.
    for BLOCK: array[array[real]]:= array_empty[array[real] ; % Element of P_BLOCK.
        SV_REC1: SV_REC_type := SV[K1,L1];
        KØ: integer := 1
    do if KØ > KS then BLOCK, SV_REC1
        else iter BLOCK, SV_REC1 :=
                let BCOL: array[real], SV_REC2: SV_REC_type :=
                    for BCOL1: array[real]:= array_empty[real];
                        SV_REC3 : SV_REC_type := SV_REC1;
                        LØ: integer := 1
                    do if LØ > LS then  BCOL1, SV_REC3
                        else iter BCOL1, SV_REC3 :=
                                let P_EL: real, SV_REC4: SV_REC_type :=
                                    JES_VAL(1, TEMP[K1*KS+KØ, L1*LS+LØ],
                                    RHO[K1*KS+KØ, L1*LS+LØ], SV_REC3)
                                in BCOL1[LØ: P_EL], SV_REC4 endlet;
                                LØ := LØ + 1
                        enditer
                    endif
                endfor
                in BLOCK KØ: BCOL , SV_REC2 endlet;
```

```
        KØ := KØ + 1;

      enditer

    endif

  endfor

endall
in % P1: zonal, SV: SV_type :=
  forall K in [LIM.KN+1,LIM.KX], L in [LIM.LN+1, LIM.LX] construct
  P_BLOCK[K/KS, L/LS, MOD(K,KS), MOD(L,LS)] , SV1
endlet % Completes production of P1 and SV.
```

Because of the explicit reference to LIM.DS, a parameter of
implementation, this example gives a glimpse of the type of expression
needed when a programmer assists in compilation. It is generally recognized
that hardware can be used more effectively if the programmer tailors the
program to it. In simple cases one hopes that the algorithm will not have
to be changed to effect such tailoring, but we have just seen a case in
which the algorithm (though not its numerical result) <u>did</u> change. To
facilitate compilation of the whole SIMPLE code, one might well express
all the arrays in blocked (i.e. partitioned) form for internal computation.
If this were done then the conversion to 2-dimensional form would not
be done as part of the above example, but would be deferred to the
generation of output, as in the module PHYS_REPORT of SIMPLE_VAL.

HYDRO_TOTAL, like HWORK, is straightforward, being essentially the
execise of the construct <u>forall-eval-plus</u>.

ENERGY_HEAT is the main bottleneck in the SIMPLE problem, because of
the sequencing constraints due to the back-substitution method chosen
for solving for heat flow. The sequencing constraints are illustrated

in Appendix B, Fig. B.1. The constraints are in the "R-sweep" and "Z-sweep" portions of subroutine CONDUCT of the FORTRAN code of SIMPLE. This code steps from one element of an array to another, using results of a previous element to calculate a next element.

Subroutine CONDUCT saves TEMP as TS in line 1586, and then restores TEMP to TS in line 1673, so that after the execution of CONDUCT, TEMP is unchanged; what is calculated is really a temporary variable which we call TEMP1 in the code below. Its use is not to get a new TEMP, but rather to help in adjusting E to account for heat flow. The FORTRAN code partially inializes arrays A and B outside of the sweeps; we incorporate this initialization into the sweeps. The VAL arrays CBB and DBB are like those of the FORTRAN code, but re-indexed to clarify the connectivity actually required (see note be following the VAL code below). The production of TEMP1 in the VAL code for ENERGY_HEAT would then appear inside a LET construct as follows:-

```
% Z-sweep  (per line 1612 of the FORTRAN code of subroutine CONDUCT)
TEMP1: zonal := let TEMP2: zonal :=    % Z-sweep calculates TEMP2
forall K in [LIM.KN + 1, LIM.KX] construct
let A, B: array[real]:= % range over L
    for L: integer := LIM.LN +1;
        ACOL, BCOL: array[real]:= array_fill(LIM.LN, LIM.LX, 0.), TEMP[K]
    do if L > LIM.LX then ACOL, BCOL
        else let DUM1: real := SIG[K,L] + CBB[K,L] + CBB[K,L-1] * (1 - ACOL[L-1])
            in iter ACOL, BCOL := ACOL[L: CBB[K,L]/ DUM1], BCOL[L: SIG[K,L] *
                TEMP[K,L] + CBB[K,L-1] * B[K,L-1] /DUM1];
                L := L+1
enditer endlet endif endfor
```

```
% ... ALPHA, BETA FORWARD

in for L: integer := LIM.LX; TCOL: array[real]:= TEMP[K]

    do if L < LIM.LN + 1 then TCOL

        else iter TCOL := TCOL[L: A[L] * TCOL[L+1] + B[L]];  L := L-1 enditer

endif endfor endlet endall  % end of Z sweep; returns TEMP2

in % Feed TEMP2 through R-sweep to produce TEMP1:

% R sweep

let A, B: array[array[real]] :=

    for K: integer := LIM.KN + 1; A2D, B2D: array[array[real]] :=

        array_fill(LIM.KN, LIM.KX, array_fill(LIM.LN, LIM.LX, 0.)), TEMP2

    do if K > LIM.KX then A2D, B2D

        else let ACOL, BCOL: array[real] :=

                forall L in [LIM.LN + 1, LIM.LX] DUM1: real := SIG[K,L]

                 + DBB[K,L] + DBB[K-1,L] * (1- A2D[K-1,L])

                construct DBB[K,L] / DUM1, SIG[K,L] * TEMP2[K,L] +

                        DBB[K-1,L] * B2D[K-1,L] / DUM1

                endall

            in iter A2D, B2D := A2D[K: ACOL], B2D[K, BCOL];

        enditer endlet endif endfor

% ALPHA, BETA FORWARD SWEEP

in for K: integer := LIM.KX; T2D: array[array[real]] := TEMP2

    do if K < LIM.KN + 1 then T2D

        else iter T2D := T2D[K:

                forall L in [LIM.LN + 1, LIM.LX]

                construct A[K,L] * T2D[K+1,L] + B[K,L]

                endall]; K := K-1

enditer endif endfor endlet endlet % Returns TEMP1
```

Notes:

a. In VAL the syntax for operating on a two-dimensional array with
    a _forall_ construct over one index and a _for-iter_ over the other
    index is different according to which index is subjected to which
    construct. For this reason the Z-sweep and the R-sweep, which
    look much the same in FORTRAN, look different in VAL.

b. The FORTRAN code uses an awkward convention in indexing CBB and DBB,
    with the result that there appears to be more coupling of array
    elements than is in fact the case; to clarify this we write
    CBB[K,L] in place of what in the FORTRAN code would be written
    CBB[K-1,L]; similarly we write DBB[K,L]in place of DBB[K,L-1].

c. In FORTRAN only one edge of the array A is initialized prior to
    the loop; in VAL it was convenient to initialize the whole array.
    The VAL code re-initializes A in the R-sweep. This is permissible
    because although the A array is operated on in the Z-sweep, the
    only column that matters (i.e. LIM.KN) is not changed in the Z-sweep.

HEAT_TOTAL uses _forall_ _eval_ _plus_.

TIME_STEP combines Hirshman's module TINCR with the calculation of DTEN,
which in the FORTRAN is done in subroutine CONDUCT. Calculation of KC, LC,
KEN, and LEN is not done in TIME_STEP, but is deferred to CYCLE_REPORT.

PHYS_REPORT is similar to EDIT.

CYCLE_REPORT is straightforward except for needing the coordinates of an
array where a maximum or minimum value is found, as was the case with EDIT.

MODIFY is an augmentation of SIMPLE to allow for real-time interaction with an analyst; e.g. MODIFY is to provide for receiving a change in say DTMAX, or even for receiving an entire "STATE", as would be needed to restart the computation after an analytic "catastrophe".

6. Conclusions and Possible Next Steps

6.1. Speed, input-output, and expression of the abstract algorithm

As shown in Table 1, except for outputting results, the application of D processors configured as a dataflow computer can reduce the execution time of the SIMPLE code by a factor of at least $D^{\frac{1}{2}}$. The sequencing constraints that limit improvement to this factor occur in the calculation of heat flow, as illustrated in Fig. 6. These constraints stem from the method chosen in the SIMPLE code for the inversion of a tri-diagonal matrix: back-substitution. It would appear feasible to find or develop a method with weaker sequencing constraints. If this were done, then all phases of the program, except output, would execute in times that decrease at least as D/log D with increasing D.

As discussed in Sec. 4.5, the outputting of results called for in the SIMPLE code amounts to a "dump" of raw data. There is a minimum time for such a dump that grows with the size of the mesh and is independent of D. As discussed in Sec. 4.5 and illustrated at the end of Sec. 5.2.3, it appears essential to pre-process the data so as to extract significant features. If this is done, then output need not be a bottleneck.

The VAL language is demonstrated as satisfactory for the expression of the SIMPLE problem as an abstract algorithm, provided that certain extensions are made in it. These extensions are listed in Sec. 5.1 and their use is shown in Secs. 5.2.3 and 5.2.4. The need for additional extensions to promote efficiency of execution is discussed below.

6.2. Implications of the spatio-temporal structure of the algorithm

Following Holt (1979) we have analyzed the SIMPLE problem as given
in an abstract algorithm expressed first in FORTRAN and then translated into
VAL. The algorithm expressed in either language is called 'abstract' when it
is viewed as independent of physical arrangements in space and time for its
execution. Our analysis of the SIMPLE algorithm in terms of role diagrams
reveals spatial and temporal structure which will have to be found in any and
all implementations. For example, by tracing through the algorithm for
possible references to computational variables we discover the existence of
algorithm-defined times when some number n of such variables must be co-maintained.
This in turn implies that in any implementation of the algorithm there will
have to be available, for some period, a space large enough to hold n values.
(As the algorithm is to be executed by electronic circuits, this number n places
a lower bound on the physical space which the algorithm can occupy.) To
be more specific, E[J,K], P[J,K], Q[J,K], etc. meet in a zone and phase
shown in Fig. 6 and in a relational sense define a time and location.

As a second example, we discover in Fig. 6 that for any instruction
of the main loop there are times-- i.e. phases -- when a given instruction
may be executed and times when it certainly will not be. In other words one
can determine prior to execution and independent of implementation that in
any given phase a certain large majority of the instructions of the main loop
will not be called. This property can be used both to guide compilation and
also to guide the design of hardware for a dataflow computer: it suggests
a programmable instruction cell that can make ready first one instruction
and then another, much like a sequential processor.

Finally the discussion of Sec. 3 and Figs. 3, 4 and 6 show that only
a few of the myriad possible patterns of communication are actually needed

for a set of processing resources to execute the SIMPLE problem. In configuring a dataflow computer there are many possible alternatives for the arrangement of processing units, instruction cells, packet memory, and communications resources. Different arrangements offer different advantages for different problem classes, and place different demands on compilation. As discussed in Sec. 3, any hardware arrangement will reflect compromises which will detract from the execution of some classes of problems. Prior to large-scale investment, these relations between physical arrangement and problem class need to be examined in connection with various sample problems.

6.3. The balance between programming ease and efficient use of hardware

As a first step in exploring relations between hardware and problem class, VAL was employed to help express a problem in hydrodynamics in support of two anticipated tasks, relative to a dataflow computer that is not yet fully specified:

.1. the design task of choosing a physical arrangement of hardware resources suitable to the SIMPLE problem; and

.2. the compilation task of mapping the coded problem into machine instructions appropriate to a given physical arrangement of resources.

Both tasks concern the mapping of a problem onto physical resources. The mapping is done in two steps: coding in a source language (VAL); followed by compilation which maps the source language into machine instructions. Historically a source language has been intended for the expression of a

problem as an abstract algorithm -- 'abstract' meaning that the algorithm
was not tied to a particular physical arrangement of resources. But note:

.3. To achieve <u>efficient</u> use of resources a programmer must allow

for at least some features of implementation (e.g. "multiply" takes

longer than "add").

.4. If the physical arrangement changes too much, a given source

language become inappropriate.

Indeed concurrently operability of resources contributed to the need to
express concurrency in the problem, and hence to the need for VAL; i.e. VAL
is superior to FORTRAN in expressing concurrency. A source language is
shaped in part by assumptions concerning the physical arrangement of
computational resources. FORTRAN was designed to facilitate a two-step
mapping of a problem to machine instructions. In step one FORTRAN is used
to map the problem essentially into instructions for a machine that is
an <u>idealized</u> sequential computer -- idealized for instance in that it is
imagined to have a random-access memory so big as not to be a constraining
factor. In step two the FORTRAN code is compiled into machine code for
an actual machine that departs in limited ways from the idealization --
e.g. by using a "small" random-access memory backed up by secondary storage.

As FORTRAN corresponds to an idealized sequential computer, VAL
presently corresponds to an idealized dataflow computer -- e.g. a dataflow
computer imagined to have so many instruction cells that the number poses
no constraint on how a problem might be executed. Note that:

.5. A program like SIMPLE is a major task for programmers who can

afford to learn the salient features of implementation;

.6. The program is expected to run many hours per execution, and to be executed many times on a machine that costs enough to justify a large investment in efficient execution;

.7. The program is written to answer questions of physics that are progressively better answered as larger mesh sizes become executable in a day's run; the need for answers to these questions justifies a large investment in speed of execution.

Whatever hardware design is chosen, the resources of a dataflow computer will be more complex than those of a sequential computer, and less susceptible to fully automated resource allocation. Within the dataflow context, the balance between ease of programming and efficiency weighs more toward the demand for efficiency. For problems of the SIMPLE type it appears unwise to force a separation between source-language programming and resource allocation. Some current languages -- e.g. PL/1 -- provide facilities for the control of resources; however these facilities are added ad hoc to a language that conceptually is inhospitable to the expression of physical arrangements in time and space. Because VAL encompasses the expression of concurrency, it offers at least a chance of extension to cover the control of resources in a more systematic way. The discussion of ENERGY_HYDRO in Sec. 5.2.4 illustrates a related issue, the adaptation of the algorithm to a particular physical arrangement.

6.4. Extending VAL to support resource allocation

We have seen that the SIMPLE problem has spatio-temporal structure
that is germane to physical design, and for a given design, germane to the
allocation of physical resources to execute the problem. Presently, a VAL
program is thought of as having a "meaning" only to the extent that it
defines a dataflow graph at the descriptive level of machine instructions.
At this level of description the dataflow graph of SIMPLE is an enormous
lacework, with something on the order of a thousand computational events
per zone, times thousands of zones. If a compiler works only from a dataflow
graph at this level of detail, is it reasonsable to imagine that it could
efficiently distribute all the instructions throughout the "space-time" of
the computational resources?

One might hope for some future "genious" to design such a compiler,
but there is another approach:

.1. Recognize that compilation will in fact use higher-level and/or
auxiliary descriptions of the problem in allocating resources; and

.2. Extend the programmer's task and his power of expression -- VAL --
to express properties of the problem that can greatly reduce the
burden of compilation -- properties such as those expressed in
the role diagram of Fig. 6.

In this approach the programmer would be supported in structuring the problem
in a way that eases compilation for a given machine organization. This
requires that the programmer be more explicit in guiding the "when" and
"where" of program execution. It might be objected that such guidance depends
too much on the details of a particular implementation, but this is not

necessarily so. There is a middle ground, where the programmer would formally express the information now conveyed by Fig. 6. The "where" implied by a "zone" of Fig. 6 is not directly a "machine location", but rather a relational location inherent in the SIMPLE algorithm. In that algorithm E[J,K], P[J,K], Q[J,K], etc. meet many times, and in a relational sense meetings define "times and locations" -- e.g. Zone[K,L] of Fig. 6. In essence we see the programmer as calling the compiler's "attention" to grosser regions of a dataflow graph than appear at a machine-instruction level of description. The compiler would thus block out the assignment of gross regions to resources in a first phase, and then subsequently deal with further details. To pursue this course additional effort is needed to:

.3. Bring under control the expression of the space-time aspect of an algorithm at different levels of detail, so as to guide the algorithm toward a particular machine organization;

.4. Show what changes would be needed for VAL to express such aspects;

.5. Evaluate the advantage of expressing SIMPLE and other examples in this way with respect to:

a. what suggestions are offered for the organization of the resources of a dataflow computer; and

b. how to distribute the burden of computing a problem over a given organization of resources.

REFERENCES

Ackerman, W. B. and J. B. Dennis (1979) "VAL -- A Value-Oriented Algorithmic Language; Preliminary Reference Manual" Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts 02139 (March 22).

Crowley, W. P., C. P. Hendrickson and T. E. Rudy (1978), "The SIMPLE Code", Lawrence Livermore Laboratory Report UCID 17715, February 1.

Dennis, J. B. (1978) "Data Flow Computer Architecture", Computation Structures Group Memo 160, Laboratory for Computer Science, Massachusetts Institute of Technology (May)

Hirshman, D. S. (1978) "SIMPLE, A Lawrence Livermeor Laboratories Program Translated into Data Flow Language", Massachusetts Institute of Technology, Laboratory for Computer Science, Computation Structures Group (May 18)

Holt, A. W. (1979) "Roles and Activities, A System for Describing Systems" (Incomplete draft) Boston University, Academic Computing Center, 111 Cummington Street, Boston, Mass. 02215

Misner, C. W., K. S. Thorne and J. S. Wheeler (1970) "Gravitation", W. H. Freeman and Co., San Francisco.

Morse, P. M. and H. Feshbach (1953) "Methods of Theoretical Physics" McGraw-Hill Book Co, New York.

SIMPLE (1979): FORTRAN code of Lawrence Livermore Laboratory, Edition of February 12 as provided by John Woodruff. (Reproduced in App. C.)

Woodruff, J. P. (1978) VAL code for one-dimensional hydrodynamics (Edition of December 4).
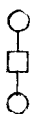
Appendix A: Interpreting Role Diagrams

<u>SECTION DIRECTORY</u>

<u>Section</u>
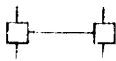
A.1. Vertical string as path of a role player.

A.2. Tokens
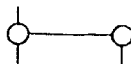
A.3. Circuits.
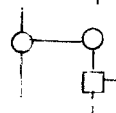
A.4. Initialization and termination.

A.5. Fragments

A.6. Coincident activity of multiple role players.

A.7. Invariance of value.
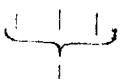
A.8. Branching to alternative consumers.

A.9. Steering.

A.10. Encoding
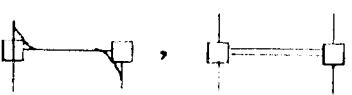
A.11. Decoding

A.12. Merging from alternative producers.

A.13. Bundling.
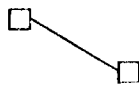
A.14. Unbundling.

A.15. Compression of representation.

A.16. Copying.

A.17. Saving an old value.

A.18. Operations (+, -, etc.)

A.19. Buffered communication.

Appendix A: Interpreting Role Diagrams

Throughout the report we have used role diagrams, invented by A. W. Holt (1979) to show the flow of values carried by physical actors. The notation presented here allows us to distinguish participations of actors in activities according to whether they are coincident, concurrent, alternative, or sequenced.

The interpretation of role diagrams differs from that of dataflow graphs in that the former is based on this attitude: anything that is (e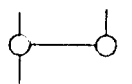ven a value) must be someplace. Hence the flow of a value is a flow of effect over physical actors. A role diagram can be partitioned into strips; each strip is a locality in system space, and thus a place where some actor is resident.

A.1: A vertical line is read downward as the advance of a role player (i.e. an actor) from one state to another through a sequence of activities. A state is drawn as a vertical line segment; an activity is drawn as a box. Here we show a role player "carrier of the value PRESSURE" proceeding through activity 1, followed by activity 2.

<u>A.2:</u> The vertical line can be thought of as marked by a token. The position of the token shows the state of the role player. The token for pressure carries an inscription which states the value of the pressure.

<u>A.3:</u> Circles at the top and bottom of a vertical line denote the same location of a circuit. In other words the figure

```
    ┌─┐
    │P│
    └─┘
     │
    ┌─┐
    │1│
    └─┘
     │
    ┌─┐
    │2│
    └─┘
     │
    ┌─┐
    │3│
    └─┘
     │
    ( )
```
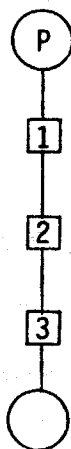
denotes a cyclic progression through activity 1, activity 2, activity 3, back to activity 1, and so on.

<u>A.4:</u> If a role P is initialized in activity 1 and terminated in activity 3 we draw the following.

```
    ┌─┐
    │1│
    └─┘
     │
     P
     │
    ┌─┐
    │2│
    └─┘
     │
    ┌─┐
    │3│
    └─┘
```

Note that the initiation of a role (shown in activity 1) requires that a physical actor be on hand to play the role.

<u>A.5:</u> In contrast to A.4, a fragment of a longer chain is drawn
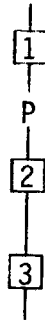
<u>A.6:</u> When several roles participate in a common activity their coincident participation is denoted by horizontal links.
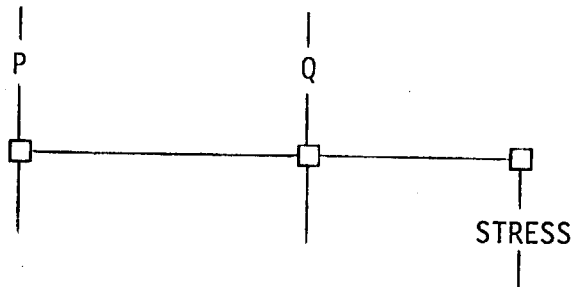
As shown, P and Q must coincidently be present at the creation of STRESS. The horizontal line of boxes converts inputs (above) to outputs (below).

<u>A.7:</u> The diagram A.6 indicates that P and Q change values as a consequence of taking part in the creation of STRESS. If we wish to indicate <u>no change</u> of value of P, we draw
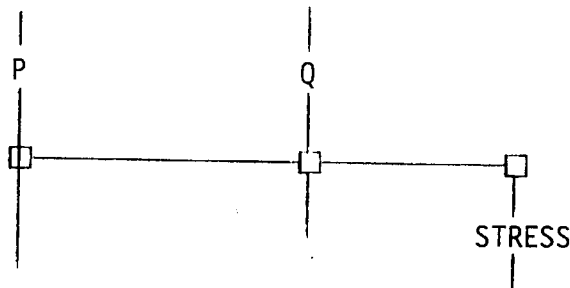
A.8: A role can branch into alternative states, shown as
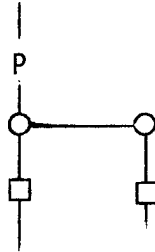
A.9: In case of a branch, the choice of path can be resolved by interaction with other value-carrying roles. Suppose that exactly one of B1 or B2 will be present, and will resolve the choice for P; then A.8 could be filled out as
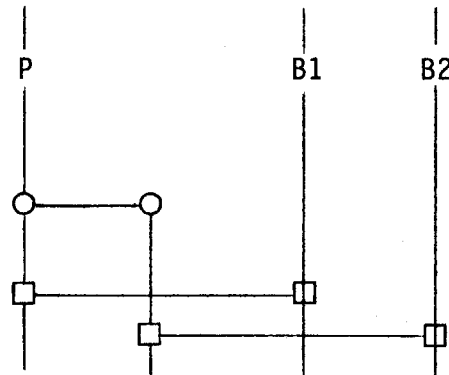
A.10: In drawing a diagram with two alternative states, such as B1 and B2 in A.9, it may be convenient to pull the two lines into one:
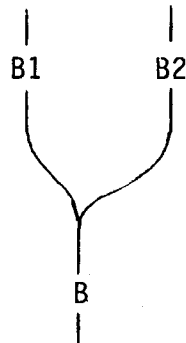
This pulling together is not an "objective" fact of the "system", but rather a matter decided by the drawer of the diagram. He decides to view the distinction formerly borne by the separation of the lines as "encoded" into an attribute of a token that travels on the joined line.

A.11: If the person who draws the diagram has <u>encoded</u> B1 and B2, as in A.10, then in drawing A.9 he would have to "<u>decode</u>" them -- i.e. to reporduce separated lines, one for each of the encoded alternatives. In this case A.9 would be drawn with a fork:
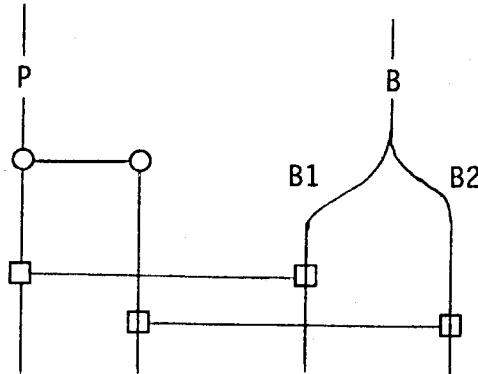
A.12: Two activities can be alternatives to the production of a single state, in which case two states of a role can <u>merge</u>.
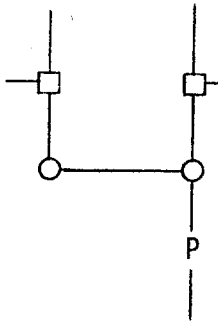
A.12 can be compared with A.9. Lines joined by branches and merges of a role form a state component of a Petri net.

A.13: For convenience of presentation one may wish to bundle several roles together and picture them as a single "cable", as in an image of cabling together of different "wires". We illustrate this by roles A, B and C which are "cabled" into a compound strand called L . In other words,

$$L = \{A, B, C\}.$$
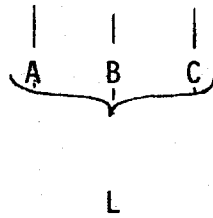
Unlike encoded alternatives (see A.10) all the roles of a bundle can be concurrently played

A.14: Unbundling corresponding to the undoing of A.13 is drawn as follows.
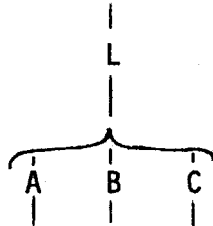
A.15: Brackets around a row indicate that the row is compressed from a more detailed diagram shown elsewhere; for example the figure
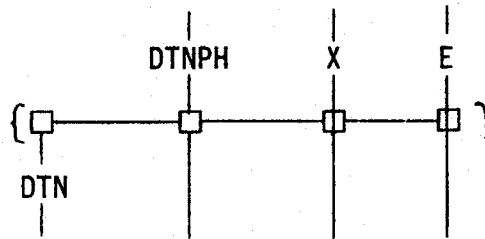
is compressed from
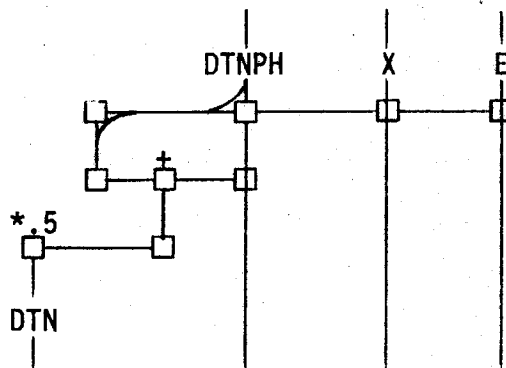
A.15.1: The outputs of a bracketed row can be produced by an internal loop, containing internal variables. TNUP is such a variable in the following diagram, where
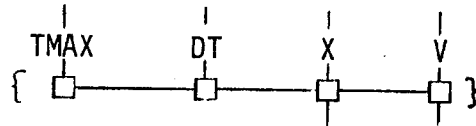
is compressed from
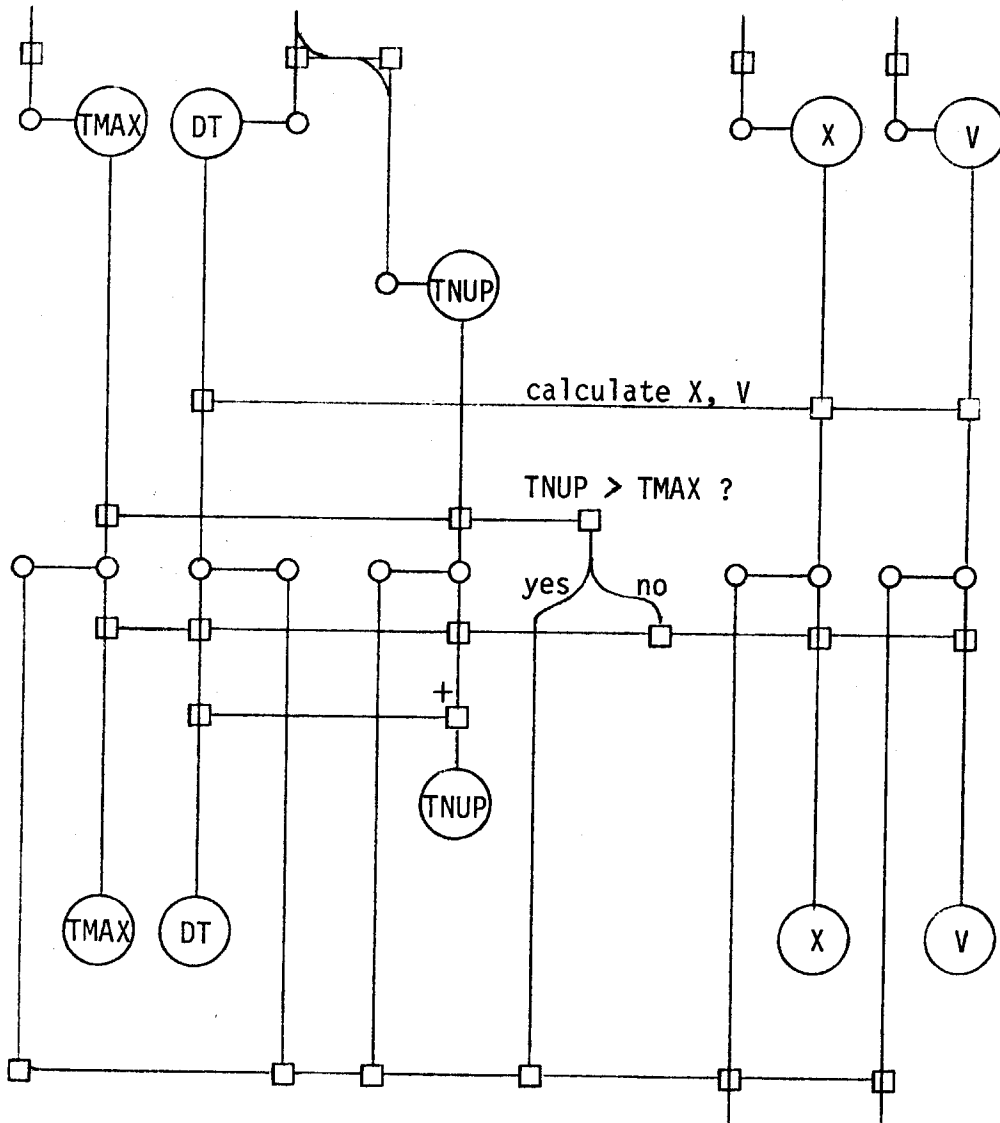
A.16: The following illustrates <u>fanout</u>.

P
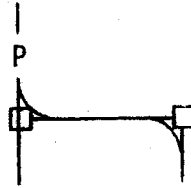
(This notation was used in A.15.1.)

A.16.1: Fanout can also be shown as follows.
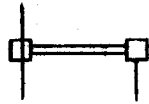
A.16.2: We link two boxes by a double bar to assert identity of output values; the following asserts that after the occurrence of the activity, B and B' carry copies of the same value; the figure does not assert anything about the relation between inputs, nor about the relation between inputs and outputs.

A

B   B'
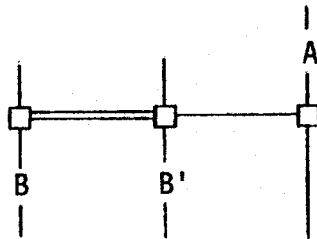
A.17: The following illustrates the saving of the value of P as OLD_P, while P is changed.

P

P   OLD_P
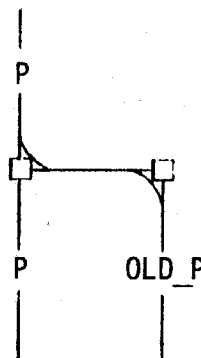
(This notation was used in A.15)

<u>A.18:</u> On occasion we indicate arithmetic operations on values, as in this picture. After the activity of the row occurs, C carries the value A+B.
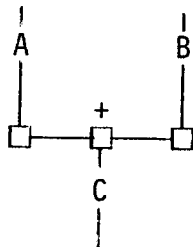
<u>A.18.1:</u> If A is a matrix, then B as the sum over the elements of A could be pictured as follows.
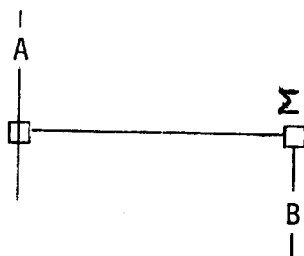
<u>A.19: Buffered communication.</u> A fragment of Figure 2 (of the main report is

.1.
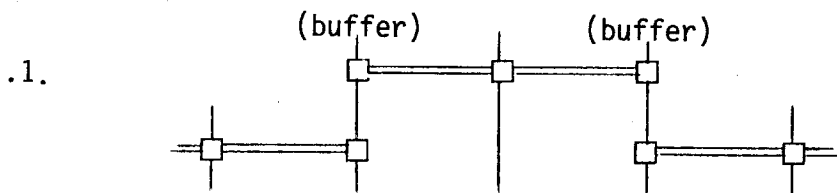
(buffer)   (buffer)

This can be expanded to

.2.
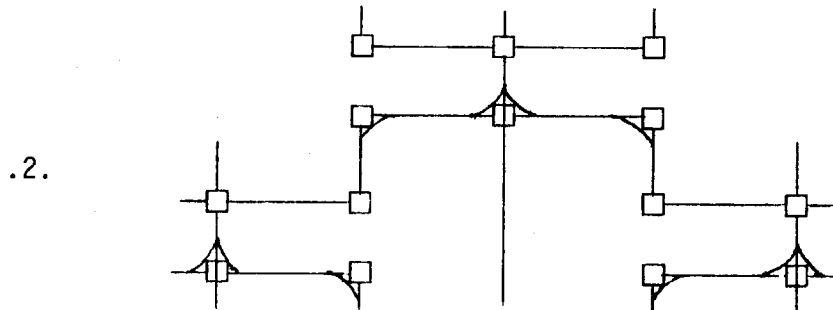
The figure .2 contains the fragment

.3

for which we introduce the abbreviated notation:

.4
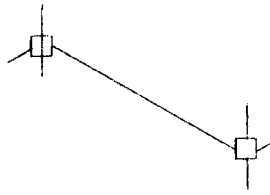
The slanted bar asserts that the lower activity consumes something produced in the upper activity, and that a buffer not explicitly shown mediates the transfer from the producing to the consuming activity. With this notation, Figure 4 of the main report is transformed into Fig. 5.

APPENDIX B

Notes on Fitting the SIMPLE Code into Role Diagrams and VAL Modules

Figure 6 of the main report somewhat schematically shows the connectivity of communication among processors, when one processor is assigned to each nodal and each zonal region of the dataflow graph. In this appendix we discuss the connections in more detail, and also discuss certain ways in which the algorithm of SIMPLE has been restated to clarify the connectivity. The objective is to help in considering hardware requirements, and to clarify aspects of the translation from FORTRAN into VAL.

B.1. Interpretation of the cycle

Fig. 6 shows phase 3 as producing new values for zone [K,L] as follows.
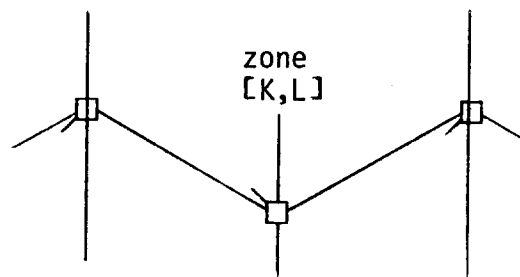
zone
[K,L]

.1: Schematic representation of production of zonal value.

The fragment .1 is a schematic picture of an activity at zone K,L that draws on values from the four neighboring (i.e. corner) nodes to feed into the production of new values for the zone. With the indexing convention defined in Fig. 1 of the main report, one sees that the fragment .1 stands for the connections shown in .2:
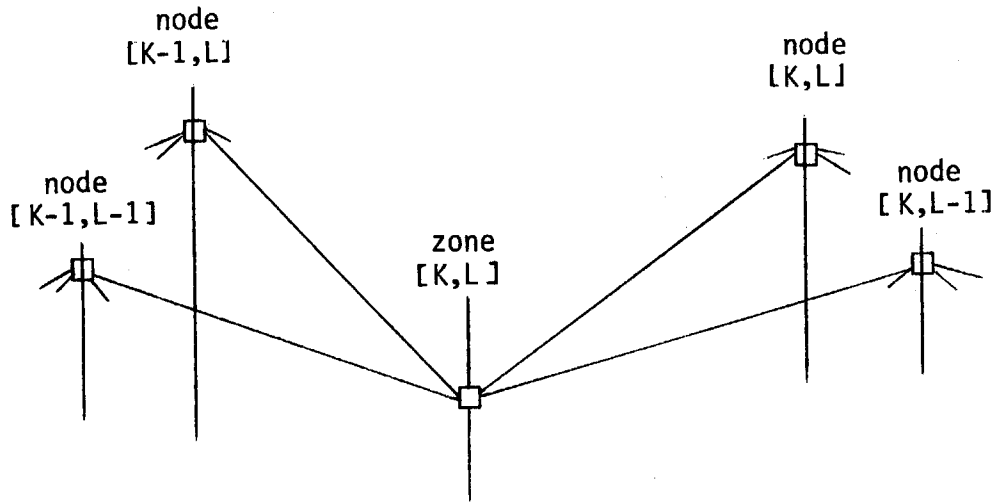
.2: Completed fragment showing all connections of nodes to a zone.

The nodal values are a vector (with R and Z components) for velocity and a vector for position at each node. The corresponding type definitions and declarations in SIMPLE_VAL are:

type vector = record[R, Z: real];

type nodal = array[array[vector]];

X, % position

V: % velocity

nodal %.

The correspondence between these names as used in SIMPLE_VAL and the names used in the FORTRAN code of SIMPLE is:

FORTRAN code	VAL code
R	X.R
Z	X.Z
U	V.R
W	V.Z .

In order to clarify the connectivity, as well as to eliminate some
unnecessary arithmetic, we introduce auxiliary variables, starting with
a kind of tensor -- GX -- that describes the diagonal dimensions of each
zone:

GX[K,L].W   GX[K,L].E
(a vector)   (a vector)
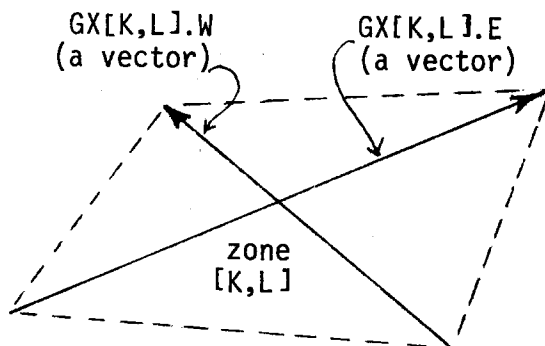
zone
[K,L]

.3: Definition of GX.

GX is, at least in spirit, a tensor; GX[K,L].W is the vector difference
between the vector position at the nortwest corner and the vector position
at the southeast corner. GX is produced for interior zones by ZONE_GEOM
in phase 3, and for boundary zones by BOUNDARY_PROJECT; in the first case
the defining relation is

.4.   type zone_tensor = array array record E, W: vector   ;

         GX: zone_tensor :=

             forall K in [LIM.KN+1, LIM.KX], L in [LIM.LN+1, LIM.LX] construct
             record[E: X[K,L]- X[K-1,L-1]; W: X[K-1,L]- X[K,L-1]] endall; %.

Note that X is a vector, so that .4 is a shorthand expression; strictly speaking
one must define a subtraction function with vector arguments. This is
easy to do, but clutters the presentation. With the understanding that
we are abbreviating, we shall apply "-", "+" and multiplication by a scalar ("*")
to vectors.   The node-to-zone communications needed to form GX are shown in

the picture .2.   The auxiliary variable GV is a zonal tensor formed from V

in exactly the same way that GX is formed from X.

Now we address phase 2 and the calculation of V.   Prior to

communicating from the zones around a given node to the node, a tensor

STRESS is calculated for each zone; this calculation for a given zone

draws only on array elements for that zone.   The computation covers boundary

zones, set up in phase 1, as well as interior zones.

.5   STRESS: zone_tensor :=

         forall K in [LIM.KN, LIM.KX+1], L in[LIM.LN, LIM.LX+1] construct

         record[E: (P[K,L]+ Q[K,L])*GX[K,L].E; %scalar * vector

               W: (P[K,L]+ Q[K,L])*GX[K,L].W] endall ; %,

where P and Q are pressure and artificial viscosity, respectively, just as

in the FORTRAN code.   In phase 1 the auxilliary variable RHOJ is produced as:

.6   RHOJ : zonal :=

         forall K in[LIM.KN, LIM.KX+1], L in [LIM.LN, LIM.LX+1] construct

         RHO[K,L]*AJ[K,L] endall; %,

where RHO and AJ are density and area jacobian, just as in the FORTRAN code.

Phase 2 of the cycle produces new values for each node, namely

V and X.   The fragment that produces values for a particular node, say

node K,L appears in phase 2 of Fig. 6 as follows.
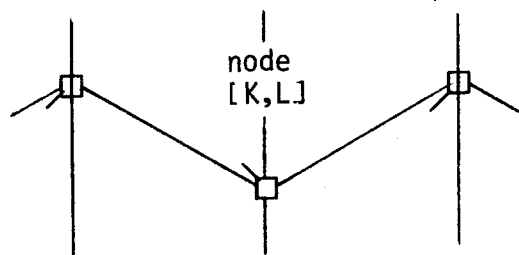
.7: Schematic representation of the production of a nodal value.

The fragment .7 is a schematic picture of an activity that draws on values
from the four zones around node [K,L] to feed into the production of new
values of X and V for the node. Thus the fragment .7 stands for
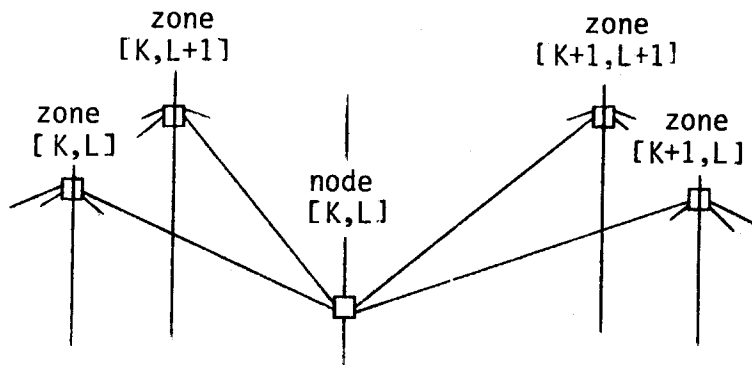
.8: Completed fragment showing all zones connected to a node

Each "cable" of values from a zone to node [K,L] must carry STRESS and
RHOJ from the zone, and at least one of these cables must bring the time
steps DTNPH and DTN as well. (DTNPH and DTN are used here as they are in
the FORTRAN code of SIMPLE (1979).) The activity of the node in .8 during
phase 2 of the cycle is to calculate an acceleration (ACC), to use this
acceleration to update velocity (V), and then to use the velocity to update
position (X). In updating velocity a time step DTN is used. Position times
interleave the times at which velocity is calculated, so that a different
time step (DTNPH) is used to update position. Continuing to use the
abbreviation of scalar operation signs for operations on vector values,
this activity can be expressed in VAL as:

V, X :=

forall K in [LIM.KN, LIM.KX+1], L in [LIM.LN, LIM.LX+1] construct

   let Y: vector := (2./(RHOJ[K,L] + RHOJ[K,L+1] + RHOJ[K+1,L] + RHOJ[K+1,L+1]))

     *(STRESS[K,L+1].E + STRESS[K,L].W - (STRESS[K+1,L+1].W + STRESS[K+1,L].E));

     ACC: vector := record[R: -Y.Z; Z: Y.R];

     V1: vector := DTN*ACC + V[K,L]

  in V1, DTNPH*V1 + X[K,L] endlet

endall

After expansion of the vector operations, this code would provide the

functions VELOCITY and POSITION of Sec. 5.2.3.

     Phase 4 involves arrays that are partly nodal and partly zonal in

character. An element of CBB is obtained as an intermediate between two

nodes of the same L-coordinate but adjoining K coordinates, and two zones

bounded by the nodal K coordinates and on either side of the L coordinate:

     In calculating heat conduction subroutine CONDUCT of the SIMPLE

FORTRAN code generates arrays CBB and DBB, per lines 1583 through 1608.

CBB and DBB draw on values from both nodes and zones, as shown:

For CBB[K,L]   For DBB[K,L]
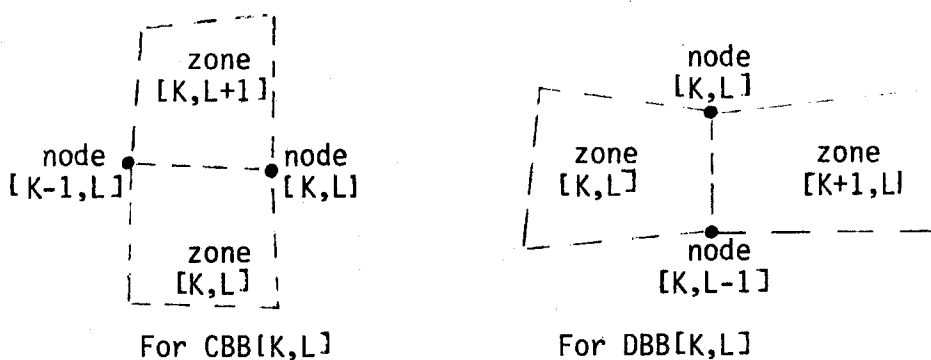
.9: Nodes and zones that supply values to the calculation of CBB[K,L]
     and DBB[K,L].

To adhere strictly to the connectivity shown in Fig. 6, one programs the calculation of CBB and DBB in two parts, one as an augmentation of ZONE_GEOM and the other as part of an augmented ENERGY_HYDRO. The augmentation consists of generating geometrical quantities as part of ZONE_GEOM, referring these to zones, as was done for GX, and then using these quantities to simplify the connectivity needed in ENERGY_HYDRO. An alternative which is displayed in SIMPLE_VAL of Sec. 5.2.3 is to accept a slightly more complex connectivity and thereby avoid the introduction of more auxilliary variables.

CBB and DBB are partly zonal and partly nodal in character, so that fitting them to either class of processors is arbitrary. Because the nodal processors are less heavily used, we have assumed that they would be used to compute CBB and DBB from zonal quantities (CC in FORTRAN). The consequent connectivity is shown in phase 4 of Fig. 6. For the Z-sweep this connectivity is shown in more detail in .10:

Node
[K,L-1]

Zone
[K,L]

Node
[K,L]

Zone
[K,L+1]

CBB

$\left\{\begin{matrix} \text{TEMP,} \\ \text{SIG} \end{matrix}\right\}$

CBB

$\left\{\begin{matrix} \text{TEMP,} \\ \text{SIG} \end{matrix}\right\}$

$\{A,B\}$

$\left\{\begin{matrix} \text{CBB*(1-A),} \\ \text{CBB*B} \end{matrix}\right\}$

CBB

$\{A,B\}$

$\left\{\begin{matrix} \text{CBB*(1-A),} \\ \text{CBB*B} \end{matrix}\right\}$

CBB

$\{A,B\}$

$\{A,B\}$

TEMP2[K,L+2]

TEMP2[K,L+1]

TEMP2[K,L]

TEMP2[K,L]

TEMP2[K,L+1]
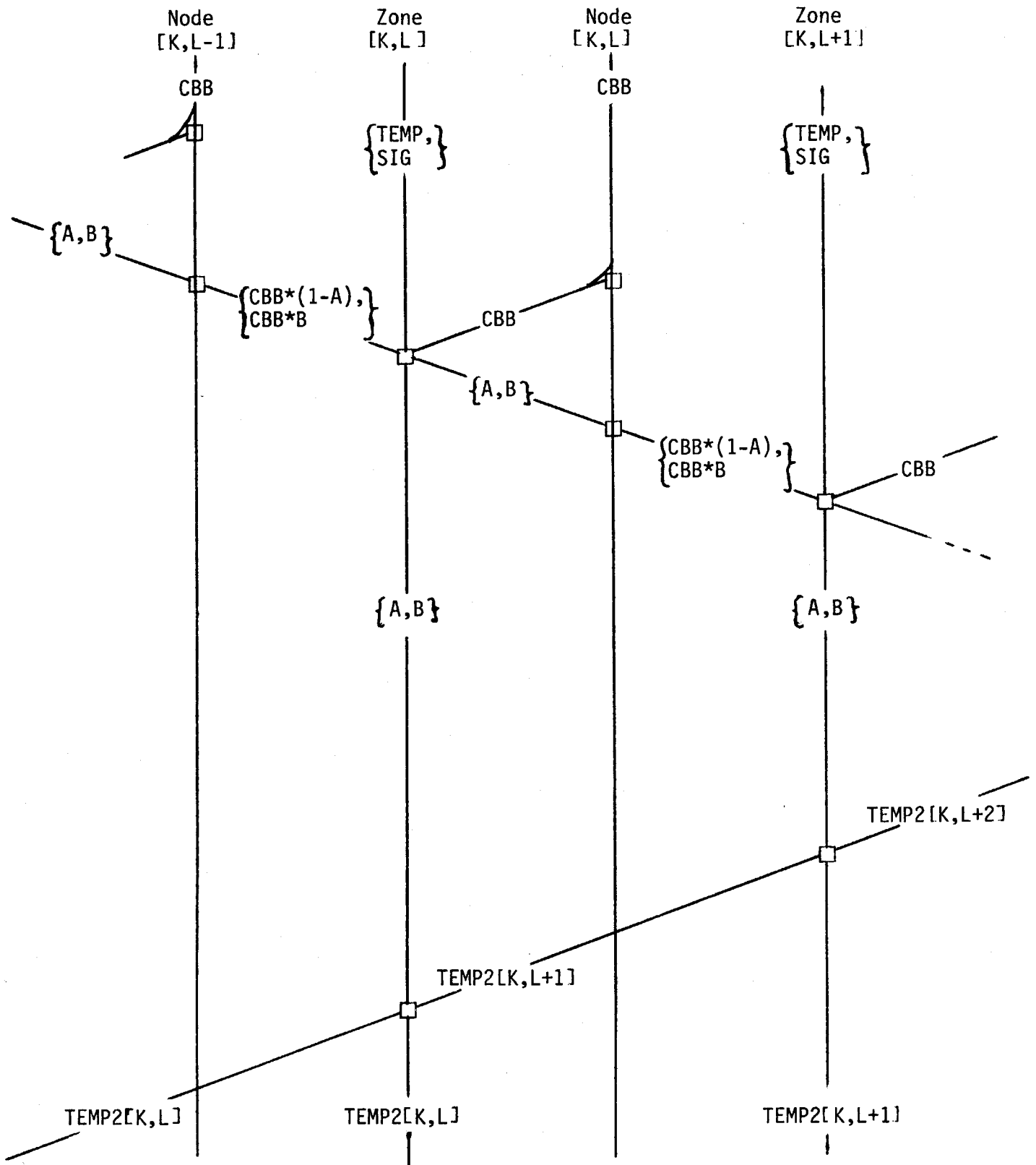
.10: Detail of Z-sweep of ENERGY_HEAT.

Appendix C: The SIMPLE code in FORTRAN

Edition of February 12, 1979 as provided by John Woodruff

```
 61       DATA TMAX/12.001/
 62       DATA VCUT/1.E-10/
 63       DATA DTEN/1.E+10/
 64       DATA DTC/1.E+10/
 65 C
 66       CALL CHANGE(2H+H)
 67       CALL ASSIGN(3,2RPH)
 68       CALL CLOCK(WHER,WHEN)
 69 C
 70 C     ZERO OUT ALL ARRAYS
 71 C
 72       L=21*33*33
 73 C
 74       DO 10 K=1,L
 75 C
 76       ARRAY(K)=0.
 77 C
 78 10    CONTINUE
 79 C
 80 C     SET UP EOS TABLES
 81 C
 82       CALL SETUP
 83 C
 84 C     SET PARAMETERS FOR TEST PROBLEM
 85 C
 86       GAM=1.4
 87       P2=6.
 88       P3=0.
 89       PB(1)=1.
 90       PB(2)=0.
 91       PB(3)=0.
 92       QB(1)=1.
 93       QB(2)=0.
 94       QB(3)=0.
 95       PBB(1)=0.
 96       PBB(2)=P2
 97       PBB(3)=P3
 98       RHO0=1.4
 99       KMN=2
100       LMN=2
101       KMX=5
102       LMX=22
103       DR=1.
104       DZ=1.
105       DTNPH=.01
106       TMAX=10.
107       DTN=DTNPH
108       DTNMH=DTNPH
109       TNUP=0.
110       E0=0.
111       U0=0.
112       P0=0.
113       W0=0.
114       NBCU=1
115       NBCR=2
116       NBCL=1
117       NBCD=1
118       C2=1.5
119       COF=C2*.25
120       C1=.5
```

```
121       C1F=.5*C1
122       GAMZ=GAM-1.
123       DTC=100.
124       P1D8=1./8.
125       HN=0.
126       WN=0.
127 C
128 C     GET INPUT PARAMETERS
129 C
130       WRITE(NTTY,4)
131     4 FORMAT(23H ENTER INPUT PARAMETERS)
132 C
133       READ(NTTY,5)KMN,KMX,LMN,LMX,EDDT,EDTIME,TMAX
134     5 FORMAT(4I2,3F5.2)
135 C
136       KMNP=KMN+1
137       LMNP=LMN+1
138 C
139       KMXP=KMX+1
140       LMXP=LMX+1
141 C
142       KMXZ=KMX-1
143       LMXZ=LMX-1
144 C
145 C     GENERATE PROBLEM
146 C
147       CALL GEN
148 C
149       IGEN=0
150 C
151 C     INITIALIZE TIMER
152 C
153       NED=1
154       I2=NECOND(I1)
155 C
156 C     START CYCLE HERE
157 C
158     1 CONTINUE
159 C
160       DTC2=1.E+12
161       SKE=0.
162       ENC=0.
163       ENH=0.
164       DTEN=1.E+12
165 C**********************************************************
166 C*                                                      *
167 C*    GEOMETRY CALCULATION FOR BOUNDARY ZONES           *
168 C*                                                      *
169 C**********************************************************
170 C
171 C     SET UP BOTTOM SIDE BOUNDARY ZONES
172 C
173       P(K,L+1)
174       O(K,L)      1(K+1,L)
175       R(K,L-1)
176 C
177       L=LMN
178 C
179       RO=R(KMN,L)
180
```

```
181          ZO=Z(KMN,L)
182   C
183   C     DO 200 K=KMN,KMXZ
184   C
185          R1=R(K+1,L)
186          Z1=Z(K+1,L)
187   C
188          RP=R(K,L+1)
189          ZP=Z(K,L+1)
190   C
191   C     CALL PROJCT
192   C
193          R(K,L-1)=RR
194          Z(K,L-1)=ZZ
195   C
196          RO=R1
197          ZO=Z1
198   C
199   C 200 CONTINUE
200   C
201   C     SET UP BOTTOM RIGHT CORNER
202   C
203   C     1(K-1,L)     P(K,L+1)
204   C                  O(K,L)
205   C                  R(K,L-1)
206   C
207          K=KMX
208          L=LMN
209   C
210          RO=R(K,L)
211          ZO=Z(K,L)
212   C
213          R1=R(K-1,L)
214          Z1=Z(K-1,L)
215   C
216          RP=R(K,L+1)
217          ZP=Z(K,L+1)
218   C
219   C     CALL PROJCT
220   C
221          R(K,L-1)=RR
222          Z(K,L-1)=ZZ
223   C
224   C     SET UP TOP SIDE BOUNDARY ZONES
225   C
226          R(K,L+1)
227          O(K,L)     1(K+1,L)
228          P(K,L-1)
229   C
230          L=LMX
231          RO=R(KMN,L)
232          ZO=Z(KMN,L)
233   C
234   C     DO 204 K=KMN,KMXZ
235   C
236          R1=R(K+1,L)
237          Z1=Z(K+1,L)
238   C
239          RP=R(K,L-1)
240          ZP=Z(K,L-1)
```

```
241  C
242  C     CALL PROJCT
243  C
244        R(K,L+1)=RR
245        Z(K,L+1)=ZZ
246  C
247        RO=R1
248        ZO=Z1
249  C
250  204 CONTINUE
251  C
252  C
253  C     SET UP TOP RIGHT CORNER
254  C
255  C     1(K-1,L)   R(K,L+1)
256  C                O(K,L)
257  C                P(K,L-1)
258  C
259        K=KMX
260        L=LMX
261  C
262        RO=R(K,L)
263        ZO=Z(K,L)
264  C
265        R1=R(K-1,L)
266        Z1=Z(K-1,L)
267  C
268        RP=R(K,L-1)
269        ZP=Z(K,L-1)
270  C
271  C     CALL PROJCT
272  C
273        R(K,L+1)=RR
274        Z(K,L+1)=ZZ
275  C
276  C
277  C     SET UP LEFT SIDE BOUNDARY ZONES
278  C
279  C     R(K-1,L)   1(K,L+1)   P(K+1,L)
280  C                O(K,L)
281  C
282        K=KMN
283        RO=R(K,LMN)
284        ZO=Z(K,LMN)
285  C
286        DO 207 L=LMN,LMXZ
287  C
288        R1=R(K,L+1)
289        Z1=Z(K,L+1)
290  C
291        RP=R(K+1,L)
292        ZP=Z(K+1,L)
293  C
294  C     CALL PROJCT
295  C
296        R(K-1,L)=RP
297        Z(K-1,L)=ZZ
298  C
299        RO=R1
300        ZO=Z1
     207 CONTINUE
```

```
301 C
302 C     SET UP TOP LEFT CORNER
303 C
304 C     R(K-1,L)   O(K,L)     P(K+1,L)
305 C                1(K,L-1)
306 C
307       K=KMN
308       L=LMX
309 C
310       RO=R(K,L)
311       ZO=Z(K,L)
312 C
313       R1=R(K,L-1)
314       Z1=Z(K,L-1)
315 C
316       RP=R(K+1,L)
317       ZP=Z(K+1,L)
318 C
319       CALL PROJCT
320 C
321       R(K-1,L)=RR
322       Z(K-1,L)=ZZ
323 C
324 C
325 C     SET UP RIGHT SIDE BOUNDARY ZONES
326 C
327 C     P(K-1,L)   1(K,L+1)   R(K+1,L)
328 C                O(K,L)
329 C
330       K=KMX
331       RO=R(K,LMN)
332       ZO=Z(K,LMN)
333 C
334       DO 210 L=LMN,LMXZ
335 C
336       R1=R(K,L+1)
337       Z1=Z(K,L+1)
338 C
339       RP=R(K-1,L)
340       ZP=Z(K-1,L)
341 C
342       CALL PROJCT
343 C
344       R(K+1,L)=RR
345       Z(K+1,L)=ZZ
346 C
347       RO=R1
348       ZO=Z1
349 210   CONTINUE
350 C
351 C     SET UP TOP RIGHT CORNER
352 C
353 C     P(K-1,L)   O(K,L)     R(K+1,L)
354 C                1(K,L-1)
355 C
356       K=KMX
357       L=LMX
358 C
359       RO=R(K,L)
360       ZO=Z(K,L)
```

```
361   C
362         R1=R(K,L-1)
363         Z1=Z(K,L-1)
364   C
365         RP=R(K-1,L)
366         ZP=Z(K-1,L)
367   C
368   C     CALL PROJCT
369   C
370         R(K+1,L)=RR
371         Z(K+1,L)=ZZ
372   C
373   C
374   C     SET UP TOP RIGHT CORNER
375   C
376   C     P(K-1,L+1)  1(K,L+1)  R(K+1,L+1)
377   C                 O(K,L)
378   C
379         K=KMX
380         L=LMX
381   C
382         RO=R(K,L)
383         ZO=Z(K,L)
384   C
385         R1=R(K,L+1)
386         Z1=Z(K,L+1)
387   C
388         RP=R(K-1,L+1)
389         ZP=Z(K-1,L+1)
390   C
391   C     CALL PROJCT
392   C
393         R(K+1,L+1)=RR
394         Z(K+1,L+1)=ZZ
395   C
396   C
397   C     SET UP BOTTOM LEFT CORNER
398   C
399   C     R(K-1,L-1)  O(K,L)  1(K,L-1)  P(K+1,L-1)
400   C
401         K=KMN
402         L=LMN
403   C
404         RO=R(K,L)
405         ZO=Z(K,L)
406   C
407         R1=R(K,L-1)
408         Z1=Z(K,L-1)
409   C
410         RP=R(K+1,L-1)
411         ZP=Z(K+1,L-1)
412   C
413   C     CALL PROJCT
414   C
415         R(K-1,L-1)=RR
416         Z(K-1,L-1)=ZZ
417   C
418   C     SET UP BOTTOM RIGHT CORNER
419   C
420   C     O(K,L)  P(K+1,L+1)  1(K+1,L)
```

```
421   C          R(K+1,L-1)
422   C
423             K=KMX
424             L=LMN
425   C
426             RO=R(K,L)
427             ZO=Z(K,L)
428   C
429             R1=R(K+1,L)
430             Z1=Z(K+1,L)
431   C
432             RP=R(K+1,L+1)
433             ZP=Z(K+1,L+1)
434   C
435             CALL PROJCT
436   C
437             R(K+1,L-1)=RR
438             Z(K+1,L-1)=ZZ
439   C
440   C     SET UP TOP LEFT CORNER
441   C
442             R(K-1,L+1)
443             1(K-1,L)      O(K,L)
444             P(K-1,L-1)
445   C
446             L=LMX
447             K=KMN
448   C
449             RO=R(K,L)
450             ZO=Z(K,L)
451   C
452             R1=R(K-1,L)
453             Z1=Z(K-1,L)
454   C
455             RP=R(K-1,L-1)
456             ZP=Z(K-1,L-1)
457   C
458             CALL PROJCT
459   C
460             R(K-1,L+1)=RR
461             Z(K-1,L+1)=ZZ
462   C************************************************
463   C**
464   C**   SET UP BOUNDARY ZONE ATTRIBUTES   *
465   C**                                     *
466   C************************************************
467   C
468   C     SET UP BOTTOM SIDE BOUNDARY ZONES
469   C
470   C
471             (K,L) = (K,L+1)
472   C
473             L=LMN
474   C
475             DO 255 K=KMNP,KMX
476   C
477             RHO(K,L)=RHO(K,L+1)
478             AJ(K,L)=AJ(K,L+1)
479             IP=NBC(K-1,L)
480             Q(K,L)=QB(IP)*Q(K,L+1)
```

```
481  C     P(K,L)=PBB(IP)+PB(IP)*P(K,L+1)
482  C  255 CONTINUE
483  C
484  C     SET UP RIGHT SIDE BOUNDARY ZONES
485  C
486  C     (K+1,L) = (K,L)
487  C
488  C     K=KMX
489  C
490  C     DO 265 L=LMNP,LMX
491  C
492  C     RHO(K+1,L)=RHO(K,L)
493  C     AJ(K+1,L)=AJ(K,L)
494  C     IP=NBC(K,L)
495  C     Q(K+1,L)=QB(IP)*Q(K,L)
496  C     P(K+1,L)=PBB(IP)+PB(IP)*P(K,L)
497  C
498  C  265 CONTINUE
499  C
500  C     SET UP TOP SIDE BOUNDARY ZONES
501  C
502  C     (K,L+1) = (K,L)
503  C
504  C     L=LMX
505  C
506  C     DO 275 K=KMNP,KMX
507  C
508  C     RHO(K,L+1)=RHO(K,L)
509  C     AJ(K,L+1)=AJ(K,L)
510  C     IP=NBC(K-1,L)
511  C     Q(K,L+1)=QB(IP)*Q(K,L)
512  C     P(K,L+1)=PBB(IP)+PB(IP)*P(K,L)
513  C
514  C  275 CONTINUE
515  C
516  C     SET UP LEFT SIDE BOUNDARY ZONES
517  C
518  C     (K,L) = (K+1,L)
519  C
520  C     K=KMN
521  C
522  C     DO 285 L=LMNP,LMX
523  C
524  C     RHO(K,L)=RHO(K+1,L)
525  C     AJ(K,L)=AJ(K+1,L)
526  C     IP=NBC(K,L-1)
527  C     Q(K,L)=QB(IP)*Q(K+1,L)
528  C     P(K,L)=PBB(IP)+PB(IP)*P(K+1,L)
529  C
530  C  285 CONTINUE
531  C
532  C     SET UP BOTTOM LEFT CORNER
533  C
534  C     P(KMN,LMN)=P(KMNP,LMNP)
535  C     Q(KMN,LMN)=Q(KMNP,LMNP)
536  C     RHO(KMN,LMN)=RHO(KMNP,LMNP)
537  C     AJ(KMN,LMN)=AJ(KMNP,LMNP)
538  C
539  C     SET UP BOTTOM RIGHT CORNER
540  C
```

```
541   C
542         P(KMXP,LMN)=P(KMXP,LMN+1)
543         Q(KMXP,LMN)=Q(KMXP,LMN+1)
544         RHO(KMXP,LMN)=RHO(KMXP,LMN+1)
545         AJ(KMXP,LMN)=AJ(KMXP,LMN+1)
546   C
547   C     SET UP TOP RIGHT CORNER
548   C
549         P(KMXP,LMXP)=P(KMXP,LMX)
550         Q(KMXP,LMXP)=Q(KMXP,LMX)
551         RHO(KMXP,LMXP)=RHO(KMXP,LMX)
552         AJ(KMXP,LMXP)=AJ(KMXP,LMX)
553   C
554   C     SET UP TOP LEFT CORNER
555   C
556         P(KMN,LMXP)=P(KMNP,LMXP)
557         Q(KMN,LMXP)=Q(KMNP,LMXP)
558         RHO(KMN,LMXP)=RHO(KMNP,LMXP)
559         AJ(KMN,LMXP)=AJ(KMNP,LMXP)
560   C
561   C     GET BOUNDARY CONDITION COMPUTE TIME
562   C
563         I2=NECOND(I1)
564         NBT(NED)=NBT(NED)+I2
565   C
566   C     DEBUG EDIT
567   C
568         IF(DEBUG.EQ.0.) GO TO 442
569         IGEN=1
570   C
571         WRITE(NO,441)
572   441   FORMAT(9H DEBUG   1)
573   C
574         CALL EDIT
575   C
576   442   CONTINUE
577   C
578         DO 450 L=LMN,LMX
579         DO 445 K=KMN,KMX
580   COMPUTE ACCELERATION
581         AU=(P(K,L)+Q(K,L)) * (Z(K,L-1)-Z(K-1,L)) +
582        X   (P(K+1,L)+Q(K+1,L))*(Z(K+1,L)-Z(K,L-1)) +
583        X   (P(K+1,L+1)+Q(K+1,L+1))*(Z(K,L+1)-Z(K+1,L)) +
584        X   (P(K,L+1)+Q(K,L+1))*(Z(K-1,L)-Z(K,L+1)) +
585         AW=(P(K,L)+Q(K,L)) * (R(K,L-1)-R(K-1,L)) +
586        X   (P(K+1,L)+Q(K+1,L)) * (R(K+1,L)-R(K,L-1)) +
587        X   (P(K+1,L+1)+Q(K+1,L+1)) * (R(K,L+1)-R(K+1,L)) +
588        X   (P(K,L+1)+Q(K,L+1)) * (R(K-1,L)-R(K,L+1)) +
589         AUW=RHO(K,L)*AJ(K,L)+RHO(K+1,L)*AJ(K+1,L)+RHO(K+1,L+1)*AJ(K+1,L+1) +
590        X   +RHO(K,L+1)*AJ(K,L+1)+RHO(K+1,L+1)*AJ(K+1,L+1)+RHO(K,L+1)*AJ(K,L+1)
591         AUW=2./AUW
592         AU=-AU*AUW
593         AW=AW*AUW
594         U(K,L)=U(K,L)+DTN*AU
595   C     ADVANCE VELOCITIES TO N+1/2 FROM N-1/2
596         W(K,L)=W(K,L)+DTN*AW
597   C     POSITION (N+1)
598         IF(ABS(U(K,L)).LE.VCUT)U(K,L)=0.
599         IF(ABS(W(K,L)).LE.VCUT)W(K,L)=0.
600         A(K,L)=U(K,L)**2+W(K,L)**2
```

```
601   445 CONTINUE
602   450 CONTINUE
603   C
604       IF(NOHYD.EQ.1) GO TO 455
605   C   NOHYD=1 TO SKIP HYDRO
606       DO 452 L=LMN,LMX
607       DO 451 K=KMN,KMX
608       R(K,L)=R(K,L)+DTNPH*U(K,L)
609       Z(K,L)=Z(K,L)+DTNPH*W(K,L)
610   451 CONTINUE
611   452 CONTINUE
612   C   ACCELERATION VELOCITY AND
613   C   CO-ORDINATES DONE        END OF NECOND PASS
614   C
615   C   TEMPRY *********      BEGIN LOOP 3
616   C
617   C   DEBUG EDIT
618   C
619       IF(DEBUG.EQ.0.) GO TO 455
620       IGEN=1
621   C
622       WRITE(NO,456)
623   456 FORMAT(9H DEBUG    2)
624   C
625       CALL EDIT
626   C
627   455 CONTINUE
628   C
629       CALL HWORK
630   C
631   C   COMPUTE HYDRO WORK ON THE BOUNDARY
632       DO 490 L=LMNP,LMX
633       DO 485 K=KMNP,KMX
634       AJ1=R(K,L)*(Z(K-1,L)-Z(K,L-1)) +
635       X   R(K-1,L)*(Z(K,L-1)-Z(K,L)) +
636       X   R(K,L-1)*(Z(K,L)-Z(K-1,L))
637       AJ3=R(K-1,L-1)*(Z(K,L-1)-Z(K-1,L)) +
638       X   R(K,L-1)*(Z(K-1,L)-Z(K-1,L-1)) +
639       X   R(K-1,L)*(Z(K-1,L-1)-Z(K,L-1))
640   C
641   C   JACOBIAN AREA IN (R,Z) PLANE
642   C
643       SN=S(K,L)
644       AJ(K,L)=P1D2*(AJ1+AJ3)
645       S(K,L)=P1D6*((R(K-1,L)+R(K-1,L)+R(K,L-1))*AJ1 +
646       X   (R(K-1,L)+R(K-1,L-1)+R(K,L-1))*AJ3 )
647   C
648   C   S=VOLUME/2X  (CM**3/RADIAN)
649       VN=1./RHO(K,L)
650   C   VN=SPECIFIC VOLUME AT (N)
651   C   VNP=SPECIFIC VOLUME AT (N+1)
652       RHO(K,L)=RHO(K,L)*SN/S(K,L)
653       DUM(K,L)=RHO(K,L)*S(K,L)
654   C   DUM=MASS
655   C   DENSITY AT N+1
656       VNP=1./RHO(K,L)
657       DELV=VNP-VN
658   C   COMPUTE ARTIFICIAL VISCOSITY
659       DRK=R(K,L)-R(K-1,L-1)+R(K,L-1)-R(K-1,L)
660       DRL=R(K,L)-R(K-1,L-1)+R(K,L-1)-R(K-1,L)
```

```fortran
661          DZK=Z(K,L)-Z(K-1,L-1)+Z(K,L-1)-Z(K-1,L)
662          DZL=Z(K,L)-Z(K-1,L-1)+Z(K-1,L)-Z(K,L-1)
663          DUK=U(K,L)-U(K-1,L-1)+U(K,L-1)-U(K-1,L)
664          DUL=U(K,L)-U(K-1,L-1)+U(K-1,L)-U(K,L-1)
665          DWK=W(K,L)-W(K-1,L-1)+W(K,L-1)-W(K-1,L)
666          DWL=W(K,L)-W(K-1,L-1)+W(K-1,L)-W(K,L-1)
667    C
668          DRK=2DR/DK
669          DRL=2DR/DL
670    C
671          W1= DRK*DWL-DZK*DUL
672          W2= DUK*DZL-DWK*DRL
673          Q(K,L)=0.
674          W3=0.
675          W4=0.
676          IF(W1.LT.0.)W3=W1**2/(DRK**2+DZK**2)
677          IF(W2.LT.0.)W4=W2**2/(DRL**2+DZL**2)
678          IF((W3+W4).EQ.0.) GO TO 465
679          CA=SQRT(GAM*P(K,L)/RHO(K,L))
680    C DON'T COMPUTE Q IF ZONE IS NOT BEING COMPRESSED
681          Q(K,L)=COF*RHO(K,L)*(W3+W4) + C1F*CA*RHO(K,L)*SQRT(W3+W4)
682    C C1F= C1*.5      COF=.25*CO**2      CA=SOUND SPEED
683    C
684          IF(CA.EQ.0.) GO TO 465
685          TSO=(AJ(K,L)**2)/(CA*CA*(DRK**2+DRL**2+DZK**2+DZL**2))
686          IF(DTC2.LE.TSO) GO TO 462
687    C HAVE A NEW MINIMUM DELTA T
688          DTC2=TSO
689          KC=K
690          LC=L
691    462   CONTINUE
692    465   CONTINUE
693          EPS=E(K,L)-(P(K,L)+Q(K,L))*DELV
694          E(N+1)
695    C
696          RARG1=RHO(K,L)
697          CALL TEMPCAL
698          TARG1=TEMPS
699          CALL IES1
700          PNP=FUNC1
701    C GAMMA-LAW EOS    GAMZ=GAM-1
702          E(K,L)=E(K,L)-(.5*(PNP+P(K,L))+Q(K,L))*DELV
703          E(K,L)=AMAX1(E(K,L),1.E-30)
704          EPS=E(K,L)
705          CALL TEMPCAL
706    C GET TEMPERATURE  AS FUNCTION OF E.RHO
707          TARG1=AMAX1(TEMPS,TFLR)
708          TEMP(K,L)=TARG1
709          CALL IES1
710    C GET PRESSURE
711          P(K,L)=FUNC1
712          E(N+1)
713          P(N+1)
714    C
715    C KINETIC ENERGY FOR THE ZONE
716          SKE=SKE+P1D8*DUM(K,L)*(A(K,L)+A(K-1,L)+A(K,L-1)+A(K-1,L-1))
717    485   CONTINUE
718    490   CONTINUE
719    C***********************************END OF LOOP 3
720    C
```

```
721   C   DEBUG EDIT
722   C
723         IF(DEBUG.EQ.0.) GO TO 495
724         IGEN=1
725   C
726         WRITE(NO,493)
727   493   FORMAT(9H DEBUG   3)
728   C
729         CALL EDIT
730   C
731   495   CONTINUE
732   C
733         IGEN=0
734         I2=NECOND(I1)
735         NXT(NED)=NXT(NED)+I2
736   C
737         CALL CONDUCT      DO HEAT CONDUCTION
738   C
739   C
740   C   ADVANCE CYCLE COUNTER
741         NYCL=NYCL+1
742         DTNMH=DTNPH
743         DTC=SQRT(DTC2)
744         DTNPH=DTC
745   C   LIMIT MAGNITUDE OF DT
746         DTNPH=AMIN1(DTNPH,DTEN,DTMAX)
747         DTN=.5*(DTNPH+DTNMH)
748         TNUP=TNUP+DTNPH
749         IF(DTNPH.GE.DTMIN) GO TO 602
750   C********** DT IS BELOW ALLOWED MINIMUM **********
751         WRITE(NO,601)NYCL,TNUP,DTNPH,DTMIN
752         WRITE(NO,601)NYCL,TNUP,DTNPH,DTMIN
753   601   FORMAT(12H DTSTOP NYCL,I6,3H T ,E12.4,4H DT ,E12.4)
754         GO TO 999
755   602   CONTINUE
756         TE=SKE+ENH
757         CN=TE-HN-WN
758         IF(NYCL.EQ.1)   CNOLD=CN
759         CNN=CN-CNOLD
760         ENCG=ENCG+CNN
761         CNOLD=CN
762         IF(MOD(NYCL,NCP).NE.0) GO TO 603
763         WRITE(NO,706)
764   706   FORMAT(6H CYCLE, 4X, 5HTIME , 7X, 2HDT, 10X, 3HDTC, 5X, 8H     KC     LC,
765        X     4X, 3HDTE, 5X, 8H KEN LEN)
766         WRITE(NO,707)NYCL,TNUP,DTNPH,DTC,KC,LC,DTEN,KEN,LEN
767   707   FORMAT(I6,3E12.4,2I4,E12.4,2I4)
768         Z2=ABS(ENC-(ENH+HN))/ENC
769         WRITE(NO,708)
770   708   FORMAT(4X, 4HETOT, 8X, 4HIE , 8X, 4HKE  , 8X, 4HHN     , 8X, 4HWN
771        X     8X, 5HECONS, 7X, 5HCN(N), 7X, 4HECNG)
772         WRITE(NO,709)TE,ENC,SKE,HN,WN,Z2,CNN,ENCG
773   709   FORMAT(8E12.4)
774   603   CONTINUE
775         I2=NECOND(I1)
776         NCT(NED)=NCT(NED)+I2
777   C   RUN TIME FOR PHYSICS
778         IF(TNUP.LT.EDTIME) GO TO 605
779   C   TIME TO EDIT
780         CALL EDIT
```

```
781        WRITE(NO,604)NYCL,TNUP,DTNPH,RMAX,KR,LR
782    604 FORMAT(12H EDIT NYCL= ,I6,2E12.4,E14.5,2I4)
783  C MESSAGE TO TTY
784  C
785  C ADVANCE EDITIME=EDITIME+EDDT
786    605 CONTINUE
787        IF(TNUP.LT.TMAX) GO TO 610
788  C******* PROBLEM HAS REACHED TMAX***********
789        WRITE(NO,607)NYCL,TNUP,TMAX
790        WRITE(NO,607)NYCL,TNUP,TMAX
791    607 FORMAT(12H STOP TMAX  ,I6,2E12.4)
792        GO TO 999
793    610 CONTINUE
794        GO TO 1
795    999 CONTINUE
796  C PROBLEM COMPLETED GET OFF
797  C     CALL PLOTE
798        WRITE(NO,616)(NBT(K),NCT(K),NET(K),NPT(K),NXT(K),K=1,NED)
799    616 FORMAT(5(1X,I10))
800        I1=0
801        DO 618 K=1,NED
802        I1=I1+NBT(K)+NCT(K)+NET(K)+NPT(K)+NXT(K)
803    618 CONTINUE
804  C
805  C
806        IF(I1.EQ.0) I1=1
807        DO 619 K=2,NED
808        NBT(1)=NBT(1)+NBT(K)
809        NCT(1)=NCT(1)+NCT(K)
810        NET(1)=NET(1)+NET(K)
811        NPT(1)=NPT(1)+NPT(K)
812        NXT(1)=NXT(1)+NXT(K)
813    619 CONTINUE
814  C
815        AES(1)=(NBT(1)*100)/I1
816        AES(2)=(NCT(1)*100)/I1
817        AES(3)=(NET(1)*100)/I1
818        AES(4)=(NPT(1)*100)/I1
819        AES(5)=(NXT(1)*100)/I1
820        WRITE(NO,620)I1,(AES(I2),I2=1,5)
821    620 FORMAT(I10,5E12.4)
822  C
823        RETURN
824        END
```

```
825         SUBROUTINE GEN
826   C
827   C
828   C     THIS SUBROUTINE GENERATES THE INITIAL PROBLEM TO BE RUN
829
830         COMMON /KLS/ K,L,DEBUG,VERSION,WHER,WHEN,P1D6,PIE,IGEN,P1D2
831        X ,DTC,KC,LC,DTEN,KEN,LEN,SKE,HN,SIEL,CNN,ENC,ENH,ENCG,WN
832        X ,NCP
833
834   C
835         COMMON /PROGG/ RO,ZO,R1,Z1,RP,ZP,RR,ZZ
836   C
837         COMMON /COMN/  R(33,33),Z(33,33),U(33,33),RHO(33,33),Q(33,33)
838        X               E(33,33),P(33,33),AJ(33,33),S(33,33),NBC(33,33)
839        X ,     W(33,33),TEMP(33,33)
840        X , A(33,33),B(33,33),CC(33,33),DUM(33,33),CBB(33,33)
841        X , DBB(33,33),CAP(33,33),SIG(33,33),TS(33,33)
842   C
843         COMMON /PARAM/ NYCL,TNUP,DTNUP,DTN,DTNPH,DTNMH,EDTIME,EDDT
844        X  ,GAM,GAMZ,COF,C1F,C1,TMAX,DTMAX,DTMIN,TFLR,NOHYD
845        X  ,C2,P2,P3,NO,NTTY,NED
846   C
847         COMMON /KLSPACE/ KMN,LMN,KMX,LMX,KMXZ,LMXZ,KMNP,LMNP,KMXP,LMXP
848   C
849         COMMON /GENCOM/ RHOO,EO,UO,PO,WO,DR,DZ,NBCU,NBCD,NBCL,NBCR
850        X ,PB(3),PBB(3),QB(3)
851   C
852         COMMON /MINMAX/ XMIN,XMAX,YMIN,YMAX,PMIN,PMAX,QMIN,QMAX
853        X,RMIN,RMAX,KQ,LQ,KR,LR,KP,LP
854        X ,XMINX,XMAXX,YMINX,YMAXX
855   C
856   C     IGEN NOT EQUAL O WILL CAUSE THE EDIT ROUTINE TO PRINT ALL THE VARIABL
857   C
858         DATA IGEN/1/
859   C
860   C*****************************
861   C**                        **
862   C**   GENERATE NBC ARRAY    **
863   C**                        **
864   C*****************************
865   C
866   C     SET BOTTOM AND TOP BOUNDARY CONDITIONS
867   C
868         DO 52 K=KMN,KMX
869   C
870         NBC(K,LMN)=NBCD
871         NBC(K,LMX)=NBCU
872   C
873   52    CONTINUE
874   C
875   C     SET LEFT AND RIGHT BOUNDARY CONDITIONS
876   C
877         DO 54 L=LMN,LMX
878   C
879         NBC(KMN,L)=NBCL
880         NBC(KMX,L)=NBCR
881   C
882   54    CONTINUE
883   C
884   C******************************************************************
```

```
885  C*
886  C*        GENERATE COORDINATES AND VELOCITIES        *
887  C*                                                   *
888  C******************************************************
889  C
890  C     INITIALIZE THE MINIMUM AND MAXIMUM VALUES OF R AND Z
891  C
892           XMINX=1.E+6
893           XMAXX=-1.E+6
894  C
895           YMINX=1.E+6
896           YMAXX=-1.E+6
897  C
898           RP=LMX-LMN
899           ZP=KMX-KMN
900  C
901           DO 58 K=KMN,KMX
902  C
903           Z1=10+K-KMN
904  C
905           DO 57 L=LMN,LMX
906  C
907  C     COMPUTE THE COORDINATES R AND Z
908  C
909           RR=L-2
910           ZZ=(-.5+RR/RP)*PIE
911  C
912           R(K,L)=Z1*COS(ZZ)
913           Z(K,L)=Z1*SIN(ZZ)
914  C
915  C     FIND THE MINIMUM AND MAXIMUM VALUES OF R AND Z
916  C
917           XMINX=AMIN1(XMINX,R(K,L))
918           XMAXX=AMAX1(XMAXX,R(K,L))
919  C
920           YMINX=AMIN1(YMINX,Z(K,L))
921           YMAXX=AMAX1(YMAXX,Z(K,L))
922  C
923  57       CONTINUE
924  C
925  58       CONTINUE
926  C******************************************************
927  C*                                                   *
928  C*     GENERATE ZONE QUANTITIES RHO, P, E AND COMPUTE AREA  *
929  C*                                                   *
930  C******************************************************
931  C
932           P1D6=1./6.
933  C
934           DO 65 L=LMNP,LMX
935  C
936           DO 63 K=KMNP,KMX
937  C
938           RHO(K,L)=RHOO
939           P(K,L)=PO
940           E(K,L)=EO
941  C
942  C     COMPUTE JACOBIAN
943  C
944  C
```

PAGE   17

```
945        AJ1=R(K,L)*(Z(K-1,L)-Z(K,L-1))+R(K-1,L)*(Z(K,L-1)-Z(K,L))
946      X   +R(K,L-1)*(Z(K,L)-Z(K-1,L))
947  C
948        AJ3=R(K-1,L)*(Z(K-1,L-1)-Z(K,L-1))+R(K-1,L-1)*(Z(K,L-1)-Z(K-1,L))
949      X   +R(K,L-1)*(Z(K-1,L)-Z(K-1,L-1))
950  C
951        AJ(K,L)=P1D2*(AJ1+AJ3)
952  C
953        S(K,L)=P1D6*((R(K,L)+R(K-1,L)+R(K,L-1))*AJ1 +
954      X            (R(K,L-1)+R(K-1,L)+R(K-1,L-1))*AJ3)
955  C
956  63  CONTINUE
957  65  CONTINUE
958  C
959  C*****************
960  C*                *
961  C*    DEBUG EDIT   *
962  C*                *
963  C*****************
964  C
965        IF(DEBUG.EQ.0.)GO TO 80
966  C
967  C
968  C   PRINT NBC BOUNDARY SENTINELS
969  C
970        WRITE(NO,71)(NBC(K,LMN),K=KMN,KMX)
971  71  FORMAT(3HLMN,80I1)
972        WRITE(NO,72)(NBC(K,LMX),K=KMN,KMX)
973  72  FORMAT(3HLMX,80I1)
974        WRITE(NO,73)(NBC(KMN,L),L=LMN,LMX)
975  73  FORMAT(3HKMN,80I1)
976        WRITE(NO,74)(NBC(KMX,L),L=LMN,LMX)
977  74  FORMAT(3HKMX,80I1)
978  C
979  C   CALL EDIT
980  C
981  80  CONTINUE
982  C
983        WRITE(NO,85)
984        WRITE(NTTY,85)
985  85  FORMAT(21H GENERATION COMPLETED)
986  C
987        RETURN
988        END
```

```
992   C      SUBROUTINE EDIT
993   C
994          COMMON /KLS/ K,L,DEBUG,VERSION,WHER,WHEN,P1D6,PIE,IGEN,P1D2
995         X ,DTC,KC,LC,DTEN,KEN,LEN,SKE,HN,SIEL,CNN,ENC,ENH,ENCG,WN
996         X ,NCP
997   C
998          COMMON /COMN/  R(33,33),Z(33,33),U(33,33),RHO(33,33),Q(33,33)
999         X       ,E(33,33),P(33,33),AJ(33,33),S(33,33),NBC(33,33)
1000        X ,   W(33,33),TEMP(33,33)
1001        X , A(33,33),B(33,33),CC(33,33),DUM(33,33),CBB(33,33)
1002        X , DBB(33,33),CAP(33,33),SIG(33,33),TS(33,33)
1003  C
1004         COMMON /PARAM/ NYCL,TNUP,DTNUP,DTN,DTNPH,DTNMH,EDTIME,EDDT
1005        X       ,GAM,GAMZ,COF,C1F,C1,TMAX,DTMAX,DTMIN,TFLR,NOHYD
1006        X ,C2,P2,P3,N0,NTTY,NED
1007  C
1008         COMMON /KLSPACE/ KMN,LMN,KMX,LMX,KMXZ,LMXZ,KMNP,LMNP,KMXP,LMXP
1009  C
1010         COMMON /MINMAX/ XMIN,XMAX,YMIN,YMAX,PMIN,PMAX,QMIN,QMAX
1011        X,RMIN,RMAX,KQ,LQ,KR,LR,KP,LP
1012        X ,XMINX,XMAXX,YMINX,YMAXX
1013  C
1014         COMMON /TIMING/ NBT(20),NCT(20),NET(20),NPT(20),NXT(20)
1015  C
1016  C
1017  C
1018  C   TEMPIS SUBROUTINE EDITS ALL MESH VARIABLES
1019         DATA N100/100/
1020         I1=0
1021  C
1022  C
1023  C   INITIALIZE MINIMUM AND MAXIMUM VALUES OF RHO, P, Q, R AND Z
1024  C
1025         RMIN=1.E+6
1026         RMAX=-1.E+6
1027  C
1028         PMIN=1.E+6
1029         PMAX=-1.E+6
1030  C
1031         QMIN=1.E+6
1032         QMAX=-1.E+6
1033  C
1034         XMIN=1.E+6
1035         XMAX=1.E+6
1036  C
1037         YMIN=1.E+6
1038         YMAX=1.E-6
1039  C
1040  C   INITIALIZE LOCATION OF MAXIMUM VALUES OF RHO, P AND Q
1041  C
1042         KR=0
1043         LR=0
1044  C
1045         KP=0
1046         LP=0
1047  C
1048         KQ=0
1049         LQ=0
1050  C
1051  C   FIND THE MINIMUM AND MAXIMUM VALUES OF RHO, P, Q, R AND Z
```

```
1052         DO 715 L=LMNP,LMX
1053   C
1054         DO 714 K=KMNP,KMX
1055   C
1056         IF(RHO(K,L).LE.RMAX)GO TO 701
1057         RMAX=RHO(K,L)
1058         KR=K
1059         LR=L
1060   C
1061   701   CONTINUE
1062   C
1063         IF(P(K,L).LE.PMAX)GO TO 702
1064         PMAX=P(K,L)
1065         KP=K
1066         LP=L
1067   C
1068   702   CONTINUE
1069   C
1070         IF(Q(K,L).LE.QMAX)GO TO 703
1071         QMAX=Q(K,L)
1072         KQ=K
1073         LQ=L
1074   C
1075   703   CONTINUE
1076   C
1077         RMIN=AMIN1(RMIN,RHO(K,L))
1078         PMIN=AMIN1(PMIN,P(K,L))
1079         QMIN=AMIN1(QMIN,Q(K,L))
1080   C
1081         XMIN=AMIN1(XMIN,R(K,L))
1082         XMAX=AMAX1(XMAX,R(K,L))
1083   C
1084         YMIN=AMIN1(YMIN,Z(K,L))
1085         YMAX=AMAX1(YMAX,Z(K,L))
1086   C
1087   714   CONTINUE
1088   C
1089   715   CONTINUE
1090   C
1091   C     PRINT PROBLEM PARAMETERS
1092   C
1093         WRITE(NO,717) NYCL,TNUP,DTNPH,DTN,VERSION,WHER,WHEN
1094   717   FORMAT(6H NYCL ,I6,6H TIME ,E12.4,7H DTNPH ,E12.4,5H DTN ,
1095       X E12.4,9H VERSION ,F4.1,2A10)
1096   C
1097         WRITE(NO,718) PMAX,KP,LP,QMAX,KQ,LQ,RMAX,KR,LR
1098   718   FORMAT(14H MAXIMUM (K,L),E12.4,2I4,3H P ,E12.4,2I4,3H Q ,
1099       X E12.4,2I4,5H RHO )
1100         UVTEST=1.E-5
1101         KL=KMN
1102         LL=LMN
1103         KU=KMX
1104         LU=LMX
1105         IF(IGEN.EQ.0) GO TO 720
1106         UVTEST=-100.
1107   C     PRINT ALL MESH POINTS
1108   C     IGEN.NE.0 WILL RESULT IN EDIT OF ENTIRE MESH,=0 ONLY ACTIVE ZONES
1109         KL=KMN-1
1110         LL=LMN-1
1111         KU=KMX+1
```

```
1112       LU=LMX+1
1113 720   CONTINUE
1114 C BEGIN EDIT
1115       DO 740 L=LL,LU
1116       WRITE(NO,725)
1117 725   FORMAT(8H     L      K, 4X,1HR,10X,1HZ,10X,1HU,10X,1HW,10X,3HRHO,
1118      X                       8X,1HE,10X,1HP,10X,1HQ,10X,2HAJ,9X,5HTHETA)
1119       DO 738 K=KL,KU
1120       IF((ABS(U(K,L))+ABS(W(K,L))).LE.UVTEST)GO TO 738
1121 C DONT PRINT VARIABLES IF NO MOTION
1122       WRITE(NO,726)L,K,R(K,L),Z(K,L),U(K,L),W(K,L),RHO(K,L),E(K,L)
1123      X,P(K,L),Q(K,L),AJ(K,L),TEMP(K,L)
1124 726   FORMAT(2I4,10E11.3)
1125 738   CONTINUE
1126 740   CONTINUE
1127 C
1128       NET(NED)=NECOND(11)
1129 C
1130       NPT(NED)=NECOND(11)
1131       NED=NED+1
1132       IF(NED.GT.20) NED=1
1133 C
1134       RETURN
1135       END
```

```fortran
1136          SUBROUTINE TEMPCAL ( RHO, EPS )
1137    C
1138          COMMON /EOSCOM/ KEOS,TARG1,TARG2,TARG3,RARG1,RARG2,RARG3,
1139        X   FUNC1,FUNC2,FUNC3,TEMPS,EPS,EPS0
1140    C
1141    C   INVERSE TABLE LOOK-UP
1142    C
1143          DATA P1M6/1.E-6/
1144          TARG1=0.
1145    C
1146          CALL IES2
1147    C   E=EOS(0,RHO)
1148          EPS0=FUNC1
1149          TEMPS=0.
1150          IF(EPS.LT.EPS0) RETURN
1151    C   RETURN TEMPETA=0 IF BELOW TABLE
1152          TEMPS=10.*EPS
1153    C   INITIAL GUESS
1154    10    TARG1=TEMPS
1155    C
1156          CALL IES2
1157    C
1158          FUNC2=FUNC1
1159          TARG1=TARG1+P1M6
1160    C
1161          CALL IES2
1162    C
1163          DTEMP=P1M6*((EPS-FUNC2)/(FUNC1-FUNC2))
1164          TEMPS=TEMPS+DTEMP
1165          IF(TEMPS.LT.P1M6) GO TO 20
1166          IF(ABS(DTEMP).GT.P1M6) GO TO 10
1167    C   CONVERGED
1168          RETURN
1169    20    TEMPS=0.
1170          RETURN
1171          END
```

```
1172          SUBROUTINE JES
1173    C
1174          COMMON /EOSCOM/ KEOS,TARG1,TARG2,TARG3,RARG1,RARG2,RARG3,
1175         X  FUNC1,FUNC2,FUNC3,TEMPS,EPS,EPSO
1176    C
1177          COMMON /COM2/ NTSV(2),NRSV(2),MSV(2),TES(7),RES(9),
1178         X ,AES(12),BES(12),CES(12),DES(12),EES(12),FES(12),GES(12)
1179         X ,HES(12),PES(12),ITES(3),IRES(3),IZES(3)
1180    C
1181          N=1
1182          RETURN
1183          ENTRY IES1
1184          N=1
1185          EXTT=1.
1186          EXTR=1.
1187          TARG=TARG1
1188          RARG=RARG1
1189          IBOUND=0
1190          IESTB=1
1191          GO TO 5000
1192    110   CONTINUE
1193          FUNC = AES(M)+RARG*(BES(M)+RARG*DES(M))
1194         1  +TARG*(CES(M)+RARG*(FES(M)+RARG*GES(M))
1195         2  +TARG*(EES(M)+RARG*(HES(M)+RARG*PES(M))))
1196          FUNC1=FUNC*EXTT*EXTR
1197          RETURN
1198    C IES2 ENERGY=FUNCTION(TEMPETA RHO)
1199    C
1200          ENTRY IES2
1201          N=2
1202          EXTT=1.
1203          TARG=TARG1
1204          RARG=RARG1
1205          IBOUND=0
1206          IESTB=2
1207          GO TO 5000
1208    210   CONTINUE
1209          FUNC = AES(M)+RARG*(BES(M)+RARG*DES(M))
1210         1  +TARG*(CES(M)+RARG*(FES(M)+RARG*GES(M))
1211         2  +TARG*(EES(M)+RARG*(HES(M)+RARG*PES(M))))
1212          FUNC1=FUNC*EXTT
1213          RETURN
1214    C            TABLE LOOK UP
1215    C
1216    5000  NT=NTSV(N)
1217          NR=NRSV(N)
1218          MLR = 0
1219          MLT = 0
1220    C
1221          IF(TES(NT).GT.TARG)  GO TO 5100
1222          IF(TES(NT+1).LE.TARG)  GO TO 5200
1223    C
1224    C     TARG IN SAME T STRIP AS FOR PREVIOUS ENTRY
1225    C
1226          IF(RES(NR).GT.RARG)   GO TO 5300
1227          IF(RES(NR+1).LE.RARG) GO TO 5400
1228    C
1229    C     TARG AND RARG IN SAME BOX AS FOR PREVIOUS ENTRY
1230    C     M SAME AS FOR PREVIOUS ENTRY,FAST RETURN
1231    C
```

```
1232       M=MSV(N)
1233       GO TO (110,210) ,IESTB
1234 C
1235 C     T SEARCH
1236 C
1237 C     TARG BELOW T STRIP OF PREVIOUS ENTRY
1238 C
1239 C     OUT OF TABLE TEST, LOW T
1240 C
1241 5100  IF(NT.LE.ITES(N)) GO TO 5115
1242 C
1243 C     SEARCH TO NEXT LOWER T STRIP
1244 C
1245 5105  NT=NT-1
1246       IF(TES(NT).GT.TARG) GO TO 5120
1247 C
1248 C     STRIP CONTAINING TARG FOUND, BEGIN R SEARCH
1249 C
1250       IF(RES(NR)-RARG) 5410,5310,5320
1251 C
1252 C     TARG BELOW LOWEST TABLE ARGUMENT AND
1253 C     WAS BELOW TEMPAT ARGUMENT ON PREVIOUS ENTRY
1254 C
1255 5115  MLT=-1
1256       EXTT=EXTT*TARG/TES(NT)
1257       TARG=TES(NT)
1258 C
1259       IF(RES(NR).GT.RARG) GO TO 5300
1260       IF(RES(NR+1).LE.RARG) GO TO 5400
1261       M = MSV(N)
1262       GO TO (110,210) ,IESTB
1263 C
1264 C     OUT OF TABLE TEST, LOW T
1265 C
1266 5120  IF(NT.GT.ITES(N)) GO TO 5105
1267 C
1268 C     TARG BELOW LOWEST TABLE ARGUMENT BUT
1269 C     WAS NOT BELOW TEMPAT ARGUMENT ON PREVIOUS ENTRY
1270 C
1271       MLT=-1
1272       EXTT=EXTT*TARG/TES(NT)
1273       TARG=TES(NT)
1274 C
1275 C     BEGIN R SEARCH
1276 C
1277       IF(RES(NR)-RARG)    5410,5310,5320
1278 C
1279 C     OUT OF TABLE TEST, HIGH T
1280 C
1281 5200  IF(NT-ITES(N+1)+2) 5205,5215,5205
1282 C
1283 C     SEARCH TO NEXT HIGHER T STRIP
1284 C
1285 5205  NT=NT+1
1286       IF(TES(NT+1).LE.TARG) GO TO 5220
1287 C
1288 C     STRIP CONTAINING TARG FOUND, BEGIN R SEARCH
1289 C
1290       IF(RES(NR)-RARG)    5410,5310,5320
1291 C
```

```
1292 C
1293 C            TARG ABOVE HIGHEST TABLE ARGUMENT AND
1294 C            WAS ABOVE TEMPAT ARGUMENT ON PREVIOUS ENTRY
1295 5215 MLT=1
1296      EXTT=EXTT*TARG/TES(NT+1)
1297      TARG=TES(NT+1)
1298 C
1299      IF(RES(NR).GT.RARG) GO TO 5300
1300      IF(RES(NR+1).LE.RARG) GO TO 5400
1301      M = MSV(N)
1302      GO TO (110,210) ,IESTB
1303 C
1304 C            OUT OF TABLE TEST, HIGH T
1305 C
1306 5220 IF(NT-ITES(N+1)+2) 5205,713,5205
1307 C
1308 C            TARG ABOVE HIGHEST TABLE ARGUMENT BUT WAS
1309 C            NOT ABOVE TEMPAT ARGUMENT ON PREVIOUS ENTRY
1310 C
1311 713  MLT=1
1312      EXTT=EXTT*TARG/TES(NT+1)
1313      TARG=TES(NT+1)
1314 C
1315 C            BEGIN R SEARCH
1316 C
1317      IF(RES(NR)-RARG)  5410,5310,5320
1318 C
1319 C            OUT OF TABLE TEST, LOW R
1320 C
1321 5320 IF(NR.GT.IRES(N)) GO TO 5305
1322 C
1323 C            RARG BELOW LOWEST TABLE ARGUMENT BUT WAS
1324 C            NOT BELOW TEMPAT ARGUMENT ON PREVIOUS ENTRY
1325 C
1326      MLR=-1
1327      EXTR=EXTR*RARG/RES(NR)
1328      RARG=RES(NR)
1329      GO TO 5310
1330 C
1331 C            R SEARCH
1332 C            RARG BELOW R STRIP OF PREVIOUS ENTRY
1333 C            OUT OF TABLE TEST, LOW R
1334 C
1335 5300 IF(NR.GT.IRES(N)) GO TO 5305
1336 C
1337 C            RARG BELOW LOWEST TABLE ARGUMENT AND
1338 C            WAS BELOW TEMPAT ARGUMENT ON PREVIOUS ENTRY
1339 C
1340      MLR=-1
1341      EXTR=EXTR*RARG/RES(NR)
1342      RARG=RES(NR)
1343      M = MSV(N)
1344      GO TO (110,210) ,IESTB
1345 C
1346 C            SEARCH TO NEXT LOWER R STRIP
1347 C
1348 5305 NR=NR-1
1349      IF(RES(NR) - RARG) 5310,5310,5320
1350 C
1351 C            BOX CONTAINING TARG AND RARG FOUND, COMPUTE NEW M
```

```
1352       C
1353 5310  M=IZES(N)+(ITES(N+1)-ITES(N)-1)*(NR-IRES(N))+NT-ITES(N)
1354       NTSV(N)=NT
1355       NRSV(N)=NR
1356       MSV(N)=M
1357       GO TO (110,210) ,IESTB
1358       C
1359       C          OUT OF TABLE TEST, HIGH R
1360       C
1361 5400  IF(NR - IRES(N+1)+2) 5405,5415,5405
1362       C
1363       C          SEARCH TO NEXT HIGHER R STRIP
1364       C
1365 5405  NR=NR+1
1366 5410  IF(RES(NR+1).GT.RARG) GO TO 5310
1367       IF(NR-IRES(N+1)+3) 5405,5405,719
1368       C
1369       C          RARG ABOVE HIGHEST TABLE ARGUMENT BUT WAS
1370       C          NOT  ABOVE TEMPAT ARGUMENT ON PREVIOUS ENTRY
1371       C
1372 719   MLR=1
1373       EXTR=EXTR*RARG/RES(NR+1)
1374       RARG=RES(NR+1)
1375       GO TO 5310
1376       C
1377       C          RARG ABOVE HIGHEST TABLE ARGUMENT BUT
1378       C          M SAME AS ON PREVIOUS ENTRY
1379       C
1380 5415  MLR=1
1381       EXTR=EXTR*RARG/RES(NR+1)
1382       RARG=RES(NR+1)
1383       M = MSV(N)
1384       GO TO (110,210) ,IESTB
1385       END
```

```
1386          SUBROUTINE SETUP
1387    C
1388          COMMON /COM2/ NTSV(2),NRSV(2),MSV(2),TES(7),RES(9)
1389         X ,AES(12),BES(12),CES(12),DES(12),EES(12),FES(12),GES(12)
1390         X ,HES(12),PES(12),ITES(3),IRES(3),IZES(3),IZES(3)
1391    C
1392          CALL JES
1393    C DEFINE A GAMMA LAW GAS EQUATION OF STATE FOR BIQUAD ROUTINE
1394          NTSV(1)=1
1395          NRSV(1)=1
1396          MSV(1)=1
1397          NTSV(2)=4
1398          NRSV(2)=5
1399          MSV(2)=7
1400          ITES(1)=1
1401          IRES(1)=1
1402          IZES(1)=1
1403          ITES(2)=4
1404          IRES(2)=5
1405          IZES(2)=7
1406          ITES(3)=7
1407          IRES(3)=9
1408          IZES(3)=13
1409          TES( 1)=
1410          TES( 2)=  1.0000E+00
1411          TES( 3)=  1.0000E+02
1412          TES( 4)=
1413          TES( 5)=  1.0000E+00
1414          TES( 6)=  1.0000E+02
1415          TES( 7)=
1416          RES( 1)=
1417          RES( 2)=  3.0000E+00
1418          RES( 3)=  3.0000E+02
1419          RES( 4)=  3.0000E+10
1420          RES( 5)=
1421          RES( 6)=  3.0000E+00
1422          RES( 7)=  3.0000E+02
1423          RES( 8)=  3.0000E+10
1424          RES( 9)=
1425          AES( 1)=
1426          BES( 1)=
1427          CES( 1)=
1428          DES( 1)=
1429          EES( 1)=
1430          FES( 1)=  6.6667E-02
1431          GES( 1)= -1.2953E-16
1432          HES( 1)= -4.4409E-16
1433          PES( 1)= -9.2519E-17
1434          AES( 2)=
1435          BES( 2)= -4.7184E-16
1436          CES( 2)=
1437          DES( 2)= -1.7146E-16
1438          EES( 2)=
1439          FES( 2)=  6.6667E-02
1440          GES( 2)= -4.8247E-17
1441          HES( 2)= -1.0408E-17
1442          PES( 2)= -2.3426E-18
1443          AES( 3)=
1444          BES( 3)=
1445          CES( 3)= -8.0183E-17
```

```
1446   DES( 31) =  -5.8595E-16     0E+00
1447   EES( 31) =   6.6667E-02
1448   FES( 31) =   2.2166E-18
1449   GES( 31) =  -4.4409E-16
1450   HES( 31) =  -4.8187E-19
1451   PES( 31) =  -1.9697E-15
1452   AES( 41) =  -1.1102E-15
1453   BES( 41) =   3.3180E-15
1454   CES( 41) =  -3.5824E-19
1455   DES( 41) =  -3.4404E-17
1456   EES( 41) =   6.6667E-02
1457   FES( 41) =  -1.3792E-18
1458   GES( 41) =  -3.4941E-18
1459   HES( 41) =  -2.7257E-21
1460   PES( 41) =            0E+00
1461   AES( 51) =  -9.4344E-14
1462   BES( 51) =            0E+00
1463   CES( 51) =            0E+00
1464   DES( 51) =  -3.3555E-14
1465   EES( 51) =   6.6667E-02
1466   FES( 51) =            0E+00
1467   GES( 51) =            0E+00
1468   HES( 51) =            0E+00
1469   PES( 51) =            0E+00
1470   AES( 61) =  -4.5247E-14
1471   BES( 61) =  -8.3441E-15
1472   CES( 61) =  -8.8567E-27
1473   DES( 61) =   6.3663E-16
1474   EES( 61) =   2.0424E-02
1475   FES( 61) =            0E+00
1476   GES( 61) =   1.8567E-28
1477   HES( 71) =            0E+00
1478   PES( 71) =            0E+01
1479   AES( 71) =            0E+00
1480   BES( 71) =            0E+00
1481   CES( 71) =   2.0364E-15
1482   DES( 71) =            0E+00
1483   EES( 71) =            0E+00
1484   FES( 81) =            0E+00
1485   GES( 81) =            0E+00
1486   HES( 81) =            0E+00
1487   PES( 81) =            0E-17
1488   AES( 81) =            0E-01
1489   BES( 81) =            0E+00
1490   CES( 91) =   2.9800E-15
1491   DES( 91) =   6.4311E-16
1492   EES( 91) =   6.5417E-20
1493   FES( 91) =   2.0933E-18
1494   GES( 91) =            0E+00
1495   HES( 91) =            0E+00
1496   PES( 91) =            0E+00
1497   AES( 91) =   1.0000E-01
1498   BES( 91) =            0E+00
1499   CES( 91) =            0E+00
1500   DES( 91) =   4.4409E-16
1501   EES( 91) =  -2.6587E-17
1502   FES( 91) =  -6.0233E-21
1503   GES( 91) =  -1.9214E-17
1504   HES( 91) =   6.0233E-20
1505   PES( 91) =
```

```
1506   AES(  1) =  1.8874E-15
1507         =  -.1273E-17
1508   ...   = -1.0000E-01
1509         = -6.2168E-20
1510         =  -.8510E-18
1511   ...   = -4.2030E-20
1512   AES(  1) = -1.3521E-19
1513         = -1.2168E-23
1514   ...   =    .0E+00
1515   DES(  1) =  1.0000E+00
1516         =    .0E+00
1517   EES(  1) =  8.8818E-16
1518   FES(  1) = -5.8341E-26
1519   GES(  1) =  6.3947E-36
1520   HES(  1) =  5.8341E-26
1521   PES(  1) = -1.8808E-36
1522   AES(  2) = -1.9429E-15
1523   BES(  2) =  8.9878E-26
1524   CES(  2) = -1.0000E-01
1525   DES(  2) = -5.3701E-36
1526   EES(  2) =  6.9389E-18
1527   FES(  2) =  8.9962E-26
1528   GES(  2) = -2.8867E-36
1529   HES(  2) =  6.4134E-29
1530   PES(  2) = -1.8745E-38
1531   RETURN
1532   END
1533
1534
```

```
1535   C     SUBROUTINE PROJCT
1536   C
1537   C     THIS SUBROUTINE REFLECTS AN INTERIOR POINT ACROSS THE BOUNDARY
1538
1539         COMMON /PROGG/RO,ZO,R1,Z1,RP,ZP,RR,ZZ
1540   C
1541   C     REFLECT (RP,ZP) TO (RR,ZZ)
1542   C     WHERE (RO,ZO) AND (R1,Z1) ARE BOUNDARY POINTS
1543   C
1544         WW=(2.*(Z1-ZO))/((R1-RO)**2+(Z1-ZO)**2)
1545         ALP=1.-(Z1-ZO)*WW
1546         BET=(R1-RO)*WW
1547         RR=RO+(RP-RO)*ALP + (ZP-ZO)*BET
1548         ZZ=ZO+(RP-RO)*BET - (ZP-ZO)*ALP
1549   C
1550         RETURN
1551         END
```

```
1552            SUBROUTINE CONDUCT
1553      C
1554            COMMON /KLS/ K,L,DEBUG,VERSION,WHER,WHEN,P1D6,PIE,IGEN,P1D2
1555          X ,DTC,KC,LC,DTEN,KEN,LEN,SKE,HN,SIEL,CNN,ENC,ENH,ENCG,WN
1556          X ,NCP
1557      C
1558            COMMON /COMN/ R(33,33),Z(33,33),U(33,33),RHO(33,33),Q(33,33)
1559          X          ,E(33,33),P(33,33),AJ(33,33),S(33,33),NBC(33,33)
1560          X , W(33,33),TEMP(33,33)
1561          X , A(33,33),B(33,33),CC(33,33),DUM(33,33),CBB(33,33)
1562          X , DBB(33,33),CAP(33,33),SIG(33,33),TS(33,33)
1563      C
1564            COMMON /PARAM/ NYCL,TNUP,DTNUP,DTN,DTNPH,DTNMH,EDTIME,EDDT
1565          X          ,GAM,GAMZ,COF,C1F,C1,TMAX,DTMAX,DTMIN,TFLR,NOHYD
1566          X          ,C2,P2,P3,NO,NTTY,NED
1567      C
1568            COMMON /KLSPACE/ KMN,LMN,KMX,LMX,KMXZ,LMXZ,KMNP,LMNP,KMXP,LMXP
1569      C
1570      C
1571            COMMON /EOSCOM/ KEOS,TARG1,TARG2,TARG3,RARG1,RARG2,RARG3,
1572          X FUNC1,FUNC2,FUNC3,TEMPS,EPS,EPSO
1573      C
1574      C     ELECTRON CONDUCTION -LU-
1575      C
1576            DO 10 L=LMN,LMX
1577            DO 10 K=KMN,KMX
1578            CAP(K,L)=.1
1579            CC(K,L)=(.0001*SQRT(TEMP(K,L))*TEMP(K,L)**2)/AJ(K,L)
1580            SIG(K,L)=DUM(K,L)*CAP(K,L)
1581            TS(K,L)=TEMP(K,L)/DTNPH
1582        10  CONTINUE
1583      C
1584            DO 12 L=LMN,LMX
1585            DO 12 K=KMN,KMXZ
1586            CBB(K,L)=(2.*CC(K+1,L)*CC(K+1,L+1))/(CC(K+1,L)+CC(K+1,L+1))
1587          X * (.5*(R(K,L)+R(K+1,L))*((R(K+1,L)-R(K,L))**2)
1588          X          +(Z(K+1,L)-Z(K,L))**2) )
1589        12  CONTINUE
1590            DO 14 L=LMN,LMXZ
1591            DO 14 K=KMN,KMX
1592            DBB(K,L)=(2.*CC(K+1,L+1)*CC(K,L+1))/(CC(K+1,L+1)+CC(K,L+1))
1593          X * (.5*(R(K,L)+R(K,L+1))*((R(K,L+1)-R(K,L))**2)
1594          X          +(Z(K,L+1)-Z(K,L))**2) )
1595        14  CONTINUE
1596      C
1597      C     BOUNDARY CONDITIONS
1598            DO 17 L=LMN,LMX
1599            A(KMN,L)=0.
1600            B(KMN,L)=TEMP(KMN,L)
1601            DBB(KMN,L)=0.
1602        17  CONTINUE
1603      C
1604            DO 19 K=KMN,KMX
1605            A(K,LMN)=0.
1606            B(K,LMN)=TEMP(K,LMN)
1607            CBB(K,LMX)=0.
1608            CBB(K,LMN)=0.
1609        19  CONTINUE
1610      C
1611      C
```

```
1612 C........... Z SWEEP
1613 C
1614       DO 53 K=KMNP,KMX
1615       DO 51 L=LMNP,LMX
1616       DUM(K,L)=SIG(K,L)+CBB(K-1,L)*(1.-A(K,L-1))
1617       A(K,L)=CBB(K-1,L)/DUM(K,L)
1618       B(K,L)=(SIG(K,L)*TEMP(K,L)+CBB(K-1,L-1)*B(K,L-1)
1619      X )/DUM(K,L)
1620 51 CONTINUE
1621 C.... ALPHA,BETA FORWARD
1622       ML=LMX+1
1623       DO 52 L=LMNP,LMX
1624       ML=ML-1
1625       TEMP(K,ML)=A(K,ML)*TEMP(K,ML+1)+B(K,ML)
1626 52 CONTINUE
1627 C BACK SUBSTITUTION
1628 53 CONTINUE
1629 C
1630 C.......... Z SWEEP END
1631 C
1632 C.......... R SWEEP
1633 C
1634       DO 43 L=LMNP,LMX
1635       DO 41 K=KMNP,KMX
1636       DUM(K,L)=SIG(K,L)+DBB(K,L-1)*(1.-A(K-1,L))
1637       A(K,L)=DBB(K,L-1)/DUM(K,L)
1638       B(K,L)=(SIG(K,L)*TEMP(K,L)+DBB(K-1,L-1)*B(K-1,L)
1639      X )/DUM(K,L)
1640 41 CONTINUE
1641 C.... ALPHA BETA FORWARD SWEEP
1642       ML=KMX+1
1643       DO 42 K=KMNP,KMX
1644       ML=ML-1
1645       TEMP(ML,L)=A(ML,L)*TEMP(ML+1,L)+B(ML,L)
1646 42 CONTINUE
1647 C BACK SUBSTITUTION R DIRECTION
1648 43 CONTINUE
1649 C
1650 C.......... R SWEEP END
1651 C COMPUTE DT CONTROL FOR HEAT CONDUCTION
1652 C
1653       YE=0.
1654       KEN=0
1655       LEN=0
1656       DO 111 L=LMNP,LMX
1657       DO 111 K=KMNP,KMX
1658 C GET NEW ENERGY
1659       ENH=ENH+E(K,L)*RHO(K,L)*S(K,L)
1660 C
1661       TARG1=TEMP(K,L)
1662       RARG1=RHO(K,L)
1663       CALL IES2
1664 C
1665       E(K,L)=AMAX1(FUNC1,1.E-30)
1666       ENC=ENC+E(K,L)*RHO(K,L)*S(K,L)
1667       IF(TS(K,L).EQ.0.) GO TO 109
1668       TEMPR=ABS((TEMP(K,L)-TS(K,L))/TS(K,L))
1669       IF(TEMPR.LE.YE) GO TO 109
1670       YE=TEMPR
1671       KEN=K
```

```
1672        LEN=L
1673 109    TEMP(K,L)=TS(K,L)
1674 111    CONTINUE
1675        IF(YE.EQ.0.) GO TO 118
1676        DTEN=(-1*DTNPH)/YE
1677 118    CONTINUE
1678 C   ENERGY BALANCE  HN
1679        DO 122 K=2,KMX
1680        HN=HN+DTNPH*CBB(K-1,LMN)*(TEMP(K,LMN)-TEMP(K,LMN+1))
1681      X    +DTNPH*CBB(K-1,LMX)*(TEMP(K,LMX+1)-TEMP(K,LMX))
1682 122    CONTINUE
1683 C
1684        DO 124 L=2,LMX
1685        HN=HN+DTNPH*DBB(KMN,L-1)*(TEMP(KMN,L)-TEMP(KMN+1,L))
1686      X    +DTNPH*DBB(KMX,L-1)*(TEMP(KMX+1,L)-TEMP(KMX,L))
1687 124    CONTINUE
1688 C
1689        RETURN
1690        END
```

```
1691          SUBROUTINE HWORK
1692    C
1693          COMMON /KLS/ K,L,DEBUG,VERSION,WHER,WHEN,P1D6,PIE,IGEN,P1D2
1694         X ,DTC,KC,LC,DTEN,KEN,LEN,SKE,HN,SIEL,CNN,ENC,ENH,ENCG,WN
1695         X ,NCP
1696    C
1697          COMMON /COMN/ R(33,33),Z(33,33),U(33,33),RHO(33,33),Q(33,33)
1698         X          E(33,33),P(33,33),AJ(33,33),S(33,33),NBC(33,33)
1699         X , W(33,33),TEMP(33,33)
1700         X , A(33,33),B(33,33),CC(33,33),DUM(33,33),CBB(33,33)
1701         X , DBB(33,33),CAP(33,33),SIG(33,33),TS(33,33)
1702    C
1703          COMMON /PARAM/ NYCL,TNUP,DTNUP,DTN,DTNPH,DTNMH,EDTIME,EDDT
1704         X ,GAM,GAMZ,COF,C1F,C1,TMAX,DTMAX,DTMIN,TFLR,NOHYD
1705         X ,C2,P2,P3,NO,NTTY,NED
1706    C
1707          COMMON /KLSPACE/ KMN,LMN,KMX,LMX,KMXZ,LMXZ,KMNP,LMNP,KMXP,LMXP
1708    C
1709    C SUM THE HYDRO WORK ON THE BOUNDARY
1710    C
1711          Z1=DTNPH/8.
1712    C
1713          DO 510 K=KMNP,KMX
1714    C
1715          WN=WN+Z1*(P(K,LMN+1)+P(K,LMN))+Q(K,LMN))
1716         X *( (U(K,LMN)+U(K-1,LMN))*(Z(K,LMN)-Z(K-1,LMN))
1717         X - (W(K,LMN)+W(K-1,LMN))*(R(K,LMN)-R(K-1,LMN))
1718         X )*(R(K,LMN)+R(K-1,LMN))
1719    C
1720          WN=WN-Z1*(P(K,LMX+1)+P(K,LMX))+Q(K,LMX))
1721         X *( (U(K,LMX)+U(K-1,LMX))*(Z(K,LMX)-Z(K-1,LMX))
1722         X - (W(K,LMX)+W(K-1,LMX))*(R(K,LMX)-R(K-1,LMX))
1723         X )*(R(K,LMX)+R(K-1,LMX))
1724    C
1725      510 CONTINUE
1726    C
1727          DO 515 L=LMNP,LMX
1728    C
1729          WN=WN+Z1*(P(KMN+1,L)+P(KMN,L))+Q(KMN,L))
1730         X *( (U(KMN,L)+U(KMN,L-1))*(Z(KMN,L)-Z(KMN,L-1))
1731         X - (W(KMN,L)+W(KMN,L-1))*(R(KMN,L)-R(KMN,L-1))
1732         X )*(R(KMN,L)+R(KMN,L-1))
1733    C
1734          WN=WN-Z1*(P(KMX+1,L)+P(KMX,L))+Q(KMX,L))
1735         X *( (U(KMX,L)+U(KMX,L-1))*(Z(KMX,L)-Z(KMX,L-1))
1736         X - (W(KMX,L)+W(KMX,L-1))*(R(KMX,L)-R(KMX,L-1))
1737         X )*(R(KMX,L)+R(KMX,L-1))
1738    C
1739      515 CONTINUE
1740    C
1741          RETURN
1742          END
```

```
1743        FUNCTION NECOND(IARG)
1744        IAG=0
1745        AA1=SECOND(IAG)
1746        NECOND=(AA1-AA2)*1.E+6
1747        AA2=AA1
1748        RETURN
1749        END
```