Much of the arbitrariness of conventional program solutions to large-scale scientific problems can be removed by the approach presented in this paper. The logical formulation of such problems can be improved by using the programming language APL, *which is mathematically compact and explicit. The language also allows the system much freedom in producing computed results.*

A meteorological problem is discussed as an illustration of APL-*augmented programming. Two approaches to programming the solution are compared.*

Problem formulation using APL

by H. G. Kolsky

The analysis of computer applications and the programming of unbuilt computers for such applications are elements of an uncertain art. A given application problem may be formulated in many different ways; there may be differing opinions even among experts in the same field, also, two different application programmers may cast the same numerical model in different ways on a new machine.

This paper proposes the use of a mathematical programming language, such as APL, for removing some of the arbitrariness from the problem-formulation stage.[1] The purpose of the proposed approach is to eliminate errors in the logical formulation early in the design of a large scientific program. Using a mathematical programming language for expressing the overall program logic in an unambiguous, compact way prevents many of the logical errors that can creep into a program because of its sheer size. As an illustration of this approach to program design, the analysis of a numerical meteorological model is discussed. In our discussion, we compare a general approach using FORTRAN alone during problem formulation with an approach in which APL augments FORTRAN in the problem formulation stage. Although a knowledge of APL is not required for an understanding of this paper, an interest in programming large mathematical problems is assumed.

The APL language is used here because it is a powerful and concise way of representing complex relationships and because it exists in the form of a time-sharing terminal language[2] that enables the testing of parts of the program as they are written.

The main theme of this paper is to illustrate the power and utility of such a language as a mode of mathematical notation and expression for programming purposes.

Although the APL language may seem alien and difficult to read at first, the power of the notation is derived from such conventions and definitions as the right-to-left execution convention. Variables in APL equations need not be scalar quantities, but may also be vectors, matrices, and arrays of higher dimensions. The notation is highly consistent internally, whereas standard mathematical notation is a conglomerate of conventions that have developed over many years from many different sources. The structure of the *monadic* and *dyadic* functions (shown in the Appendix) is very general and applies to data arrays of many ranks and mixed types. (A monadic function is one that takes a single argument, and a dyadic function takes two arguments.) In APL one writes $\mid A$ for the absolute value of A. The vertical bar is a monadic operator, whereas the bar in $B \mid A$ means the residue of A modulo B. The standard mathematical notations are $|A|$ and A mod B. In FORTRAN, on the other hand, one writes ABS(A) and MOD1 (B, A).

From a machine architecture point of view, the most important aspect of APL is the large amount of freedom in the order of execution of the individual arithmetic steps. This can be very important in the allocation of resources of a multiprocessor or a vector processor. For example, when one writes $A = B + C$ in APL, where A, B, and C are three-dimensional matrices, this implies that a large number of additions of components of B and C yield the corresponding components of A. However, nothing is said concerning the order in which these additions take place or concerning the number that can take place simultaneously. A suitable compiler could use this freedom to avoid storage or other resource allocation conflicts when necessary. Thus, the APL formulation is very concise concerning the final results desired, but allows considerable intermediate freedom whereby the system achieves these results.

Large-problem formulation

general computational procedures

Since the meteorological application discussed in this paper exemplifies partial differential equations, some of the general considerations involved in their formulation for numerical solution are now presented. The first step is to understand the physical phenomena and the applicable fundamental physical laws.

From this understanding of the physical problem, a mathematical model is prepared, usually in the form of partial differential equations with approximations for *subscale* and *superscale* phenomena. These phenomena are physical effects that are either much smaller or much larger than those being studied. The differential equation model is then transformed into a difference equation form for computational purposes. The numerical meteoro-

logical simulation equations discussed later are concerned with numerically tracing the time development of large-scale cyclonic motions in the atmosphere. Small-scale atmospheric motions, such as storm fronts and local thunderstorms, are considered subscale. Long-term phenomena, such as glacial formation during ice ages, are considered superscale in time for the model. However, some subscale motions in the meteorological model are of sufficient importance to be included in approximations that take into account average effects of turbulent and convective motions smaller than the grid size in the difference equations. This is done by including diffusion terms in the model.

Difference equations are conventionally formulated using a notation involving the components of the variables. This formulation usually involves the use of subscripts to index spatial locations and superscripts for the time steps of a given variable. For example, $T^n_{k,l,m}$ implies the temperature at time step n at point indexed location k, l, and m. When the differentials in the partial differential equations are replaced by finite differences, the subscripts and superscripts in some of these equations can become quite involved.

The next step in the general procedure is to lay out storage and data flow for the model. It is here that the machine dependence of the calculation is strongest. Often, the number of points or the horizontal spacing in a meteorological model are determined largely on the basis of the storage size. A FORTRAN or PL/I program is prepared using the difference equations, storage layout, and data flow. Frequently, the difference equations and storage layout are modified during the writing of such a program, and the original mathematical model may be modified in this process. Finally, before making test experiments with the complete program, there is a debugging and checkout of the program parts.

APL procedure

The APL procedure starts with the same physical phenomena and mathematical model, but makes the transition to difference equations by using a "compact" notation. This notation is an extension of that which has been used by Shuman, Smagorinsky, and others in the field.[3,4] The aim of the procedure discussed here is toward an APL formulation in which the data arrays are considered as a whole and not as isolated components. After setting up the difference equations in compact notation, a layout and data flow analysis of the problem is made using the APL notation. The next step is to program the problem in APL and to check the logic of individual pieces using dummy data. The APL formulation is then transcribed into a higher-level language, such as FORTRAN or PL/I, for execution. Even though additional errors may creep into the problem during the program transcription, such errors should be more "localized" and easily caught in checking out the FORTRAN program. Transcription is necessary because the current APL system cannot handle data arrays of the size required in the meteorological model and is presently

available only as an interpretive time-shared system. Although program debugging, checkout, and the test experiments are conventionally performed, most of the logic and data flow of the program have been previously checked by using the APL procedure.

Of course, there is a class of errors for which segmentary debugging of the logic is not sufficient. Parts may work correctly, but the problem as a whole may be unstable. However, these errors really represent incorrect numerical modeling, not flaws in the logic or data flow. Research weather calculations have produced marked examples of such instability. Smagorinsky[4] found that certain types of neutral instabilities can require in the order of hundreds of time-steps to reveal themselves. This corresponds to advancing the weather model by as much as thirty days to verify the stability of the numerical method.

averaging and differencing operators

There are many ways in which finite differences may be written, all of which reduce to the same differential operator in the limit, when the spacing goes to zero. An example of a nonlinear partial differential equation is as follows:

$$\frac{\partial u}{\partial t} = u \frac{\partial u}{\partial x} \tag{1}$$

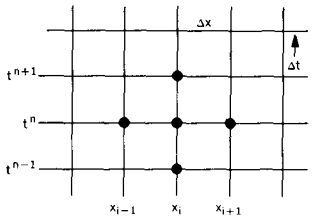
A possible difference-equation representation of Equation 1 is:

$$\frac{\Delta u}{\Delta t} = u \frac{\Delta u}{\Delta x} \tag{2}$$

A difference star for one point in a graphic representation for Equation 2 is shown in Figure 1. One particular scheme for solving Equation 2 is referred to as the *leapfrog explicit difference* method. This method is defined by Equation 3.

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} = u_i^n \left(\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \right) \tag{3}$$

Figure 1 Star representation of a difference-equation



"Explicit" implies that the value of the quantity at the next time step, $n + 1$, appears explicitly in the equation. "Leapfrog" means that the derivatives are centered differences taken straddling the point x_i at time n. This finite-difference formulation may be written in compact notation as shown in Equation 4.

$$\overline{U_t^t} = U\overline{U_x^x} \tag{4}$$

where

$$U_t = \frac{u^{t+1/2} - u^{t-1/2}}{\Delta t} \quad \text{and}$$

$$\overline{U^t} = \frac{u^{t+1/2} + u^{t-1/2}}{2} \quad \text{so that}$$

$$\overline{U_t^t} = \frac{u^{t+1} - u^{t-1}}{2\Delta t}$$

In this notation, the subscripts represent differences and the bars represent averages in the variable specified.

An important property of the averaging and differencing operators is that they change the centering of the points from integer to half-integer values. If a variable is defined at half-integer values, such as $x_{i+\frac{1}{2}}$, then applying either the differencing or the averaging operator results in a quantity that is centered at integer values x_i. If a series of operators is applied sequentially to a variable, then the results alternate between integer and half-integer values. An even number of operators causes the result to be centered at the original point. A possible by-product of this method is a saving in computation time. In performing either a differencing or an averaging operation, the resultant values have one fewer meaningful element than the original values. That is, if one averages the elements of a vector **g**, the first with the second, the second with the third, etc., the last element has no subsequent element with which to be averaged.

In addition to the averaging and differencing operators previously mentioned, it is useful to generalize the averaging operator to include a weighted averaging operator. This is necessary to describe cases where the dimension in which the differencing performed is not equally divided, and where the function being averaged is centered at the same points at which Δx is centered. The weighted averaging operator is

$$\left(\frac{\overline{f}^{x}}{f^{x}}\right)_i = A \cdot f_{i+1/2} + B \cdot f_{i-1/2} \tag{5}$$

where

$$A = \frac{\Delta x_{i-1/2}}{\Delta x_{i-1/2} + \Delta x_{i+1/2}}$$

and

$$B = \frac{\Delta x_{i+1/2}}{\Delta x_{i-1/2} + \Delta x_{i+1/2}}$$

This operator is used particularly in the vertical dimension of the weather model because the vertical dimension is not differenced equally.

Numerical meteorological modeling

We now consider some of the overall computational aspects of a large-scale problem aimed toward its later description in the APL language. Chosen as an illustrative model is a numerical meteorological research calculation originally written by C. E. Leith[5] at the University of California, Lawrence Radiation Laboratory, Berkeley, California. At the time it was written, this model (Final Large Atmospheric Model—FLAM) was an advanced research model for the numerical simulation of the earth's atmosphere. FLAM was written independently of the U. S. Weather Bureau. Since the mid-1950's, Weather Bureau models have used a plain rectangular or octagonal grid of the northern

hemisphere only, whereas Leith's model uses a latitude and longitude mapping scheme for the whole globe. His model also involves the solution of the primitive equations of atmospheric motion and energy transfer in place of empirical relationships that characterize some production-oriented models.

storage

Leith's meteorological model resembles other large fluid dynamic problems in that it can require a much larger storage for the primary variables than is usually available in existing computers. The FLAM model, originally written in FORTRAN, uses magnetic tape units as external storage. Much of the FLAM program is associated with the blocking and packing of data to and from the tapes. The preparation of output tapes for printing is another sizeable part. The main logic of the outer loops of the program is concerned with the manipulation of these storage blocks.

The size of a weather problem can be estimated by the number of mesh points used in the difference equations for the model. The length of a time step is determined by stability requirements. In the case of FLAM, the horizontal spacing of the mesh is five degrees on a side at the equator with varying angular spacing in the east-west direction when approaching the poles. A number of physical quantities are stored for each atmospheric mesh point, and several special quantities are stored for each surface mesh point. In addition, several quantities that vary only with longitude or only with latitude are stored as one-dimensional vectors. A given physical parameter of the atmosphere is thus represented in the numerical model as a three-dimensional array of numbers (or, more precisely, as two three-dimensional arrays of numbers, one for each hemisphere).

Often in the logic of a problem, similar calculations could be done on a particular physical parameter for all mesh points of the array. The nature of FORTRAN, however, is such that computations are done on one number at a time, that is, one scalar member of the multidimensional array at a time. Therefore, a three-dimensional array computation requires at least a triple DO-loop or equivalent to perform the computation. An advantage of the APL formulation, in which the equations contain multidimensional arrays as their basic elements, is that many DO-loops are eliminated, thereby improving program logic. The remaining loops are those having to do with the actual program flow.

It should not be assumed that an APL formulation is structurally machine-independent. As is true in other large computer programs, there is always a trade-off between generality and specialization and between storage capacity and speed. In principle, one could write APL equations so that an entire matrix of 500,000 words in a meteorological problem is referenced in one statement. This extreme example might result in a simpler formulation, but it certainly is not a practical formulation of the problem. It is better to reference individual parts of data arrays

in groups of possibly 10,000 words, since such groups are more likely to be containable within the high-speed storage.

In the formulation presented here, Leith's methods are used to conserve storage by processing submatrices of data for one latitude line at a time. Storage is so organized that data normally located in adjacent storage locations are transferred as a block. Thus, block transfer of data from external storage to the high-speed storage is simulated.

input/output

Leith's model is similar to other numerical meteorological problems in that the amount of computation that must be done per data block read in or read out is large enough that the total program need not be I/O-limited, provided that input/output can be scheduled to be simultaneous with the computation. In other words, total computation time is large enough that the input/output time does not cause a fundamental inefficiency. The requirement for simultaneous input/output and computation does mean that data blocks must be handled properly to avoid attempting to use data that is not yet available or destroying data that is not yet read out. The APL formulation of the FLAM program to be described breaks each time-step into two data cycles. This is done mainly to keep the high-speed storage requirement as small as possible.

fluid dynamics problems

Two difficulties of large fluid dynamics problems are those of handling special and boundary cases. In APL, these cases can be represented by the sides and edges of three-dimensional data arrays. The APL statement of a problem can thus take the form of a general expression for the array as a whole, plus special statements for certain of the sides of edges of the array. The properties of APL have two significant effects. One is that the total length of a program written in APL is much shorter than that of FORTRAN. (Ratios of five or ten to one are not uncommon.) Also, special cases are stated explicitly and not obscured in the programming.

Concerning special boundary conditions in Leith's model, quantities are continuous around the earth at a given latitude. That is, if the problem computation is indexing eastward from the Greenwich meridian at a given latitude, when returning to the starting point, quantities just to the west of Greenwich are adjacent to those at Greenwich. Such a circular boundary condition is handled naturally and automatically in APL by the Rotate function.

Primitive equation models in numerical meteorological research differ considerably in the way in which they handle physical effects such as wind friction at the earth's surface, and the treatment of mountains and snow cover. Finite difference approximation methods typically use an Eulerian spatial mesh that is fixed in time and through which the fluid flows. A mesh that follows the fluid is called a Lagrangian mesh. One of the main difficulties in using the Eulerian mesh is correctly computing the advection of physical quantities of the fluid flow. (Advection, as used here, implies the three-dimensional motion of an air mass.) We now

discuss Leith's treatment of stable advection calculations and methods that yield good approximations for uniform fluid flow.

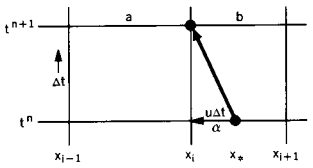
APL formulation of the meteorological problem

The problem of advection in one spatial dimension x is characterized by a fixed fluid velocity u. If Y is the value of a dependent variable, that is, a quantity imbedded in and unchanged at a material point during the flow, we have the equation

$$\frac{DY}{Dt} = \frac{\partial Y}{\partial t} + u\frac{\partial Y}{\partial x} = 0$$

Instead of directly setting up finite-difference approximations of the partial derivatives $\partial/\partial t$ and $\partial/\partial x$, it is helpful to consider again the Lagrangian point of view. The heavy arrow in Figure 2 shows the characteristic space-time path of the material point, which is at a position x_i at time t^{n+1}. At time t^n, this point was at position $x_* = u\Delta t$. If we know the value of $Y = Y_*^n$ at time t^n, we can set

Figure 2 Space-time path for the advection problem



$$Y_i^{n+1} = Y_*^n$$

At time t^n, however, we only know values of Y at mesh points x_{i-1}, x_i, x_{i+1}, etc., and we have to use an interpolation procedure to determine Y_*^n. The three values Y_{i-1}^n, Y_i^n, Y_{i+1}^n determine a quadratic dependence of Y on x, which is evaluated at x_* to give Y_*^n. Assuming a uniformly spaced mesh of interval Δx, the resulting expression used by Leith is

$$Y_i^{n+1} = Y_*^n = Y_i^n - \frac{a}{2}(Y_{i+1}^n - Y_{i-1}^n) + \frac{a^2}{2}(Y_{i+1}^n - 2Y_i^n + Y_{i-1}^n) \tag{6}$$

where

$$a = \frac{x_i - x_*}{\Delta x} = \frac{u\Delta t}{\Delta x}$$

is a dimensionless interpolation parameter. This expression may be written for nonuniform spacing, as shown in Equation 7, using the averaging and differencing operators discussed earlier in this paper.

$$Y^* = Y_i + \alpha\left[\left(\frac{Y_i - Y_{i-1}}{a}\right)\left(\frac{b}{a+b}\right) + \left(\frac{Y_{i+1} - Y_i}{b}\right)\left(\frac{a}{a+b}\right)\right] + a^2\left[\frac{\left(\frac{Y_{i+1} - Y_i}{b}\right) - \left(\frac{Y_i - Y_{i-1}}{a}\right)}{a+b}\right] \tag{7}$$

where

$$\alpha = x^* - x_i = u\Delta t$$

$$a = x_i - x_{i-1}$$

$$b = x_{i+1} - x_i$$

$$Y_i^{n+1} = Y_*^n$$

Using the compact notation previously discussed, Equation 7 may be written in the simpler form of Equation 8.

$$Y_i^* = Y_i + \alpha\left[\frac{w}{(Y_x)^x}\right]_i + \alpha^2[(Y_x)_x]_i \tag{8}$$

Equation 8 is equivalent to the advection formula given in Equation 6, but it is more general because it also applies to nonuniformly spaced meshes. Equation 8 can be written in terms of APL operators for weighted averages and differences as shown in Equation 9.

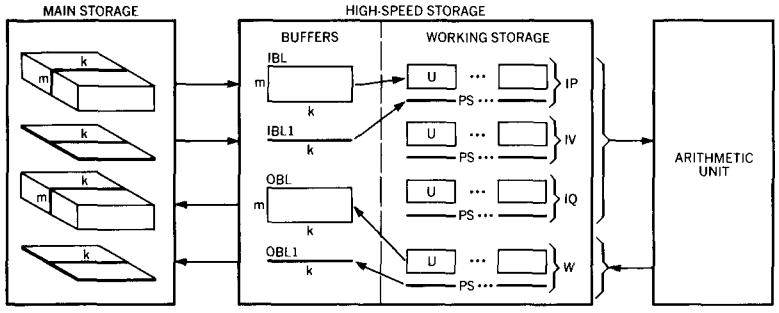
```
YW←YI+(AL1×DXAWX YI)+AL2×DXDX YI          (9)
```

In APL formulations of problems such as those represented by Equation 9, it is convenient to define combination operators, such as *difference then weighted average* (DXAWX), *double average* (AXAY), and *double difference* (DXDX). Also, in the case of our meteorological application, ΔY is constant throughout the problem, and ΔX is constant at a given latitude. Thus, weighted

Table 1 Combination and integration operators for the meteorological problem

```
    ∇ A←AXAYAWP B
[1]   TE←B[IV;;]+B[IP;;]+1⌽[1](B[IP;;]+B[IV;;])
[2]   A←0.25×((DPL×¯1⌽TE)+TE×¯1⌽DPL)÷(DPL+¯1⌽DPL)
[3]   A[;1]←0.25×TE[;1]
    ∇

    ∇ A←DYAY B
[1]   A←(B[IP;;]-B[IQ;;])÷DY×2
    ∇

    ∇ A←DYDY B
[1]   A←(B[IP;;]+B[IQ;;]-2×B[IV;;])÷DY*2
    ∇

    ∇ A←DYAY1 B
[1]   A←(B[IP;]-B[IQ;])÷DY×2
    ∇

    ∇ A←DYDY1 B
[1]   A←(B[IP;]+B[IQ;]-2×B[IV;])÷DY*2
    ∇

    ∇ A←DXAX B
[1]   A←((¯1⌽[1] B)-1⌽[1] B)÷DX×2
    ∇

    ∇ A←DXDX B
[1]   A←((¯1⌽[1] B)+(1⌽[1] B)-2×B)÷DX*2
    ∇

    ∇ A←AYDX B
[1]   A←(1⌽[1](B[IV;;]+B[IP;;])-(B[IV;;]+B[IP;;]))÷DX×2
    ∇

    ∇ A←AXDY B
[1]   A←((((1⌽[1] B[IP;;])+B[IP;;])×COSL[IP]-((1⌽[1] B[IV;;])
        +B[IV;;])×COSL[IV])
    ∇

    ∇ A←INTP B
[1]   M←1
[2]   A←(⍴B)⍴0
[3]   A[;M]←-B[;M]×DPL[;M]
[4]   →(MM<M←M+1)/0
[5]   A[;M]←A[;M-1]-B[;M]×DPL[;M]
[6]   →4
    ∇

    ∇ A←DPAWP B
[1]   TE←((1⌽B)-B)÷DPEL
[2]   A←((TE×¯1⌽DPEL)+DPEL×¯1⌽TE)÷(DPEL+¯1⌽DPEL)
[3]   A[;1]←TE[;1]
    ∇

    ∇ A←DPDP B
[1]   TE←(B-¯1⌽B)÷DPL
[2]   A←((1⌽TE)-TE)÷(DPL+1⌽DPL)
    ∇

    ∇ A←INTPH B
[1]   M←MM-1
[2]   A←(⍴B)⍴0
[3]   A[;M]←(B[;M-1]×HAL[;M])+B[;M]×HBL[;M]
[4]   →(1=M←M-1)/7
[5]   A[;M]←A[;M+1]+(B[;M-1]×HAL[;M])+(B[;M]×HBL[;M])
        +(B[;M+1]×HCL[;M])
[6]   →4
[7]   A[;M]←A[;M+1]+(B[;M]×HBL[;M])+(B[;M+1]×HCL[;M])
    ∇

    ∇ A←AXAYAP B
[1]   TE←B[IV;;]+B[IQ;;]+1⌽[1](B[IV;;]+B[IQ;;])
[2]   A←0.125×(TE+¯1⌽[2] TE)
[3]   A[;1]←0.125×TE[;1]
    ∇

    ∇ A←DPAP B
[1]   TE←((1⌽B)+B)÷DPL
[2]   A←0.5×(TE+¯1⌽TE)
    ∇

    ∇ A←DXAY B
[1]   A←(1⌽[1](B[IV;;]+B[IQ;;])-(B[IV;;]+B[IQ;;]))÷DX×2
    ∇

    ∇ A←DYAX B
[1]   A←(1⌽[1](B[IV;;]-B[IQ;;])+1⌽[1](B[IV;;]-B[IQ;;]))÷DY×2
    ∇
```

Figure 3 Storage and data flow



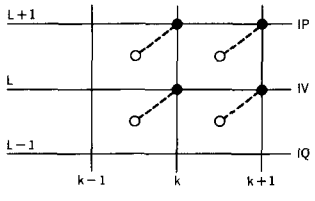
averages may be replaced by ordinary averages except in the vertical dimension. Combination operators become simpler in such special cases, as well as faster to execute on the computer because certain generalized tests are not required. APL statements for such combination operators, as used in the meteorological problems, are given in Table 1.

data flow

An idealized storage and data-flow schematic for the APL formulation of Equation 9 is shown in Figure 3. The computer is assumed to have a large main storage and a smaller high-speed storage. Data are read from the main storage, one latitude line at a time into buffer IBL. These data, called a *block*, consist of all variables in main storage that have the same dimensions; that is, IBL contains all variables having m-by-k components. A separate block (IBL1) is used for data having only k components, such as surface values. These blocked values are then assumed to be placed in working storage of one k-by-m table for each variable.

To allow for finite differencing, data for three latitudes are required to be in high-speed storage at the same time. Latitude data are labeled by indices IP, IV, and IQ in Figure 3. In order to avoid unnecessary moving of data within high-speed storage, the indices IP, IV, and IQ are rotated after each use rather than the data. Computed values are placed in high-speed storage area W, from where they are read out into buffers labeled OBL and OBL1. Contents of these output buffers are then transferred to main storage. Relationships of integer and half-integer values at latitudes (L's) to buffers (k's) and working storage (IP, IV and, IQ) are shown in Figure 4.

Figure 4 Integer, half-integer, and storage relationships



In practice, the unpacking and packing operations, indicated in Figure 3, are unnecessary when going from IBL to IP in high-speed storage, Variables in IBL are already properly separated. However, by writing the storage in this way, the following operations are carried out concurrently:

- Data is transferred from main storage to IBL.
- Data in IP, IV, and IQ are being used to compute W.
- Computed data are read from OBL to the main storage.

Thus, the model assumes simultaneous input/output and compute.

The APL time-sharing system, as it is currently implemented, cannot execute programs of this magnitude. A production meteorological problem requires data blocks from hundreds of thousands to millions of words. Although the time-sharing APL system cannot handle such blocks, it is possible to check out parts of the program and the logic of data manipulation for small data samples. The illustrations in this paper have been done in this mode.

APL advection formula

The FLAM program (Final Large Atmospheric Model) uses the fractional time-step technique in which each time step is divided into three parts: north-south advection, which is done first; east-west, second; vertical motion, third. An example of the general advection equations in one dimension is given in APL as follows:

```
AL1←DLT×AXAYAWP V                    (10)
AL2←(AL1*2)+D1×DLT                   (11)
TW←T[IV;;]+(AL1×DYAY T)+AL2×DYDY T   (12)
```

The coefficient of the AL1 term in Equation 10 uses the triple averaging operator AXAYAWP. This is required because the velocity V is centered at half-spaces in all three dimensions, whereas the temperature is located on the even points (black dots in Figure 4). Temperature is the quantity being advected.

The second step in the general advection formula (Equation 11) consists of adding two terms, i.e., the square of the first term plus a diffusion term. (Physical diffusion is characterized by a different fundamental equation than fluid flow.) Without going through the derivation, the D1 × DLT term in Equation 11 is equivalent to solving a separate diffusion equation, since the coefficient of that term is the same as the second difference in the Y direction.

The third step, indicated by Equation 12 of the general advection formula, uses the DYAY and DYDY operators. This is a case in which the weighted average can be replaced by an ordinary average.

The main APL program for the advection problem is shown in Table 2. The control program (SEQ1) calls the other programs. Referring to Figure 3 and Table 2, the first program called (ILOAD) performs the load operation, in which the first two blocks for IV and IQ are read into working storage before computations begin. This input is a three-dimensional array selected from a four-dimensional array in main storage. (The fourth dimension indicates the names of variables.) An initialization step (i.e., IQ in ILOAD) sets the three indices. Statements in ILOAD labeled with semicolons, but without indices, indicate that the entire dimension is used. Referring again to the control program (SEQ1), the second step tests whether the last latitude line has been reached. If so, the program branches to the step labeled POLE1, which performs special computations for the north or south pole

Table 2 APL storage-handling and control program

CONTROL

```
∇ SEQ1
[1]  ILOAD
[2]  →(LM<L←L+1)/8
[3]  IQ←1+3|IV←1+3|IP←1+3|IP
[4]  LOADS
[5]  CYCL1
[6]  STORES
[7]  →2
[8]  POLE1
[9]  ILOAD
[10] →(LM<L←L+1)/16
[11] IQ←1+3|IV←1+3|IP←1+3|IP
[12] LOADS
[13] CYCL2
[14] STORES
[15] →10
[16] POLE2
```

INITIALIZATION

```
∇ ILOAD
  L←1
  IBL←SM[;L;;]
  IBL1←SM1[;L;]
  IQ←1+IV←1+IP←1
  T[IP;;]←T[IV;;]←IBL[1;;]
  WT[IP;;]←WT[IV;;]←IBL[2;;]
  U[IP;;]←U[IV;;]←IBL[3;;]
  V[IP;;]←V[IV;;]←IBL[4;;]
  PS[IP;]←PS[IV;]←IBL1[1;]
  L←1+L
```

INPUT

```
∇ LOADS
  IBL←SM[;L;;]
  IBL1←SM1[;L;]
  T[IP;;]←IBL[1;;]
  WT[IP;;]←IBL[2;;]
  U[IP;;]←IBL[3;;]
  V[IP;;]←IBL[4;;]
  PS[IP;]←IBL1[1;]
```

OUTPUT

```
∇ STORES
  OBL[1;;]←WT[IV;;]
  OBL[2;;]←WTW[IV;;]
  OBL[3;;]←WU[IV;;]
  OBL[4;;]←WV[IV;;]
  OBL1[1;]←PSW[IV;]
  SM[;L-2;;]←OBL
  SM1[;L-2;]←OBL1
```

Table 3 First main advection/diffusion program in SEQ1

```
      ∇ CYCL1
        NORTH+SOUTH ADVECTION AND DIFFUSION
 [1]    AL1←DLT×AXAYAWP V
 [2]    AL2←(AL1*2)+D1×DLT
 [3]    TW←T[IV;;]+(AL1×DYAY T)+AL2×DYDY T
 [4]    WTW←WT[IV;;]+(AL1×DYAY WT)+AL2×DYDY WT
 [5]    PSW←PS[IV;]+(AL1[;MM-1]×DYAY1 PS)+AL2[;MM-1]×DYDY1 PS
 [6]    AL1←DLT×V[IV;;]
 [7]    AL2←(AL1*2)+D2×DLT
 [8]    UW←U[IV;;]+(AL1×DYAY U)+AL2×DYDY U
 [9]    VW←V[IV;;]+(AL1×DYAY V)+AL2×DYDY V
        EAST+WEST ADVECTION AND DIFFUSION
 [10]   AL1←DLT×AXAYAWP U
 [11]   AL2←(AL1*2)+D3×DLT
 [12]   TW←TW+(AL1×DXAX TW)+AL2×DXDX TW
 [13]   WTW←WTW+(AL1×DXAX WTW)+AL2×DXDX WTW
 [14]   PSW←PSW+(AL1[;MM-1]×DXAX PSW)+AL2[;MM-1]×DXDX PSW
 [15]   AL1←DLT×UW
 [16]   AL2←(AL1*2)+D4×DLT
 [17]   UW←UW+(AL1×DXAX UW)+AL2×DXDX UW
 [18]   VW←VW+(AL1×DXAX VW)+AL2×DXDX VW
      ∇
```

Table 4 Second main advection/diffusion program in SEQ1

```
      ∇ CYCL2
        VERTICAL ADVECTION AND DIFFUSION
 [1]    DIV←(AYDX U)+AXDY V
 [2]    OMW←INTP DIV
 [3]    AL1←DLT×OMW
 [4]    AL2←AL1*2
 [5]    TH←T[IV;;]×RML
 [6]    TW←T[IV;;]+((AL1×DPAWP TH)+AL2×DPDP TH)÷RML
 [7]    WTW←WT[IV;;]+(AL1×DPAWP WT[IV;;])+AL2×DPDP WT[IV;;]
 [8]    PSW←PS[IV;]+DLT×(OMW[;MM]+DIV[;MM]×(PRES[MM]-PS[IV;]))
 [9]    HBL[;MM-1]←HB[MM-1]+(PS[IV;]-PRES[MM-1])×GCNT
 [10]   PH[IV;;]←INTPH(TW×(1+D5×WTW))
 [11]   OM[IV;;]←OMW
 [12]   AL1←AXAYAP OM
 [13]   AL2←AL1*2
 [14]   TE←U[IV;;]
 [15]   WK5←TE+(AL1×DPAWP TE)+(AL2×DPDP TE)+((DPAP TE)×DIH×DLT÷DPL)+DXAY PH
 [16]   TE←V[IV;;]
 [17]   WK6←TE+(AL1×DPAWP TE)+((DPAP TE)×DIH×DLT÷DPL)+DYAX PH
 [18]   WK1←(2×COR[L])+CEN[L]×U[IV;;]
 [19]   WK2←1+WK1*2
 [20]   UW←(WK5+WK1×WK6)÷WK2
 [21]   VW←(WK6-WK1×WK5)÷WK2
      ∇
```

(not discussed in this paper). The third line in the control program rotates the indices IP, IV, and IQ, as previously mentioned. Then the standard load program (line 4) reads the input block from main storage and relabels it in terms of the data for IP.

The first main program called by SEQ1 is (CYCL1), shown in Table 3. This program performs the north-south and the east-west advection and diffusion calculation. After finishing this calculation, the control program calls the STORES program, which reads the newly computed quantities into the output block and puts it in main storage. This is repeated until the whole hemisphere is calculated. The second cycle through the hemisphere begins with the calling of the second ILOAD in the control program, and logic similar to the first cycle is performed. The second main program of SEQ2 is CYCL2 shown in Table 4. This program calculates vertical advection for *all* physical quantities. Most of the physical complications occur in the vertical dimension. When the entire hemisphere is finished, POLE2 in the control program does the completion of the north-south pole computations. The program

then repeats SEQ1 for the southern hemisphere before advancing to the next time cycle. (This is an outer control loop not indicated here.)

Regarding the function of CYCL1 (Table 3), the program is mainly a reapplication of the general advection formula (Equations 10, 11, and 12) for each of the related variables in the two directions, north-south and east-west. Most of the differencing in Equations 10, 11, and 12 in CYCL1 arises from the use of subroutines such as DYAY, DXAX, as shown in Table 1. The equations being solved stand out clearly instead of being obscured in a variety of subscripts and DO-loops, as can be the case in the conventional approach. The APL program for CYCL1 requires one half of a typewritten page, as shown in Table 3, whereas the FORTRAN program requires many pages, including references to several subroutines. The exact ratio, however, is not as important as the fact that the APL listing is much more easily understood than the FORTRAN listing.

The function of the CYCL2 program, shown in Table 4, is complicated by the fact that atmospheric functions vary more rapidly vertically than they do horizontally. The model assumes that the atmosphere is always in hydrostatic equilibrium, which means that the pressure at a given point is determined by the weight per unit area of the air above that point. A differential expression of this assumption is the hydrostatic relation $dp = -g\rho dz$, giving the increment in pressure dp in terms of an increment in height dz for given density ρ. Here g is the (assumed constant) acceleration of gravity that transforms the mass element ρdz into the weight element $g\rho dz$.

In the hydrostatic assumption, the influence of the vertical acceleration on the pressure is neglected. That is, vertical acceleration (averaged over a grid area) is small compared with g. For horizontal scales of motion, large compared to the thickness of the atmosphere, this assumption is thought to be valid. The hydrostatic assumption permits the replacement of vertical displacement z by pressure p as an independent variable. The use of pressure as a vertical coordinate is in keeping with current practice of reducing the number of observations in which the validity of the hydrostatic assumption is assumed. The substitution of pressure for vertical displacement provides a simpler upper boundary to the atmosphere and serves as a mass or weight coordinate. However, the lower boundary becomes free and, thus, more complicated.

There must also be a replacement of p by z as the dependent variable, which yields the geopotential ($\Phi = gz$) i.e., the potential energy per unit mass that serves as a measure of the height of a given pressure surface. In the topography of pressure surfaces, a region that is higher (in the z-coordinate system) corresponds to a region of higher pressure.

Because of the action of gravity, the important vertical quantity is not the absolute temperature but a quantity known as

the "potential temperature." A given parcel of air cools adiabatically when raised to a higher altitude, and potential temperature is a measure of the decrease in absolute temperature with altitude with no net energy change. CYCL2 uses a different mathematical formulation for this than is used by Leith. The present formulation is physically equivalent, but more natural for an APL formulation. In FLAM the potential temperature is computed relative to each individual pressure level. In the present formulation, potential temperatures are all referred to the pressure of one atmosphere, thereby eliminating the need to be computed sequentially.

analysis

The last program, which is not included in the main program but which must be present in a working model, can be called ANALYSIS. In practice, analysis programs are usually relatively short in running time. Averages over various physical observables are computed. Correlations between quantities at different points are calculated. Contour maps, such as surface pressure, precipitation, and geopotential are prepared for display. Often, analysis is most valuable when comparing the averages of two separate calculations having slightly different input parameters. In this case, one must assume the storage of selected results of previous problems in an archival storage, which can be accessed during the analysis phase.

Concluding remarks

The APL formulation of a large-scale scientific problem can specify the solution precisely, while allowing the system much freedom in producing computed results. The testing and debugging stages of such application programming can be carried out at a terminal. The simplicity of APL helps the programmer see the main design of the problem by reducing the program size. In the problem-formulation phase, the APL programmer is aided by the mathematical consistency of the language and by the inherent explicitness of APL program statements from a mathematical point of view.

CITED REFERENCES AND FOOTNOTE

1. K. E. Iverson, "Programming notation in system design," *IBM Systems Journal* **2**, 117–128 (June 1963).
2. The APL\360 program, 360D-03.3.007, which is not formally supported by IBM, and the *APL\360 User's Manual* by A. D. Falkoff and K. E. Iverson may be obtained through any IBM branch office.
3. F. G. Shuman and J. B. Hovermale, "An operational six-layer primitive equation model," *Journal of Applied Meteorology* **7**, No. 4, 525–547 (August 1968).
4. J. Smagorinsky, S. Manabe, and J. L. Holloway, "Numerical results from a nine-level general circulation model of the atmosphere," *Monthly Weather Review* **93**, No. 12, 727–768 (December 1965).
5. C. E. Leith, *Numerical Simulation of the Earth's Atmosphere*, University of California (Berkeley), Lawrence Radiation Laboratory Report, UCRL-7986-T (1964).

Appendix

APL primitive mixed functions

Name	Sign[1]	Definition or example[2]
Size	ρA	$\rho P \leftrightarrow 4$ $\quad \rho E \leftrightarrow 3\ 4$ $\quad \rho 5 \leftrightarrow \iota 0$
Reshape	$V\rho A$	Reshape A to dimension V $\quad 3\ 4\rho\iota 12 \leftrightarrow E$ $\quad 12\rho E \leftrightarrow \iota 12$ $\quad 0\rho E \leftrightarrow \iota 0$
Ravel	$,A$	$,A \leftrightarrow (\times/\rho A)\rho A$ $\quad ,E \leftrightarrow \iota 12$ $\quad \rho,5 \leftrightarrow 1$
Catenate	V,V	$P,\iota 2 \leftrightarrow 2\ 3\ 5\ 7\ 1\ 2$ $\quad$ 'T','HIS' $\leftrightarrow$ 'THIS'
Index[3,4]	$V[A]$	$P[2] \leftrightarrow 3$ $\quad P[4\ 3\ 2\ 1] \leftrightarrow 7\ 5\ 3\ 2$
	$M[A;A]$	$E[1\ 3;3\ 2\ 1] \leftrightarrow$ $3\ 2\ 1$ / $11\ 10\ 9$
	$A[A;..;A]$	$E[1;] \leftrightarrow 1\ 2\ 3\ 4$ $\quad E[;1] \leftrightarrow 1\ 5\ 9$ $\quad$ 'ABCDEFGHIJKL'[E] $\leftrightarrow$ ABCD / EFGH / IJKL
Index generator[3]	ιS	First S integers $\quad \iota 4 \leftrightarrow 1\ 2\ 3\ 4$ $\quad \iota 0 \leftrightarrow$ an empty vector
Index of[3]	$V\iota A$	Least index of A in V, or $1+\rho V$ $\quad P\iota 3 \leftrightarrow 2$ $\quad 5\ 1\ 2\ 5\ \iota\ 4\ 4\iota 4 \leftrightarrow 1$ $\quad P\iota E \leftrightarrow 3\ 5\ 4\ 5\ /\ 5\ 5\ 5\ 5$
Take	$V\uparrow A$	Take or drop $\lvert V[I]$ first $(V[I]\geq 0)$ or last $(V[I]<0)$ elements of coordinate I $\quad 2\ 3\uparrow X \leftrightarrow ABC$ / EFG $\quad {}^{-}2\uparrow P \leftrightarrow 5\ 7$
Drop	$V\downarrow A$	
Grade up[3,5]	$\grave{A}A$	The permutation which would order A (ascending or descending) $\quad \grave{A}3\ 5\ 3\ 2 \leftrightarrow 4\ 1\ 3\ 2$ $\quad \acute{A}3\ 5\ 3\ 2 \leftrightarrow 2\ 1\ 3\ 4$
Grade down[3,5]	$\acute{A}A$	
Compress[5]	V/A	$1\ 0\ 1\ 0/P \leftrightarrow 2\ 5$ $\quad 1\ 0\ 1\ 0/E \leftrightarrow$ $1\ 3$ / $5\ 7$ / $9\ 11$ $\quad 1\ 0\ 1/[1]E \leftrightarrow$ $1\ 2\ 3\ 4$ / $9\ 10\ 11\ 12$ $\quad 1\ 0\ 1\neq E$
Expand[5]	$V\backslash A$	$1\ 0\ 1\backslash\iota 2 \leftrightarrow 1\ 0\ 2$ $\quad 1\ 0\ 1\ 1\ 1\backslash X \leftrightarrow$ A BCD / E FGH / I JKL
Reverse[5]	ϕA	$\phi X \leftrightarrow$ DCBA / HGFE / LKJI $\quad \phi[1]X \leftrightarrow \ominus X \leftrightarrow$ IJKL / EFGH / ABCD $\quad \phi P \leftrightarrow 7\ 5\ 3\ 2$
Rotate[5]	$A\phi A$	$3\phi P \leftrightarrow 7\ 2\ 3\ 5 \leftrightarrow {}^{-}1\phi P$ $\quad 1\ 0\ {}^{-}1\phi X \leftrightarrow$ BCDA / EFGH / LIJK
Transpose	$V\mathbin{\text{⍉}}A$	Coordinate I of A becomes coordinate $V[I]$ of result $\quad 1\ 1\mathbin{\text{⍉}}E \leftrightarrow 1\ 6\ 11$ $\quad 2\ 1\mathbin{\text{⍉}}X \leftrightarrow$ AEI / BFJ / CGK / DHL
	$\mathbin{\text{⍉}}A$	Transpose last two coordinates $\quad \mathbin{\text{⍉}}E \leftrightarrow 2\ 1\mathbin{\text{⍉}}E$
Membership	$A\epsilon A$	$\rho W\epsilon Y \leftrightarrow \rho W$ $\quad P\epsilon\iota 4 \leftrightarrow 1\ 1\ 0\ 0$ $\quad E\epsilon P \leftrightarrow$ $0\ 1\ 1\ 0$ / $1\ 0\ 1\ 0$ / $0\ 0\ 0\ 0$
Decode	$V\perp V$	$10\perp 1\ 7\ 7\ 6 \leftrightarrow 1776$ $\quad 24\ 60\ 60\perp 1\ 2\ 3 \leftrightarrow 3723$
Encode	$V\top S$	$24\ 60\ 60\top 3723 \leftrightarrow 1\ 2\ 3$ $\quad 60\ 60\top 3723 \leftrightarrow 2\ 3$
Deal[3]	$S?S$	$W?Y \leftrightarrow$ Random deal of W elements from ιY

Notes:

1 Restrictions on argument ranks are indicated by: S for scalar, V for vector, M for matrix, A for Any. Except as the first argument of $S\iota A$ or $S[A]$, a scalar may be used instead of a vector. A one-element array may replace any scalar.

2 Arrays used in examples: $P \leftrightarrow 2\ 3\ 5\ 7$ $\quad E \leftrightarrow$ $1\ 2\ 3\ 4$ / $5\ 6\ 7\ 8$ / $9\ 10\ 11\ 12$ $\quad X \leftrightarrow$ ABCD / EFGH / IJKL

3 Function depends on index origin.

4 Elision of any index selects all along that coordinate.

5 The function is applied along the last coordinate; the symbols $\neq$, $\backslash$, and $\ominus$ are equivalent to $/$, $\backslash$, and ϕ, respectively, except that the function is applied along the first coordinate. If $[S]$ appears after any of the symbols, the relevant coordinate is determined by the scalar S.

APL primitive scalar functions

Monadic form fB		f	Dyadic form AfB	
Definition or example	Name		Name	Definition or example
$+B \leftrightarrow 0+B$	Plus	+	Plus	2+3.2 ↔ 5.2
$-B \leftrightarrow 0-B$	Negative	-	Minus	2-3.2 ↔ ¯1.2
$\times B \leftrightarrow (B>0)-(B<0)$	Signum	×	Times	2×3.2 ↔ 6.4
$\div B \leftrightarrow 1\div B$	Reciprocal	÷	Divide	2÷3.2 ↔ 0.625
B ⌈B ⌊B 3.14 4 3 ¯3.14 ¯3 ¯4	Ceiling	⌈	Maximum	3⌈7 ↔ 7
	Floor	⌊	Minimum	3⌊7 ↔ 3
$\star B \leftrightarrow (2.71828..)\star B$	Exponential	⋆	Power	2⋆3 ↔ 8
$\circledast\star N \leftrightarrow N \leftrightarrow \star\circledast N$	Natural logarithm	⍟	Logarithm	$A\circledast B \leftrightarrow$ Log B base A $A\circledast B \leftrightarrow (\circledast B)\div\circledast A$
$\lvert$¯3.14 ↔ 3.14	Magnitude	|	Residue	Case | $A \mid B$ $A\neq 0$ | $B-(\lvert A)\times\lfloor B\div\lvert A$ $A=0, B\geq 0$ | B $A=0, B<0$ | Domain error
!0 ↔ 1 $!B \leftrightarrow B\times!B-1$ or $!B \leftrightarrow$ Gamma$(B+1)$	Factorial	!	Binomial coefficient	$A!B \leftrightarrow (!B)\div(!A)\times!B-A$ 2!5 ↔ 10 3!5 ↔ 10
$?B \leftrightarrow$ Random choice from ιB	Roll	?	Deal	A Mixed Function
$\circ B \leftrightarrow B\times 3.14159...$	Pi times	○	Circular	See Table at left
~1 ↔ 0 ~0 ↔1	Not	~		
		∧	And	A | B | $A\wedge B$ | $A\vee B$ | $A\,\tilde{\wedge}\,B$ | $A\,\tilde{\vee}\,B$ 0 | 0 | 0 | 0 | 1 | 1 0 | 1 | 0 | 1 | 1 | 0 1 | 0 | 0 | 1 | 1 | 0 1 | 1 | 1 | 1 | 0 | 0
		∨	Or	
		⍲	Nand	
		⍱	Nor	
		<	Less	Relations
		≤	Not greater	Result is 1 if the relation holds, 0 if it does not: 3≤7 ↔ 1 7≤3 ↔ 0
		=	Equal	
		≥	Not less	
		>	Greater	
		≠	Not Equal	

$(-A)\circ B$	A	$A\circ B$
$(1-B\star 2)\star .5$	0	$(1-B\star 2)\star .5$
Arcsin B	1	Sine B
Arccos B	2	Cosine B
Arctan B	3	Tangent B
$(\bar{}1+B\star 2)\star .5$	4	$(1+B\star 2)\star .5$
Arcsinh B	5	Sinh B
Arccosh B	6	Cosh B
Arctanh B	7	Tanh B

Table of Dyadic ○ Functions