1981

Array Facilities in Programming Languages

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
81-380

ARRAY FACILITIES IN PROGRAMMING LANGUAGES

John R. Rice
Computer Science
Purdue University

CSD-TR 380

10 October 1981

ABSTRACT

This report presents some general principles for incorporating facilities for array processing languages. The discussion is not intended to cover all aspects, for example, almost nothing is said about facilities for building or subsetting arrays. The current Fortran proposal is then discussed as it relates to these principles; some substantial conflicts are noted. Finally, there is a set of 11 problems to "test" specific proposals; these are stated in general, then programmed in a "natural" (artificial) language and in the language proposed by the X3J3 standards committee for the next Fortran.

Array Facilities in Programming Languages

OBJECTIVE

This note discusses how facilities for array processing should be incorporated into programming languages. It is assumed that one is considering:

(a) extending an existing language by adding array processing

(b) designing a language which at some future time may be extended to add new array facilities.

This is a first draft intended to serve as a basis for discussion and further development.

GENERAL PRINCIPLES

1. Arrays are data structures to which both element by element operators and aggregate operators may be applied.
2. The storage allocation for arrays is disassociated from the working size of arrays.
3. The facilities will not conflict with natural extensions of arrays to "non-rectangular" array structures e.g. band matrices, ragged tables. Similarly, array operators (e.g. Σ, transpose) will not conflict with natural extensions.
4. Parallel operations on array elements are specified by special operators and procedures. Aggregate operators (e.g. Σ, matrix multiply) need not have special operators and procedures for parallel operation.

5. Parallel and aggregate operators or procedures will not conflict with masking. For example, the language will allow natural extensions for the following statements:

   (a) $a_{ij} = 1/b_{ij}$ for all i,j where $b_{ij} \neq 0$

   (b) $x = \sum_{\substack{i,j=1 \\ i \neq j}}^{N} |a_{ij}|$

6. Array processing needs the support of numerous procedures for processing and manipulation. Manipulation includes facilities for identifying and selecting subarrays. The language will not restrict the introduction of such facilities.

NOTATION, TERMINOLOGY AND DISCUSSION

Arrays A,B,X, etc. have elements a_{ij}, b_{mn}, x_k, etc. The range variables (indices) have values between an upper and lower limit denoted by RANGE_HI and RANGE_LO. If the language involves explicit storage allocation for arrays then the bounds on the range limits are denoted by BOUND_LO and BOUND_HI so that the jth index i of the array A satisfies

$$\text{BOUND\_LO}(A,j) \leq \text{RANGE\_LO}(A,j) \leq i \leq \text{RANGE\_HI}(A,j) \leq \text{BOUND\_HI}(A,j)$$

Operators may be applied to array elements (e.g. $|a_{ij}|$ in the absolute value of the i,j element of array A of numbers) or to arrays as aggregates (e.g. A*B is the "product" of the arrays A and B which are numerical arrays interpreted as matrices). The language can, of course, choose syntax completely different from the usual notations,

but such choices are unnatural. There are numerous operators which apply to all the elements of aggregates such as arrays (e.g. Σ, max,$\cup$). I identify two sets for such operators: the range set R and the mask set M. For example, one has

$$\sum_{\substack{(i,j)\in R \\ \text{and} \quad (i,j)\in M}} a_{ij}$$

For arrays the range set R is determined by range limits RANGE_LO and RANGE_HI for each index. The mask set is in general an arbitrary set, in practice it is usually defined in terms of relational and/or logical operators applied to the indices or elements of the array. For example

$$\sum_{\substack{i,j=1 \\ i\neq j}}^{N} a_{ij}, \qquad \sum_{\substack{i=-k \\ x_i\neq 0}}^{k} 1/x_i, \qquad \max_{\substack{-10\le i,j\le 20 \\ x_i^2+y_j^2\ge 50}} f(x_i-y_i)/g(x_i+y_i)$$

I use $\Leftarrow$ to denote parallel assignment for elements of arrays. The parallel assignment operator is similar to the usual Σ and max in that there is both a range set R and mask set M. Thus

$$a_{ij} \Leftarrow f(i,j) \qquad (i,j)\in R,\ (i,j)\in M$$

means that f(i,j) is computed for all indices within the range set R and the mask set M. Then the values are assigned, by element, to the corresponding elements of A if the range set R is not given then it defaults to the range of A. There is no implied order in the assignment or the evaluation of f and, of course, f(i,j) must be well defined for all i,j.

Most special support procedures are implemented as functions because there are no standard symbols for them. For all arrays one expects to find functions to determine simple attributes such as range limits and bounds and the number of dimensions. Functions expected for numerical arrays include Σ, Π, max, min, masked versions of these and functions to provide the indices (locations) of maxima and minima. Such functions can have expressions as arguments. Functions expected for logical arrays include ANY, ALL, COUNT and masked versions of these.

Facilities for subarray selection may be of several natural types. The parallel assignment statement is natural in many contexts e.g.

$$\text{ASUBK}_{i,j} \Leftarrow a_{i+k,j+k} \qquad 1 \leq i,j \leq m-k$$

$$d_i \Leftarrow b_{i,i,i}$$

Functions are natural when there are common names for the selection to be made, e.g. ROW(A,j) for the jth row of A, DIAGONAL(B) for the diagonal of B.

The most usual facility for array construction is the parallel assignment statement, e.g.

$$\text{SEQ}_i \Leftarrow i$$

$$\text{EN}_i \Leftarrow 1 \qquad i \neq N$$

$$\text{ident}_{i,j} \Leftarrow 0$$

$$\text{ident}_{i,j} \Leftarrow 1 \qquad i=j$$

Array constants are included in this context as follows (The index i is given merely to emphasize that we are defining arrays; it would be omitted in most languages.

$SEQ_i \Leftarrow (1,2,3,4,5,6,7,8)$

$TWON_i \Leftarrow (1,2,4,16,32,64)$

$N33_{ij} \Leftarrow$ (1,2,3)
(4,5,6)
(7,8,9)

The ordinary assignment operator = without subscripts on the left might be more appropriate for the assignment of array constants. More elaborate assignments are natural if adequately supported by the user interface e.g.

```
  A = COL1, COL2, COL3, COL4
BIG = (A I ZERO)
      (I B I)
      (ZERO I C)
```

Arrays are to be passed as arguments to procedures as a simple aggregate of elements. The nature of the aggregate (index ranges and bound, dimensions) is an intrinsic part of the information and included with the values when the array is an argument.

APPLICATION TO FORTRAN

These notes are motivated by the current effort to produce a new Fortran standard called Fortran 8X. It is widely acknowledged that a broad segment of the Fortran user community wants array facilities added to Fortran. A draft proposal (S6.78, 1 June 1981, pages 3-1 to 3-34) uses a considerably different approach which we call the S6 proposal. I apply the above principles and ideas to provide facilities essentially similar to the S6 proposal within the Fortran context. I

believe this new approach is simpler, more natural and does not interfer with further evolution of Fortran which might occur within its "application modules" or with some new standardization. Furthermore, this approach allows one to introduce matrix/vector arithmetic into Fortran without perturbing the language even though this was not one of my principal objectives.

There are two main differences between our approach and the S6 proposal which we discuss first. I propose to use indices (or range variables). I view the omission of indices from the S6 proposal as simply a mistake. It is essential that a programmer be aware of the working size or actual size of any arrays in his program. He must do that whether or not the language has facilities to help him. Storage allocation for arrays is prominent in Fortran (and many other languages) but we can hope that this aspect of programming languages will disappear in the future. In the meantime, many (if not the majority) of the Fortran programmers confuse working size and storage allocation for arrays. I know several very sophisticated and experienced Fortran programmers who do not understand the array mechanisms of Fortran. The process of passing variable dimensional arrays in Fortran is black magic for the vast majority of the Fortran programmers. The introduction of indices into Fortran would clarify many programs and be a first step in the natural transition to the time when storage allocation for arrays will virtually disappear.

The second big difference between our proposal and the S6 proposal is the parallel assignment operator. There is no obvious way to introduce this operator which is both natural and nicely compatible with Fortran. However, the WHERE facility of the S6 proposal

is a special case of the parallel assignment statement so some such operator is already planned. Recall that the simple WHERE statement in the S6 proposal is

WHERE(MASK) A = expression

where MASK is a logical array of the same size as A. In the previous, more general notation, this is

$$a_{ij} \Leftarrow \text{expression } (i,j) \text{ for } (i,j) \in S,\ (i,j) \in M$$

The set M is where $MASK_{ij}$ is .TRUE. and the set S is the storage allocation bounds on A. That is the assignment takes place for

BOUND_LO(A,1) $\leq$ i $\leq$ BOUND_HI(A,1)

instead of

RANGE_LO(A,1) $\leq$ i $\leq$ RANGE_HI(A,1)

This is very inflexible and a severe weakness in the S6 proposal.

There seems to be three ways to introduce the parallel assignment statement into Fortran:

(1) New, one or two character assignment operator: This is used in the previous discussion, its disadvantages are (a) a special character is used up, (b) specification of the range and mask is awkward. Even if the range of the assignment is automatically the range of the array, there is still a problem with the mask.

(2) New keyword statement: This is the WHERE approach and it is within the Fortran tradition (e.g. PRINT, READ, GOTO, IF). The word WHERE is not very appropriate; ARRAY, ASSIGN and PARALLEL are better.

Examples:

PARALLEL A(I,J) = B(I,J) + I-J

ASSIGN (CONDITION_4) X(I) = F(I,B)*COS(I*PI)

ARRAY (A(I,I) .GT. 1.) B(I,J) = I*J/COSH(A(I,I))

(3) Intrinsic Function or Subprogram: This appears to be unnatural e.g.

CALL PARALLEL (A,I,J, expression)

SPECIFIC PROPOSALS FOR FORTRAN 8X

There is no discussion of sections, packing, I/O and IDENTIFY as these topics are somewhat orthogonal to the proposals presented here.

1. Declarations

DIMENSION A(10, 10), X(10)

All defaults are taken and there are no explicit indices for A and X. We have, for example,

BOUND_LO(A,1) = RANGE_LO(A,1)=1, RANGE_HI(A,1) = BOUND_HI(A,1)=10

DIMENSION A(N=10, N=10), X(NLOW = -2: NHIGH=7)

Defaults are taken on the lower ranges and bounds of A. A is an N by N array with N $\leq$ 10. We have

BOUND_LO(X,1) = -2, BOUND_HI(X,1)=7

RANGE_LO(X,1) = NLOW, RANGE_HI(X,1) = NHIGH

and (NLOW, NHIGH) is initialized to (-2, 7).

2. Arrays as Arguments. For purposes of discussion, I present a simple program involving arrays.

```
DIMENSION A(N≤100, N≤100), X(N≤100), B(N≤100), RES2D(N≤100)
A(I,J) <= 1./(I+J-1.)
READ N,B
PRINT N,B
CALL LINEAR_EQ _SOLVER(A,X,B)
RESID = A*X-B
RESID_MAX = MAXVAL(ABS(RESID(I)); I=1 TO N)
PRINT X, 'MAX RESIDUAL IS' RESID_MAX
```

The size of the linear system is N, determined by the READ statement. The subroutine LINEAR_EQ_SOLVER knows the value of N as it is passed with each of the arrays. Note that this subroutine can check if the problem is well defined (i.e. that the arrays are compatible). Other aspects of this little program are discussed later.

3. Assignment of Arrays. We use the keyword ASSIGN here. The little program in 2. above has A(I,J) set to 1/(I+J-1) for the full 100 by 100 array. The assignment of RESID is an ordinary assignment, the aggregate RESID is assigned the value computed on the right. The more general masked parallel assignment is illustrated by

```
ASSIGN(A(I,J) .GE. EPS) B(I,J) = C(I,J-2)/A(I,J)
```

The values of B(I,J) are assigned for

$$\text{RANGE\_LO}(B,1) \leq I \leq \text{RANGE\_HI}(B,1)$$

$$\text{RANGE\_LO}(B,2) \leq J \leq \text{RANGE\_HI}(B,2)$$

and A(I,J) .GE. EPS = .TRUE. The right side and A must be compatible with B, in the sense that the ranges (not storage declarations) are large enough so the range of B does not involve out-of-range values from A or the right side.

In particular, we must have

RANGE_LO(A,1) $\leq$ RANGE_LO(B,1)

RANGE_LO(C,2) $\leq$ RANGE_LO(B,2)-2

RANGE_HI(C,2) $\leq$ RANGE_HI(B,2)-2

Note that side-effects of functions on the right must be strictly limited so that the result is independent of how the right side is evaluated.

4. Standard Operators Applied to Arrays. The arithmetic operators applied to numeric arrays of dimension 2 or less are the operators of linear algebra. Those included in Fortran 8X are

*, +, -, '	for numeric arrays (' is for transpose)
.NE., etc.	for numeric arrays
.AND., etc.	for logical arrays

The compatibility conditions are those of linear algebra (in terms of ranges). I feel that the broadcasting of constants should not be allowed (i.e. 2.*A is allowed but A+2. is not), but I do not object strongly.

Note: One may claim that allowing A .LE. B to be done for two numeric arrays in aggregate forms contradicts the stated principle that element by element operations are done explicitly on elements. I reply that A .LE. B is standardly defined as A(I,J) $\leq$ B(I,J) for all I,J so A .LE. B has a single logical value.

Further, one may legitimately claim that allowing A .AND. B to be done for two logical arrays in aggregate form contradicts the stated principle. I reply that (as far as I know) there are no standard operators for logical arrays and thus I am free to define A .AND. B this way.

5. Standard Aggregate Operators Applied to Arrays. The most

perplexing problem is how to give Σ and related operators a modest functionality (as done in the S6 proposal) without either introducing completely new syntatic constructions or incompatibilities with doing it right in the future. Recall that the future requires the construction

SUM(expression in I, I ε Range set, I ε Mask set)

The natural way to do this in Fortran is to use

SUM(F(I); FOR(I = I1, I2, I3))

MASK_SUM(F(I); FOR(I = I1, I2, I3), MASK_SET)

The S6 proposal is flawed both by being inflexible and, worse, being incompatible with doing it right later.

If one cannot see the way clear to provide a reasonable mechanism for the range set in SUM, then a limited capability similar to that of the S6 proposal should be included, i.e. SUM(A) has default range over the indices of A. To SUM an expression the S6 proposal requires two statements (plus one declaration) as follows:

```
DIMENSION  TEMP_SUM (I1 ≤ 100, J1 ≤ 100)
ASSIGN  TEMP_SUM(I,J) = EXPRESSION(I,J)
SUMA = SUM(TEMP_SUM)
```

6. Library Functions to Support Arrays. We note that the parallel assignment statements greatly reduces the need for the "shifting" functions in the S6 proposal. One can, for example shift an array two units "down" by

ASSIGN A(I,J) = A(I-2, J-2)

These functions are classified into four groups below (except for ALT and PROJECT of the S6 proposal whose function is not immediately apparent).

A. Essentially unaffected by changes proposed here:

MERGE	FIRSTLOC
DETERMINANT (Consider deleting as useless)	LASTLOC
COUNT	RANK (don't like the name)
ANY	EXTENT
ALL	SIZE (not very worthwhile)
LBOUND	UBOUND
DIAGONAL	

B. Changed semantics or syntax:

SUM	MASK_SUM
PRODUCT	MASK_PRODUCT
MAXVAL	MASK_MAXVAL
MINVAL	MASK_MINVAL

C. Candidates for deletion (* means certain)

INVERSE	SPREAD
DOTPRODUCT*	REPLICATE
MATMULT*	CSHIFT
SEQ	EOSHIFT

D. New library functions:

URANGE, LRANGE	(provide index range information)
ROW, COLUMN	(select rows and columns of 2D arrays)

7. Array Facilities in Core Fortran. I believe that core Fortran should have SUM, MAX and MIN operators. Unfortunately, these are currently linked tightly to arrays.

There appear to be three logical places to "cut"

(a) Have no array facilities at all

(b) Have aggregate operators only (i.e. matrix/vector arithmetic) plus SUM, MAXVAL, MINVAL

(c) Have parallel assignment only, plus SUM, MAXVAL, MINVAL

The most generally useful choice is (c). The choice of whether to include indices in the core is orthogonal to the choice between (b) and (c); it is analogous to saying whether LEN (for characters) is to be included in the core.

EXAMPLES TO TEST FORTRAN ARRAY FACILITY IDEAS

These examples are presented in three forms. The first is an English/mathematical statement of the calculation to be made. The second, labeled NATURAL, is a form which, in my opinion, is natural for a Fortran-like programming language. By and large, this form is self-explanatory if one knows Fortran plus the concepts of range variable and parallel assignment (I use the keyword ASSIGN here). The third form is that of the X3J3/S6 proposal (Summer 1981). The first five examples are taken from Alan Wilson's draft report of 14 September 1981. They tend to show the X3J3/S6 proposal works well. The remainder are examples that tend to show that the X3J3/S6 proposal works poorly in some important respects. The new examples are intended to be representative of common and important computations.

Declarations are given only when that seems relevant. Keep in mind that the X3J3/S6 proposal requires the arrays involved have a length fixed permanently as used in the program segments.

Example 1: Evaluate the trapezoidal rule estimate of an integral

$$T_N = h*(1/2\ f(a) + \sum_{i=1}^{N-1} f(a+ih) + 1/2\ f(b))$$

NATURAL:

```
T_N = H*((F(A)+F(B))/2. + SUM(F(A+I*H); FOR(I=1,N-1))
```

X3J3/S6:

```
T_N = F(A)/2.
T_N = T_N + SUM(F(A+H*SEQ(1,N-1)))
T_N = H*(F(B)/2. + T_N)
```

improved

```
T_N = H*((F(A)+F(B))/2. + SUM(F(A+H*SEQ(1,N-1)))
```

Example 2: Compute the value of

$$S = \sum_{j=1}^{n} \prod_{i=1}^{m} a_{ij}$$

NATURAL: S = SUM(PRODUCT(A(I,J); FOR(I=1,N)); FOR(J=1,N))

X3J3/S6: S = SUM(PRODUCT(A,1))

Example 3: Compute the value of

$$R = \sum_{\substack{i=1 \\ X_i \neq 0}}^{n} \frac{1}{X_i}$$

NATURAL: R = MASK_SUM(1./X(I); FOR(I=1,N); X(I).NE.0)

alternative

R = SUM(1./X(I); FOR(I=1,N); MASK(X(I).NE.0))

X3J3/S6: R = MASK_SUM(1./X, X.NE.0.)

Example 4: One has a table t_{ij} of the i-th student's score on the j-th test. One is to

(a) list the top score for each student = top_i

(b) give the number of scores above the average = NABOVE

(c) increase the above average scores by 10%

(d) give the lowest score that is above average = LOW_ABOVE

(e) say where any student has all scores above average = GENIUS

NATURAL:
```
REAL T(M≤100, N≤40), TOP(M≤100)
LOGICAL ABOVE(M≤100, N≤40), GENIUS
READ N,M,T
ASSIGN TOP(I) = MAX(T(I,J); FOR(J=1,M))

AVE_SCORE = SUM(T(I,J); FOR(I=1,M); FOR(J=1,N))/(N*M)
ASSIGN ABOVE(I,J) = T(I,J) .GE. AVE_SCORE
NABOVE = COUNT(ABOVE(I,J); FOR(I=1,M); FOR(J=1,N))
MASK_ASSIGN(ABOVE(I,J))T(I,J) = 1.1 * T(I,J)

LOW_ABOVE = MASK_MIN(T(I,J); FOR(I=1,M); FOR(J=1,N); ABOVE(I,J))
GENIUS = ANY(ALL(ABOVE(I,J); FOR(J=1,M)); FOR(I=1,M))

PRINT TOP, AVE_SCORE, NABOVE, LOW_ABOVE, GENIUS
```

Note: In a "real" language, I would consider the special construction

```
SUM(array)
```

to be equivalent to

```
SUM(array(1,2,...,K); FOR(I=RL(array,1), RH(array,1)),...,
                      FOR(K=RL(array,last), RH(array,last)))
```

using range values RL and RH as defaults for the DO-loops. This is, for example, now done in the Fortran PRINT statement using bound values rather than range values.

X3J3/S6:
```
REAL(72,21), TOP(73)
LOGICAL ABOVE(73,21), GENIUS
READ T
TOP = MAXVAL(T,2)
I don't see an easy way to compute AVE_SCORE
ABOVE = (T.GE.SUM(T))/SIZE(T)
NABOVE = COUNT(ABOVE)
WHERE(ABOVE) T = 1.1*T
LOW_ABOVE = MASK_MINVAL (T,ABOVE)
GENIUS = ANY(ALL(ABOVE,2))
PRINT TOP, AVE_SCORE, NABOVE, LOW_ABOVE, GENIUS
```

Example 5: Solve the tridiagonal system Tx=y by a special algorithm. The matrix T is represented by L,D and U, its lower diagonal, main diagonal and upper diagonal.

NATURAL:
```
REAL  L(N≤10000); D(N≤10000); U(N≤10000); X(N≤10000), Y(N≤10000)
READ N,L,D,U,Y
K=1
DO I=1, LOG2(N)
    ASSIGN L(I) = L(I)/D(I)
    ASSIGN U(I) = U(I)/D(I)
    ASSIGN Y(I) = Y(I)/D(I)
    ASSIGN D(I) = 1. - L(I)*U(I-K) - U(I)*L(I+K)
    ASSIGN Y(I) = Y(I) - L(I)*Y(I-K) - U(I)*L(I+K)
    ASSIGN L(I) = - L(I)*L(I-K)
    ASSIGN U(I) = U(I)*U(I+K)
    K=2*K
REPEAT
ASSIGN X(I) = Y(I)/D(I)
PRINT X
```

X3J3/S6:
```
REAL L(1024), D(1024), U(1024), X(1024), Y(1024)
READ L,D,U,Y
K=1
DO I=1, LOG2(N)
    L=L/D
    U=U/D
    Y=Y/D
    D = 1 - L*EOSHIFT(U,1,-K) - U*EOSHIFT(L,1,K)
    L = - L*EOSHIFT(L,1,-K)
    U = U*EOSHIFT(U,1,K)
    K = 2*k
REPEAT
X=Y/D
PRINT X
```

Example 6: Compute the value of

$$e^* = \sum_{i=1}^{n} \prod_{j=1}^{m} (1.+e^{-|i-j|})$$

NATURAL:
```
ESTAR = SUM(PRODUCT(1.+EXP(-ABS(I-J)); FOR(J=1,M)); FOR(I=1,N))
```

X3J3/S6: I do not see how to make effective use of the X3J3/S6 array facilities.

```
REAL A(20,20)
DO I = 1,N
    DO J = 1,M
        A(I,J) = 1.+EXP(-ABS(I-J))
    REPEAT
REPEAT
ESTAR = SUM(PRODUCT(A,1))
```

A program for this in current Fortran is

```
ESTAR = 0.0
DO 20 I = 1,N
    PRODUCT = 1.0
    DO 10 J = 1,M
10      PRODUCT = PRODUCT*(1.+EXP(A-ABS(I-J))
20  ESTAR = ESTAR + PRODUCT
```

Example 7: Compute the value of the denominator constant that appears in Lagrange interpolation formulas.

$$p(x) = \sum_{i=1}^{N} f(x_i)\ \ell_i(x)$$

$$\ell_i(x) = \prod_{\substack{j=1 \\ i \neq j}}^{N} (x-x_j) \ / \ \prod_{j=1}^{N} (x_i-x_j)$$

NATURAL:
```
REAL DENOM(N<=20), X(N<=20)
ASSIGN DENOM(I) = MASK_PRODUCT(X(I)-X(J); FOR(J=1,N); I.NE.J)
```

X3J3/S6:
```
REAL DENOM(14), X(14), DUMMY(14)
DO I = 1,14
    DO J = 1,14
        IF(I.NE.J) THEN
            DUMMY(J) = X(I)-X(J)
          ELSE
            DUMMY(J) = 1.0
        ENDIF
    REPEAT
    DENOM(I) = PRODUCT(DUMMY,1)
REPEAT
```

An alternative program is

```
      REAL DENOM(14), X(14), DUMMY(14)
      LOGICAL INEJ(14)
      DO I = 1,14
          DO J = 1,14
              DUMMY(J) = X(I) - X(J)
              INEJ(J) = I.NE.J
          REPEAT
          DENOM(I) = MASK_PRODUCT(DUMMY,1,INEJ)
      REPEAT
```

A program to compute this array in the current Fortran is

```
      REAL DENOM(14), X(14)
      DO 20 I = 1,14
          PROD = 1.0
          DO 10 J = 1,14
 10           IF(I.NE.J) PROD = (X(I) - X(J))*PROD
 20       DENOM(I) = PROD
```

Example 8: The divided difference table for a set of data x_i, $y_i = f(x_i)$ is defined by the formulas

$$f[x_i] = y_i$$

$$f[x_i, x_{i+1},\ldots,x_{i+k}] = \frac{f[x_{i+1},\ldots,x_{i+k}] - f[x_i,\ldots,x_{i+k}]}{x_{i+k} - x_i}$$

The problem is to compute the first M columns of the divided difference table

$$D_{ik} = f[x_i, x_{i+1},\ldots, x_{i+k-1}]$$

NATURAL:
```
REAL D(N≤1000), M≤10), X(N≤1000), Y(N≤1000)
READ NVALUS, K,X,Y,M
N = NVALUS
ASSIGN D(I,1) = Y(I)
DO J = 2,M
    N = N-1
    ASSIGN D(L,K) = (D(L+1,K-1)-D(L,K-1))/(X(L+K-1)-X(L); FOR(K=J)
REPEAT
```

X3J3/S6:
```
REAL D(750,6), X(750), Y(750), DCOL(750)
LOGICAL MASK(750)
READ X,Y
D(*,1) = Y(*)
MASK = .TRUE.
N=750
DO J = 2,6
    N= N-1
    MASK(N) = .FALSE.
    DCOL(*) = D(*,K-1)
    WHERE(MASK) DCOL = (EOSHIFT(DCOL,1,1) - DCOL)/(EOSHIFT(X,1,J-1)-X)
    WHERE(MASK) D(*,K) = DCOL(*)
REPEAT
```

Note: I am not sure that the above program is according to the X3J3/S6 proposal.

Example 9: One has an array u_{ij} of values on an N by M grid and wants to replace each value by the average of its value plus all its neighbors. This may be expressed by

$$u_{ij} = \left(\sum_{\text{Neighbors}} u_{ij}\right)/(\text{Number of neighbors})$$

While this is a somewhat artificial example, the operations are typical of what one does in solving partial differential equations, image processing and geometric modeling.

```
NATURAL: REAL U(N≤1000, M≤1000)
         READ N,M,U
         ASSIGN U(I,J) = SUM(U(K,L); FOR(K=I-1,I+1); FOR(L=J-1,J+1))/9.;
        +                            FOR(I=2,N-1); FOR(J=2,M-1)
         ASSIGN U(1,J) = SUM(U(K,L); FOR(K=1,2); FOR(L=J-1,J+1))/6.; FOR(J=2,M-1)
         ASSIGN U(I,J) = SUM(U(K,L); FOR(K=N-1,N); FOR(L=J-1,J+1))/6.; FOR(J=2,M-1)
        +                                                            FOR(I=N)
         ASSIGN U(I,1) = SUM(U(K,L); FOR(K=I-1,I+1); FOR(L=1,2))/6.; FOR(I=2,N-1)
         ASSIGN U(I,J) = SUM(U(K,L); FOR(K=I-1,I+1); FOR(L=M-1,M))/6.; FOR(I=2,N-1);
        +                                                            FOR(J=M)
         U(1,1) = (U(1,1) + U(1,2) + U(2,2) + U(2,1))/4.
         U(N,1) = (U(N,1) + U(N,2) + U(N-1,2) + U(N-1,N-1))/4.
         U(1,M) = (U(1,M) + U(2,M) + U(1,M-1) + U(2,M-1))/4.
         U(N,M) = (U(N.M) + U(N-1,M) + U(N,M-1) + U(N-1,M-1))/4.
         PRINT U
```

alternative

```
REAL U(1≤LOWN: N≤1000, 1≤LOWM: M≤1000)
READ NU, MU, U
LOWN = LOWM=2
N = NU-1
M = MU-1
ASSIGN U(I,J) = SUM(U(K,L); FOR(K=I-1,I+1); FOR(L=J-1,J+1))/9.
ASSIGN U(1,J) = SUM(U(K,L); FOR(K=1,2); fOR(L=J-1,J+1))/6.
ASSIGN U(I,1) = SUM(U(K,L); FOR(K=I-1,I+1); FOR(L=1,2))/6.
ASSIGN U(1,J) = SUM(U(K.L); FOR(K=NU-1,NU); FOR(L=J-1,J+1))/6,; FOR(I=NU)
ASSIGN U(I,J) = SUM(U(K,L); FOR(U=I-1,I+1); FOR(L=MU-1,MU))/6.; FOR(J=MU)
U(1,1) = (U(1,1) + U(1,2) + U(2,1) + U(2,2))/4.
U(NU,1) = (U(NU,1) + U(NU,2) + U(NU-1,1) + U(NU-1,2)//4.
U(1,MU) = (U(1,MN) + U(1,MU-1) + U(2,MU) + U(2,MU-1))/4.
U(NU,MU) = (U(NU,MU) + U(NU,MU-1) + U(NU-1,MU) + U(NU-1,MU-1))/4.
PRINT U
```

X3J3/S6:

```
REAL U(600,800)
LOGICAL MASK(600,800), MASK1(600), MASK2(800)
READ U
MASK = .TRUE.
MASK(*,1) = .FALSE.
MASK(1,*) = .FALSE.
MASK(600,*) = .FALSE.
MASK(*,800) = .FALSE.
```

```
 WHERE(MASK) U=(U+EOSHIFT(U,1,1)+EOSHIFT(U,1,-1)+EOSHIFT(U,2,1)+EOSHIFT(U,2,-1)
+             + EOSHIFT(EOSHIFT(U,1,1),2,1))+EOSHIFT(EOSHIFT(U,1,-1),2,1)
+             + EOSHIFT(EOSHIFT(U,1,1),2,-1)+EOSHIFT(EOSHIFT(U,1,-1),2,-1))/9.
 MASK1 = .TRUE.
 MASK1(1) = MASK1(600) = .FALSE.
 MASK = .FALSE.
 MASK(*,1) = MASK1(*)
 WHERE(MASK) U=(U+EOSHIFT(U,1,1)+EOSHIFT(U,1,-1)+EOSHIFT(U,2,1)
+             + EOSHIFT(EOSHIFT(U,1,1),2,1)+EOSHIFT(EOSHIFT(U,1,-1),2,1))/6.
 MASK(*,1) = .FALSE.
 MASK(*,800) = MASK(*)
 WHERE(MASK) U=(U+EOSHIFT(U,1,1)+EOSHIFT(U,1,-1)+EOSHIFT(U,2,-1)
+             + EOSHIFT(EOSHIFT(U,1,1),2,-1)+EOSHIFT(EOSHIFT(U,1,-2),3-1))/6.
 MASK(*,800) = .FALSE.
 MASK2 = .TRUE.
 MASK2(1) = MASK2(800) = .FALSE.
 MASK(1,*) = MASK2(*)
 WHERE(MASK)U=(U+EOSHIFT(U,1,1)+EOSHIFT(U,2,1)+EOSHIFT(U,2,-1)
+            + EOSHIFT(EOSHIFT(U,1,1),2,-1)+EOSHIFT(EOSHIFT(U,1,1),2,1))/6.
 MASK(1,*) = .FALSE.
 MASK(600,*) = MASK2(*)
 WHERE(MASKU=(U+EOSHIFT(U,1,-1)+EOSHIFT(U,2,1)+EOSHIFT(U,2,-1)
+           + EOSHIFT(EOSHIFT(U,1,-1),2,1)+EOSHIFT(EOSHIFT(U,1,-1),2,-1))/6.
 U(1,1) = (U(1,1) + U(1,2) + U(2,1))/4.
 U(600,1) = (U(600,1) + U(600,2) + U(599,1) + U(599,2))/4.
 U(1,800) = (U(1,800) + U(2,800) + U(1,799) + U(2,799))/4.
 U(600,800) + (U(600,800) + U(600,799) + U(599,800) + U(599,799))/4.
 PRINT U
```

alternative

```
      REAL U(0:601,0:801), UIN(600,800), COUNT(0:601;0:801)
      LOGICAL MASK(0:601,0:801)
      READ UIN
      MASK = .TRUE.
          COUNT = 9.
      DO I = 1,600
          U(I,0) = U(I,801) = 0
          MASK(I,0) = MASK(I,801) = .FALSE.
          DO J = 1,800
              U(I,J) = UIN(I,J)
          REPEAT
      REPEAT
      DO J = 0,801
          U(0,J) = U(601,J)=0
          MASK(0,J) = MASK(601,J)=0
      REPEAT
      DO I = 2,599
          COUNT(I,1) = COUNT(I,799)=6.
      REPEAT
      DO J =  2,799
          COUNT(1,J) = COUNT(599,J)=6
      REPEAT
      COUNT(1,1) = COUNT(600,1) = COUNT(1,800) = COUNT(600,800)=4.
      WHERE(MASK) U=(U+EOSHIFT(U,1,1)+EOSHIFT(U,1,-1)+EOSHIFT(U,2,1)+EOSHIFT(U,2,-1)
     +             + EOSHIFT(EOSHIFT(U,1,1)2,1)+EOSHIFT(EOSHIFT(U,1,1),2,-1)
     +             + EOSHIFT(EOSHIFT(U,1,-1)+EOSHIFT(EOSHIFT(U,1,-1),2,-1))/COUNT
```

```
DO I=1,600
    DO J=1,800
        UIN(I,J) = U(I,J)
    REPEAT
REPEAT
PRINT UIN
```

Example 10: LU factorization of the N by N matrix A = a_{ij} with pivoting.

NATURAL:

```
REAL A(1≤NLO: N≤1000; 1≤NLO: N≤1000), TEMP(1≤NLO: N≤1000)
READ N,A
DO NLO = 1,N
    COLMAX = MAX(A(IM,NLO); FOR(IM=NLO,N))
    NLOSAVE = NLO
    NLO = 1
    TEMP(*) = A(*,NLO)
    A(*,NLO) = A(*,IM)
    A(*,IM) = TEMP(*)
    NLO = NLOSAVE
    DO NROW = NLO+1,N
        A(NROW,NLO) = A(NROW,NLO)/COLMAX
        ASSIGN A(I,J) = A(I,J) - A(NROW,NLO)*A(NLO,J); FOR(I=NROW)
    REPEAT
REPEAT
NLO = 1
PRINT A
```

X3J3/S6:

```
REAL A(640,640), TEMP(640), TEMP2(640)
LOGICAL MASK(640,640), MASK1(640)
MASK = .TRUE.
MASK1 = .TRUE.
DO NLO = 1,640
    TEMP(*) = A(NLO,*)
    COLMAX = MASK_MAXVAL(TEMP,MASK1)
    IM = FIRSTLOC(TEMP, TEMP .EQ. COLMAX)
    MASK1(NLO) = .FALSE.
    WHERE(MASK) TEMP = TEMP/COLMAX
    TEMP(*) = A(*,IM)
    A(*,IM) = A(*,NLO)
    A(*,NLO) = TEMP(*)
    MASK(NLO,*) = .FALSE.
    MASK(*,NLO) = .FALSE.
    DO NROW = NLO + 1,640
        TEMP2(*) = A(NROW,*)
        WHERE(MASK1) TEMP2= TEMP2- A(NROW,NLO) * TEMP2
        A(NROW,*) = TEMP2(*)
    REPEAT
    PRINT A
```

Example 11: Tead successive sets of data, trim the negative and large values off, do a logarithmic transformation, compute the first four Fourier moments then save these moments and the data ID in a data base. The processing stops when a data set has an ID of zero.

NATURAL:

```
     REAL DATA(NPTS≤5000), COS_WT(NPTS≤5000), FMOMENTS(4)
   1 READ ID, NPTS, DATA
     IF(ID .EQ. 0) STOP
     ASSIGN(DATA(I).LE.0) DATA(I)=0.0
     ASSIGN(DATA(I).GE.1000) DATA(I) = 1000.0
     ASSIGN DATA(I) = LOG(DATA(I))
     FMOMENTS(1) = SUM(DATA(I); FOR(I=1,NPTS))/NPTS
     DO K=2,4
          ASSIGN COS_WT(I) = COS(PI*I/(NPTS+1))
          FMOMENTS(K) = SUM(DATA(I)*COS_WT(I); FOR(I=1,NPTS))/NPTS
     REPEAT
     CALL SAVE_MOMENTS(ID,FMOMENTS)
     GO TO 1
```

alternative: if one has vector products then the FMOMENTS computations can be made by

```
FMOMENTS(K) = (DATA * COS_WT)/NPTS
```

X3J3/S6: I see no direct way to write this program and exploit vector processing in a reasonable way. Two "tricks" that one could apply are

(A) Create a MASK for the data and mask all the vector operations to ignore the part of the vector not filled with data. This means that all runs will compute with vectors of length 5000, even if the data has only 100 points.

(B) Make the program a dummy and have everything done via subroutine calls. This assumes that subroutines can start as

```
SUBROUTINE SUB(A, N, ANS)
REAL A(N), ANS(N)
```

and that vector operations will then have range N upon execution. I do not know if this actually works with the X3J3/S6 proposal.