Forth Advanced Scientific Tools: How Intelligent Should Arrays Be?

Ferren MacIntyre

Center for Atmospheric-Chemistry Studies, Graduate School of Oceanography
University of Rhode Island, Narragansett RI 02882-1197

Abstract: FAST, the Forth Advanced Scientific Tools package, should treat arrays as entities and not as collections. The question is how much information DATATYPE== (the grandparent word which creates data-type-defining words in the proposed Forth Floating-Point Standard Extension) should store in an array header, and how array operators should use the information. As megabyte memories and 10 MHz speeds become the PC norm, there are applications in which arrays should remember dimensions, data type, address type, and storage byte-width.

DATATYPE== is the grandfather word (hence two "=" signs appended) proposed for the Forth Floating-Point Standard Extension (MacIntyre and Dowling, 1986; henceforth MAD). It is used to create defining words for some 144 data types, all of which, at the lowest level of machine intelligence, are QUANs , fetched by name and stored by IS , (eliminating many special ! words). DATATYPE== expects four parameters on the stack: d = 0 1 2 3, the number of dimensions; w = the number of bytes per storage location; t = 0 1 2 3 4, the data type (integer, floating point, complex, BCD, string); and a = 0 1 2, the location of the stored data (after the header, elsewhere, outside of Forth's 64k segment). To create a word to define 3-dimensional floating-point complex arrays with 8 bytes per storage location, indirectly stored at a point determined after compilation (for block-moving to disc free of link addresses), one might write:

```
3 8 2 1 DATATYPE== 3DCIARRAY=
```

(MAD neglected the convention of ending creating words with "=", omitted strings as a data type, and parametrized the byte-width where it now seems better to use the actual count. Also, the order of parameters suggested in MAD may not be optimal. Time will tell which is the most useful approach.)

In passing, MAD noted that DATATYPE== might store all dimensions (one more than needed for subscripting) so that matrix-manipulation routines would have this information. I now suggest storage of the above parameters also.

Forth has traditionally abjured, e.g., bounds checking, in favor of efficiency, honoring slow, expensive hardware and fast, cheap progammers. With costs and time reversed, it may make sense to rethink these priorities in some cases. For instance, by explicitly writing J C * I + <name> + @ , one could eliminate the extra storage for dimensions and the overhead of subscripting calculations, but repetition of I J <name> more than pays for this overhead. At the next level of user friendliness, indexing itself is unnecessary because the array should be treated as an entity, even at the cost of additional overhead.

In conjunction with Forth Advanced Scientific Tools (FAST Workshop, Kelly and MacIntyre, this Conference), I have experimented with smart arrays to minimize effort in using array-manipulating words, such as those in Startz's (1985) "cookbook". User-friendliness costs initial programming and memory, but need not slow critical operations (MacIntyre et al. 1986). As an example, I presents below some words based on Startz's cookbook which work indiscriminately and without user attention for 18 integer data types: 1-, 2 , and 3-dimensional arrays, 1, 2, and 4 bytes wide, stored at the header or elsewhere.

Noting that "more scientific computation time is spent [on] inner products than on

any other single problem", Startz half-heartedly begins to treat arrays as entities, but the link to BASIC lumbers the programmer with inserting ancillary information in subroutine calls. For instance, his inner-product routine is called by

```
CALL GINPROD(A(IROW,0),B(0,JCOL),SUM,ITYPE,ITYPE,I1,I1,M)
```

where IROW and JCOL are explicit in external BASIC loops. SUM is the answer upon return, ITYPE is the array type, I1 is the byte width, and M is the number of elements.

Since the silicon knew most of the parameters in the above call at one time, we should be able to replace the above 12-parameter call with

```
A B C INPROD
```

where C is a destination array and we have also eliminated the explicit loops.

If <name> gets the PFA, as it would here, indexing would be via I J K <name> ENTRY , where ENTRY consists of all but the first line of the ;CODE portion of AR= (261:8-14). We temporize in the examples below, using ' <name> to get the PFA (269:10), and indexing in the normal manner. Such routines should use the type-conversion of IEEE-754 hardware, and need to know size and byte length. If the programmer foils the machine with incompatible arrays, it should apologize and explain.

These tasks are not difficult to automate, but the appended MMSFORTH 8088 assembly code is far from definitive, being a chemist's first experiment with smart arrays. If this approach appeals to Forth number crunchers, it might be well to seek joint decisions about intelligent operators before we fragment into idiosyncratic camps, all of which are correct--and incompatible.

EXAMPLES OF SMART ARRAY USAGE

As a step toward DATATYPE== , the integer array-maker AR= (261:6) expects 1 to 3 dimensions, w, d, and a on the stack (263:2-6). If a=0, data is stored after the header; if a=1, data is stored at OBASE OP + . The highly experimental words RDO CDO PDO (262:6-8), take the PFA of an array on the stack and initiate DO loops of the appropriate size on rows, columns, and pages, as at 263:12-15.

APARAM and ACOMP put array parameters on the stack and compare them, respectively, calling .QUIT if arrays are incompatible. RAMP , A! , and A+ fill an array with a ramp, copy it, and add two arrays, as examples of functions of 1, 2, and 3 arrays. After the first example is tested, writing related functions is simpler.

The use of memory for temporary storage where other languages use the stack (266:2), and the unForthlike use of the stack for static indexed storage (268:6-8) with clearing upon return (269:6), may not be necessary, since I was mostly copying Startz' model. Still, the 8088 architecture is far from optimal for dual-stack operation or array processing, and I certainly would not object to finding three memory-address registers, ports, and indices (2 sources and a destination) on a Forth engine.

REFERENCES

MacIntyre, F., and Dowling, T., 1986. User-Oriented Suggestions for Floating-Point and Complex-Arithmetic Forth Standard Extensions, J. Forth Applic. Res. 4:

MacIntyre, F., Estep, K., and Sieburth, J. McN., (Submitted). The Cost of User-Friendly Programming: MacImage as Example, J. Forth Applic. Res.

Startz, R., 1985. 8087 Applications and Programming, 2nd ed. (Brady, NY).

```
Smart Arrays; 12\05\86 (16:56:08) 2.4-GWORAK-860220    Page 1

Block 260 [65 :1]

\ 860417 FM 01/10 Array-maker. Dimensional offsets.
: TASK        ." Array-maker" TASK ;
QUAN OP  0 IS OP  QUAN OBASE  0 IS OBASE
VARIABLE !BP   VARIABLE !SI   VARIABLE !TP
CODE K   8 [+BP] PUSH  NEXT                    \ 3rd loop index
CODE CC@ ( '->b b) BX POP [BX] AX MOV AH DL MOV \ Fetch 2 bytes
    AH AH XOR DH DH XOR PSH2                   \ from address.
\ Calculate dimensional offsets.
: DIM2  ( rc ' w->' o) )R )R R@ 6 +  C@          ¥ +! R) R) ;
: DIM3 ( rcp ' w->' o) )R )R R@ 6 + CC@ ROT ¥ ROT + ¥ + R) R) ;

: !DIM ( ... w d->n)     0 DO SWAP 1+ DUP C, U¥ DROP LOOP ;
: @DIM ( ... ' w d->' o) NCASE 2 3 ' DIM2 DIM3 CASEND ROT ¥ ;

: !00  ( d a->'0)        2DUP IF DROP OP ELSE HERE + 5 + THEN ;
: !OFF ( nw a)           NOT IF ALLOT ELSE ' OP +! THEN ;   -->

Block 261 [66 :1]

\ 860428 FM 02/10 Array-maker. AR= , the magic word itself.
\ 0   2   4 5 6  7   8 9 789
\ CFA PFA w d a PCR CR R data
\ Use in the form:  D1 D2 D3 w d a AR= <name>
\ The children do:  ( r c p -> ')
\          'Start     a     a      d  w        a
: AR= CREATE !00 , DUP C,  )R 2DUP C, C,  !DIM R)  !OFF ;CODE
   AX AX XOR DX DX XOR          BX INC BX INC BX PUSH   \ pfa
   3 # BX ADD [BX] AL MOV BX INC [BX] DL MOV DX PUSH   \ w
                                          AX PUSH      \ d
\  ]] and [[ switch between assembler and MMSFORTH.
   ]] ( r c p ' w d)  @DIM  ( ' o ) [[
\  OVER 2+ C@ IF OBASE + THEN SWAP @ + ;
   AX POP BX POP  2 [+BX] CL MOV CL CL OR
   ~ Z IF AT OBASE [] AX ADD THEN  [BX] AX ADD PSH     \ o
                                                   -->

Block 262 [67 :1]

  860430 FM 03/10 Array-maker. Experimental autoloops.
: (RW)  ( '-> R 0) 3 + DUP C@ + 1+ C@ 0 R) ROT )R SWAP )R )R ;
: (CL)  ( '-> C 0) 3 + DUP C@ +    C@ 0 R) ROT )R SWAP )R )R ;
: (PG)  ( '-> P 0) 5 +             C@ 0 R) ROT )R SWAP )R )R ;

\ Row, Column, and Page loop. Use after  ' (array-name)
: RDO  COMPILE (RW) HERE ;                          IMMEDIATE
: CDO  COMPILE (CL) HERE ;                          IMMEDIATE
: PDO  COMPILE (PG) HERE ;                          IMMEDIATE

: 2?   2@ 5 D.R ;

                                                   -->

Block 263 [68 :1]

0 \ 860428 FM 04/10 Array-maker. Practice arrays.
1 \ Arrays with data stored at OBASE OP +
2 \ R C P w d a          R C P w d a          R C P w d a
3   4     1 1 1 AR= ANN  5 4   2 2 1 AR= BEV  6 5 4 4 3 1 AR= SUE
4 \ Compatible arrays with data stored after header
5   4     1 1 0 AR= AN1  5 4   2 2 0 AR= BE1  6 5 4 4 3 0 AR= SU1
6   4     1 1 0 AR= AN2  5 4   2 2 0 AR= BE2  6 5 4 4 3 0 AR= SU2
7
8 : !ANN  ' ANN RDO              I 10 +    I ANN C! LOOP ;
9 : !BEV  ' BEV RDO ' BEV CDO 10 J ¥ ! +  J I BEV !  LOOP LOOP ;
10 : !SUE  ' SUE RDO ' SUE CDO  ' SUE PDO
11         100 K ¥ 10 J ¥ + I +  0  K J I SUE 2! LOOP LOOP LOOP ;
12 : .AN2 CR ' AN2 RDO            I AN2 C?     LOOP ;
13 : .BE2 CR ' BE2 RDO ' BE2 CDO J I BE2 @ 3 .R LOOP CR LOOP ;
14 : .SU2 CR ' SU2 RDO ' SU2 CDO 4 COLS ( =1/4-pg jump) ' SU2 PDO
15       K J I SU2 2?    LOOP LOOP CR LOOP ;                -->

Block 264 [69 :1]

0 \ 860430 FM 05/10 Array processing. Parameters to Stack.
1 \ Leaves # of entries in array, byte-width,
2 \   # of dimensions, and address of start of array: -> n w d 00
3 \ NB: NOT for row/column operators!
4 \ PFA  a  d  w  P  C  R    = Stored values
5 \ BX   2  3+ 4+ ..4+d..    = Offsets to BX
6 LABEL APARAM  DI POP  BX POP CLC
7   3 [+BX] AL MOV   AH AH XOR   AX CX MOV   \ CX=d
8       AX BX ADD  4 # BX ADD  1 # AX MOV   \ BX='R
9  BEGIN [BX] DX MOV DH DH XOR     DX MUL
10   BX DEC  LOOP                  AX PUSH  \ n  [BX]=w
11           [BX] AL MOV           AX PUSH  \ w
12   BX DEC  [BX] AL MOV           AX PUSH  \ d
13   BX DEC  [BX] AL MOV           BX DEC   \ AL=a
14   BX DEC  [BX] BX MOV 1 # AL CMP E IF AT OBASE [] BX ADD THEN
15   BX PUSH  DI PUSH RET           -->     \ '00

Block 265 [70 :1]

0 \ 860427 FM 06/10 Array processing. Parameter comparison.
1
2 : .QUIT ." Arrays are incompatible! " 2DROP .S ABORT ;
3
4 LABEL ?QUIT CX AX CMP ~ E IF
5       !BP [] BP MOV !SI [] SI MOV ]] .QUIT [[ THEN RET
6
7 \ n  w  d  00  n  w  d  00 RET       from APARAM twice.
8 \ 14 12 10  8+ 6+ 4+ 2+ BX=SP
9 LABEL ACOMP        SP BX MOV   BX INC BX INC      \ Skip Return
10    14 [+BX] CX MOV  6 [+BX] AX MOV ?QUIT CALL    \ Test count
11    12 [+BX] CX MOV  4 [+BX] AX MOV ?QUIT CALL    \ AX=width w
12                     6 [+BX] CX MOV RET   FORTH   \ CX=count n
13
14
15                                                  -->
```

```
Smart Arrays; 12\05\86 (17:00:08) 2.4-QWORAK-860220     Page 2
```

Bl\ 266 [71 :1]

```
\ 860428 FM 07/10 Array processing. Fill array with up  ramp.
CODE RAMP ( 'Dst-))
  BP !BP [] MOV  SI !SI [] MOV  SI SI XOR  \ SI=Index
  APARAM CALL   DX DX XOR  SP BX MOV
  ( -) n w d 00 ) 6 [+BX] CX MOV       \ CX=count n  BP=width
    4 [+BX] BP MOV   [BX] BX MOV  CX AX MOV AX DEC BP MUL
    AX SI ADD                    AX AX XOR
        1 # BP CMP E IF                  \ 1-byte
    BEGIN CL [BX+SI] MOV  BP SI SUB LOOP
    ELSE 2 # BP CMP E IF                 \ 2-byte
    BEGIN CX [BX+SI] MOV  BP SI SUB LOOP
    ELSE 4 # BP CMP E IF                 \ 4-byte
    BEGIN CX 2 [+BX+SI] MOV      0 # [BX+SI] MOV
                        BP SI SUB LOOP
    THEN THEN THEN        SP BX MOV  8 # BX ADD BX SP MOV
  !BP [] BP MOV !SI [] SI MOV NEXT FORTH                 -->
```

Block 267 [72 :1]

```
( 860430 FM 08/10 Array processing. Copying/Storing.)
CODE A!    ( 'Src 'Dst -))
  BP !BP [] MOV  SI !SI [] MOV  SI SI XOR  \ SI=Index
  BP POP  APARAM CALL   BP PUSH   APARAM CALL  ACOMP CALL
\ n  w  d  00  n  w  d  00    from  APARAM  twice.
\ 14 12 10 8+  6+ 4+ 2+ BX=SP          \ CX=count AX=width
    [BX] BP MOV   8 [+BX] BX MOV       \ BX=Src BP=Dst
        1 # AX CMP E IF                \ 1-byte
    BEGIN [BX+SI] DL MOV     DL [BP+SI] MOV  AX SI ADD LOOP
    ELSE 2 # AX CMP E IF                 \ 2-byte
    BEGIN [BX+SI] DX MOV     DX [BP+SI] MOV  AX SI ADD LOOP
    ELSE 4 # AX CMP E IF                 \ 4-byte
    BEGIN [BX+SI] DX MOV  DX    [BP+SI] MOV
       2 [+BX+SI] DX MOV  DX 2 [+BP+SI] MOV  AX SI ADD LOOP
    THEN THEN THEN        SP BX MOV 16 # BX ADD BX SP MOV
    !BP [] BP MOV !SI [] SI MOV NEXT FORTH             -->
```

Block 268 [73 :1]

```
 0 ( 860427 FM 09/10 Array processing. Array addition 1)
 1 CODE A+   ( 'Src1 'Src2 'Dst -))
 2   BP !BP [] MOV  SI !SI [] MOV  SI SI XOR  \ SI=Index
 3   AX POP AX !TP [] MOV
 4   BP POP APARAM CALL    BP PUSH APARAM CALL   ACOMP CALL
 5         !TP [] AX MOV  AX PUSH APARAM CALL   ACOMP CALL
 6 \ n  w  d  00  n  w  d  00  n  w  d  00    from  APARAM  thrice.
 7 \ 22 20 18 16  14 12 10 8+  6+ 4+ 2+ BX=SP    CX=count AX=width
 8   [BX] DI MOV 8 [+BX] BP MOV 16 [+BX] BX MOV \ DI=Dst BP,BX=Srcs
 9            1 # AX CMP E IF                 \ 1-byte
10   BEGIN [BP+SI] AX MOV  AH AH XOR  [BX+SI] AX ADD
11           AL [DI] MOV  1 # DI ADD     1 # SI ADD LOOP
12       ELSE 2 # AX CMP E IF                 \ 2-byte
13   BEGIN [BP+SI] AX MOV             [BX+SI] AX ADD
14           AX [DI] MOV  2 # DI ADD     2 # SI ADD LOOP
15                                                      -->
```

Block 269 [74 :1]

```
 0 \ 860430 FM 10/10 Array processing. Array addition 2
 1       ELSE 4 # AX CMP E IF                 \ 4-byte
 2   BEGIN [BP+SI] AX MOV             [BX+SI] AX ADD
 3           AX [DI] MOV  2 # DI ADD     2 # SI ADD
 4         [BP+SI] DX MOV             [BX+SI] DX ADC
 5           DX [DI] MOV  2 # DI ADD     2 # SI ADD LOOP
 6     THEN THEN THEN        SP BX MOV  24 # BX ADD BX SP MOV
 7     !BP [] BP MOV !SI [] SI MOV NEXT     FORTH
 8 : AFILL   ' ANN RAMP     ' BEV RAMP     ' SUE RAMP ;
 9 : ACOPY   ' ANN ' AN1 A! ' BEV ' BE1 A! ' SUE ' SU1 A! ;
10 : AADD    ' ANN ' AN1 ' AN2 A+    ' BEV ' BE1 ' BE2 A+
11           ' SUE ' SU1 ' SU2 A+ ;
12 : A.      .AN2 .BE2 .SU2 ;
13 \ Set start of indirect arrays
14 HEX : GO   A000 IS OBASE ;       GO                  DECIMAL
15 : ATEST    AFILL ACOPY AADD A. ;
```