APMATH64/MAX MANUAL

VOLUME 4 OF 4

MODELS M64/140, M64/145

860–7482–001C

FLOATING POINT SYSTEMS, INC.

APMATH64/MAX MANUAL

VOLUME 4 OF 4

MODELS M64/140, M64/145

860-7482-001C

by FPS Technical Publications Staff

Publication No. 860-7482-001C
December, 1987

NOTICE

The information in this publication is subject to change without notice.

Floating Point Systems, Inc. accepts no liability for any loss, expense, or damage resulting from the use of any information appearing in this publication.

Printed in USA

The postpaid Reader's Comment Form on the last page of this document requests the user's critical evaluation to assist in preparing and revising future documents.

REVISION HISTORY

This manual is the *APMATH64/MAX Manual*, Volume 4, 860-7482-001. The letter shown under the revision number column indicates the portion of the part number that changes for each revision. The last entry is the latest revision to this manual.

REV. NO.	DESCRIPTION	DATE
-001A	The revision history begins with this manual.	8/86
-001B	Deleted Utilities Library, deleted the LPSPFI subroutine, added internal subroutine information, and added 16 new routines.	1/87
-001C	Added new routines to Basic Math Library, Double Precision Library, and Matrix Algebra Accelerated Math Library.	12/87

NOTE: For revised manuals, a vertical line "|" outside the left margin of the text signifies where changes have been made.

NOTE TO READER

This is the fourth volume of the APMATH64 Manual. It is comprised of Appendix K, Appendix L, and a key word index for the APMATH64/MAX routines. Note that Appendix A continues through Volumes 1, 2, and 3. The page numbers are listed consecutively through the volumes.

The APMATH64 Manual has three indices located at the end of Volume 3 and two at the end of Volume 4. The first index (Appendix I) is a list of the APMATH64 routines in page order by type. The second index (Appendix J) is an alphabetical list of the APMATH64 routines. The third index is a key word index of the APMATH64 routines. The fourth index (Appendix L) is an alphabetical list of the APMATH64/MAX routines. The fifth index is a key word index of the APMATH64/MAX routines.

CONTENTS (VOLUME 4)

ILLUSTRATIONS

TABLES

APPENDIX K

MATRIX ALGEBRA ACCELERATOR MATH LIBRARY ROUTINES

K.1 INTRODUCTION

This appendix contains documentation for the Matrix Algebra Accelerator (MAX) Math Library routines. A high-level description of the MAX system is presented, followed by documentation for the two categories of MAX Math Library routines: the Basic MAX routines and the Matrix Oriented MAX routines.

K.2 SYSTEM OVERVIEW

A M64/140 or M64/145 MAX system consists of a Central Processing Unit (CPU), a memory unit, and one or more MAX modules. The MAX modules are collectively referred to as the "MAX". Each MAX module is capable of performing calculations in parallel with the M64/140 or M64/145 CPU as well as with other MAX modules.

The MAX Math Library software requires an M64/140 or M64/145 that has been configured with at least 16K Table Memory RAM (TMRAM). The software can utilize an arbitrary number of MAX modules up to and including the maximum configuration of fifteen.

An individual MAX module contains eight vector memories, each 2K (2048) words in length. The Basic MAX routines are restricted to usage of the first 2K-1 (2047) locations of each vector memory. The Matrix Oriented MAX routines restrict usage as appropriate to the particular operation (refer to the manual page).

The hardware on a MAX module directly supports the basic vector operations real and complex dot product, real vector scalar multiply and add (VSMA), and real vector multiply and scalar add (VMSA). Since the dot product operation is a vector to scalar operation, up to eight separate real dot products or four separate complex dot products can be performed on a single MAX module. That is, all eight vector memories can contain input vectors for dot products. Since the VSMA and VMSA operations are vector to vector operations, up to four separate VSMA's or VMSA's can be performed on a single MAX module. For VSMA and VMSA operations, the vector memories on each MAX module are grouped into two banks of four vector memories each. One bank of four vector memories contains the input vectors, while the other bank of four vector memories contains the resulting output vectors.

The CPU supplements the MAX performance by using TMRAM as an additional set of vector memories. In this way, the CPU can perform up to four real dot products, two complex dot products, one VSMA, or one VMSA in parallel with the MAX modules. Most of the MAX Math Library routines

The ranges of values which will produce overflow or underflow during conversion from FPS to IEEE and back to FPS format are given below. This conversion could be obtained by a vector move of data from Main Memory to a MAX vector register followed by a vector move of the data from the MAX vector register back to Main Memory. Both the standard scientific notation and the FPS internal binary representation (in hexadecimal) is given for each number.

```
 0.89884656743115775 0E+308  --+
                               |
   FFEF FFFF FFFF FFFF         |
             .                 |
             .                 |  Positive overflow
             .                 |
                               |
 0.44942328371557898 0E+308    |
                               |
   FFE8 0000 0000 0000       --+


 0.44942328371557888 0E+308  --+
                               |
   FFCF FFFF FFFF FFFF         |
             .                 |
             .                 |  No overflow/underflow
             .                 |
                               |
 0.22250738585072014 0E-307    |
                               |
   0068 0000 0000 0000       --+


 0.22250738585072009 0E-307  --+
                               |
   004F FFFF FFFF FFFF         |
             .                 |
             .                 |  Positive underflow
             .                 |
                               |
 0.27813423231340017 0E-308    |
                               |
   0008 0000 0000 0000       --+


 0.0
                                  True zero
   0000 0000 0000 0000


-0.27813423231340030 0E-308  --+
                               |
   0017 FFFF FFFF FFFF         |
             .                 |
             .                 |  Negative underflow
```

MAX formatter exception interrupts can be selectively enabled or disabled in a manner similar to the arithmetic exception interrupts. The default disables all MAX formatter exception interrupts.

The following code fragment enables all MAX formatter exception interrupts:

```
INTEGER UNFL, UNNORM, DENORM, OVFL
     .
     .
     .
UNFL = 1
UNNORM = 1
DENORM = 1
OVFL = 1
CALL SYS$ENA_FMTERR(UNFL, UNNORM, DENORM, OVFL)
```

The following code fragment disables all MAX formatter exception interrupts.

```
INTEGER UNFL, UNNORM, DENORM, OVFL
     .
     .
     .
UNFL = 1
UNNORM = 1
DENORM = 1
OVFL = 1
CALL SYS$DIS_FMTERR(UNFL, UNNORM, DENORM, OVFL)
```

For more information on enabling and disabling exception interrupts, refer to the documentation for routines SYS$ENAEXC, SYS$DISEXC, SYSENA_FMTERR, SYSDIS_FMTERR in the *Operating System Manual, Volume 3, File and Memory Management*, listed in Section 1.5.

K.3 BASIC MAX ROUTINES

The Basic MAX routines provide for basic functional utilization of the MAX system and are vector oriented.

K.3.1 Overview

The Basic MAX routines are designed to access the basic functionality of the MAX modules. These routines are not as flexible as the Matrix Oriented MAX routines in terms of data management in the MAX system.

The configuration table used by the Basic MAX routines references the available MAX modules. The table is always placed into the 17 highest addressable locations in TMRAM regardless of the base address of the TMRAM workspace. Hence, for 16K TMRAM systems, the table will be situated in locations 24559-24575.

Mathematically, the operations to be performed are:

```
S(i) = SUM[A(j) * B(i,j);  j = 1,4];   i = 1,14
```

The following is the APFTN64 code to perform the desired operation. Briefly, the PLOADD routine is called to load the array of vectors B into TMRAM and the MAX vector memories. The PDOT routine is then called to compute the dot products and return the results into the array S. (Refer to the documentation on PLOADD and PDOT for a discussion of the parameter values.)

```
      REAL A(4),B(14,4),S(14)
      INTEGER I,J,N,M,ITMA,ISTART,IFUN,IERR
                 .
                 .
                 .
      I = 14
      J = 1
      N = 4
      M = 14
      ITMA = 8192
      ISTART = 1
      CALL PLOADD(B,I,J,N,M,ITMA,ISTART,IERR)
      IF(IERR.NE.Ø) GO TO 9ØØ
      I = 1
      J = 1
      N = 4
      M = 14
      ITMA = 8192
      ISTART = 1
      IFUN = Ø
      CALL PDOT(A,I,N,S,J,M,ITMA,ISTART,IFUN,IERR)
      IF(IERR.NE.Ø) GO TO 9ØØ
                 .
                 .
                 .
  9ØØ CONTINUE
```

resulting output vectors and store them in the array of vectors C. (Refer to the documentation on PLOADV, PVSMA, and PUNLDV for a discussion of the parameter values.)

```
      REAL A(4),S(7),B(7,4),C(7,4)
      INTEGER I,J,N,M,ITMA,ISTART,IFUN,IERR,IBANK
                    .
                    .
                    .
      I = 7
      J = 1
      N = 4
      M = 7
      IBANK = Ø
      ITMA = 8192
      ISTART = 1
      CALL PLOADV(B,I,J,N,M,IBANK,ITMA,ISTART,IERR)
      IF(IERR.NE.Ø) GO TO 9ØØ
      I = 1
      J = 1
      N = 4
      M = 7
      ITMA = 8192
      ISTART = 1
      IFUN = Ø
      CALL PVSMA(A,I,S,J,N,M,IBANK,ITMA,ISTART,IFUN,IERR)
      IF(IERR.NE.Ø) GO TO 9ØØ
      I = 7
      J = 1
      N = 4
      M = 7
      ITMA = 8192
      ISTART = 1
      CALL PUNLDV(C,I,J,N,M,IBANK,ITMA,ISTART,IERR)
      IF(IERR.NE.Ø) GO TO 9ØØ
                    .
                    .
                    .
  9ØØ CONTINUE
```

Upon successful execution of PLOADV, PVSMA, and PUNLDV, the array C contains the following:

```
      C =  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
           Ø.9  Ø.8  Ø.7  Ø.6
           1.8  1.6  1.4  1.2
           2.7  2.4  2.1  1.8
           3.6  3.2  2.8  2.4
           4.5  4.Ø  3.5  3.Ø
           5.4  4.8  4.2  3.6
```

Basically, PLOADD is called to load M rows of the matrix A into TMRAM and the MAX vector memories. Each call to PDOT performs the M dot products of one column of the matrix B with the M rows of the matrix A loaded by PLOADD. These calculations are shown in Figure K-1.

```
+              +    +                    +     +              +
|              |    |                    |     |   x          |
|   x          |    | x x x . . . x x x  |     |   x          |
|   x          |    | x x x . . . x x x  |     |   x          |
|   .          |    | . . . . . . . . .  |     |   .          |
|   .          | =  | . . . . . . . . .  |  *  |   .          |
|   .          |    | . . . . . . . . .  |     |   .          |
|   x          |    | x x x . . . x x x  |     |   x          |
|   x          |    | x x x . . . x x x  |     |   x          |
|              |    |                    |     |   x          |
+              +    +                    +     +              +

      C                      A                       B

   N1 x N3                N1 x N2                 N2 x N3
```

Figure K-1 Sample Calculations of Matrix Multiply

In general, the number of rows of the matrix A (N1 above) is not an integral multiple of the number of dot products that a given system can perform at the same time (M above). It is straightforward to add APFTN64 code to handle the remaining MOD(N1,M) rows of the matrix A, as well as to handle the case where N1 is less than M.

K.3.3 <u>Routine Documentation</u>

This section contains the descriptions of the Basic MAX routines.

```
***********                                                ***********
*         *                                                *         *
*  PCDOT  *      --- PARALLEL COMPLEX DOT PRODUCT ---      *  PCDOT  *
*         *                                                *         *
***********                                                ***********
```

PURPOSE: To compute the complex dot products of a single vector with each of a set of vectors that were loaded by PLDCD.

CALL FORMAT: CALL PCDOT(A, I, N, S, J, M, ITMA, ISTART, IFUN, IERR)

PARAMETERS:

```
A      = Floating-point input complex vector.
I      = Integer input element stride for A.
N      = Integer input number of elements in A.
S      = Floating-point output complex vector.
J      = Integer input element stride for S.
M      = Integer input number of vectors loaded by PLDCD.
ITMA   = Integer input TMRAM workspace base address
         from the most recent call to PLDCD.
ISTART = Integer input starting index into the TMRAM
         workspace and the MAX vector memories to begin
         loading vectors.  The first location of the MAX
         vector memories and the TMRAM workspace has an
         index value equal to one.
IFUN   = Integer input addition/subtraction flag.
         IFUN =  Ø:  Use addition.
         IFUN <> Ø:  Use subtraction.
IERR   = Integer output error flag.
```

DESCRIPTION: PCDOT performs the M complex dot products of the vector contained in A with the M vectors loaded by a previous call to PLDCD. There is no check as to whether or not the vectors were actually loaded by PLDCD. The results are stored in the vector S.

```
S(J*(2j-1))   = SUM[s * A(I*(2i-1),j) *
                  W(i+ISTART-1),j);  i = 1 to N]

S(J*(2j-1)+1) = SUM[s * A(I*(2i),j) *
                  W(i+ISTART-1),j);  i = 1 to N]

                j = 1 to M
```

where

```
s =  1.Ø if IFUN  = Ø
s = -1.Ø if IFUN <> Ø
```

and W(*,1:M) are the M vectors loaded by PLDCD.

MAX module #2

```
Vector memory A: 6.1  6.3  6.5
              B: 6.2  6.4  6.6
              C: 7.1  7.3  7.5
              D: 7.2  7.4  7.6
              E: 8.1  8.3  8.5
              F: 8.2  8.4  8.6
              G: 9.1  9.3  9.5
              H: 9.2  9.4  9.6
```

TMRAM

```
TM (8192): 1.1
TM (8193): 1.2
TM (8194): Ø.Ø
TM (8195): Ø.Ø
TM (8196): 1.3
TM (8197): 1.4
TM (8198): Ø.Ø
TM (8199): Ø.Ø
TM (82ØØ): 1.5
TM (82Ø1): 1.6
TM (82Ø2): Ø.Ø
TM (82Ø3): Ø.Ø
TM (82Ø4): x.x
     .
     .
     .
TM(24558): x.x
-------------------------------
TM(24559):       MAX
     .
     .           Configuration
     .
TM(24575):       Table
```

Given the following input parameters to PCDOT:

```
N      = 3
M      = 9
I      = 2
J      = 2
ITMA   = 8192
ISTART = 1
IFUN   = Ø

A = (1.Ø,Ø.Ø)   (1.Ø,Ø.Ø)   (1.Ø,Ø.Ø)
```

```
**********                                                    **********
*        *                                                    *        *
* PCNV2D *  -- PARALLEL 2-D CONVOLUTION AND CORRELATION --    * PCNV2D *
*        *                                                    *        *
**********                                                    **********
```

```
PURPOSE:      To perform a 2-D convolution or correlation operation
              on two matrices using the M64/140 or M64/145.

CALL FORMAT: CALL PCNV2D(A,MA,IA,JA,M,N,B,MB,NB,IB1,C,MC,IC,JC,IR,ITMA)

PARAMETERS:   A    = Floating-point input operand matrix. (column
                     ordered)
              MA   = Integer input number of rows of A
              IA   = Integer input initial row of the submatrix A'
                     of A to be processed  (1 <= IA <= MA)
              JA   = Integer input initial column of the submatrix
                     A' of A to be processed  (JA >= 1)
              M    = Integer input number of rows in A'
                     (1 <= M <= MA)
              N    = Integer input number of columns in A'
                     (N >= 1)
              B    = Floating-point input operator matrix. (column
                     ordered)
              MB   = Integer input number of rows of B
              NB   = Integer input number of columns of B
              IB1  = Integer index of the operator element B that
                     is to coincide with the first operand element
                     of A' to be processed  and output as the
                     corresponding element of C'.  For correlation,
                     this index is counted columnwise relative to
                     the upper left-hand corner element of B.
                     For convolution, this index is counted
                     columnwise relative to the lower right-hand
                     corner element of B, since B is reversed.
                     (1 <= IB1 <= MB*NB)
              C    = Floating-point output matrix. (column ordered)
              MC   = Integer input number of rows of C
              IC   = Integer input initial row of C which locates the
                     submatrix C' or C; C' will be the processed A'
                     (1 <= IC <= MC)
              JC   = Integer input initial column of C which locates
                     the submatrix C' of C (JC >= 1)
              IR   = Integer input scalar flag
                        IR  = 0:  Perform convolution
                        IR <> 0:  Perform correlation
              ITMA = Integer input TMRAM workspace base address

DESCRIPTION: C(ic,jc) = SUM[SUM[A(ka,la) * B(k,l); k=1,MB]; l=1,NB]
              for   i=1 to M
                    j=1 to N
              where ic = i+IC-1
```

```
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
   IB1  = 9
   IR   = Ø
   C:   6.Ø  5.Ø 11.Ø 23.Ø 28.Ø 28.Ø -8.Ø  Ø.Ø  Ø.Ø
        5.Ø  4.Ø 1Ø.Ø 19.Ø 24.Ø 2Ø.Ø -8.Ø  Ø.Ø  Ø.Ø
        5.Ø  4.Ø 1Ø.Ø 19.Ø 24.Ø 2Ø.Ø -8.Ø  Ø.Ø  Ø.Ø
        6.Ø  5.Ø 11.Ø 23.Ø 28.Ø 28.Ø -8.Ø  Ø.Ø  Ø.Ø
       -1.Ø -1.Ø -1.Ø -4.Ø -4.Ø -8.Ø  Ø.Ø  Ø.Ø  Ø.Ø
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
        Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø
```

If N is less than one, then

S(j) = Ø.Ø; j = 1, M

Summary of error conditions:

IERR = Ø	Normal return.
IERR = 1	Insufficient MAX vector memories and TMRAM to hold the number of vectors designated by M. Each MAX module can hold eight vectors. TMRAM can hold four vectors.
IERR = 2	Vector length too large. N+ISTART-1 must be less than or equal to 2Ø47.
IERR = 3	ISTART or M is less than or equal to zero. Both of these parameters must be positive.
IERR = 4	Insufficient TMRAM space available. There must be enough TMRAM available to hold 4*(N+ISTART-1) + 17 words, starting at the TMRAM workspace base address, ITMA. Although PDOT does not load TMRAM, it does check for consistency between N, ISTART, and ITMA.
IERR = 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

EXAMPLE: Given a system with two available MAX modules and 16K TMRAM that has been initialized by PLOADD as follows:

MAX module #1

Vector memory	A:	3.1	3.2	3.3	3.4
	B:	4.1	4.2	4.3	4.4
	C:	5.1	5.2	5.3	5.4
	D:	6.1	6.2	6.3	6.4
	E:	7.1	7.2	7.3	7.4
	F:	8.1	8.2	8.3	8.4
	G:	9.1	9.2	9.3	9.4
	H:	1Ø.1	1Ø.2	1Ø.3	1Ø.4

MAX module #2

Vector memory	A:	11.1	11.2	11.3	11.4
	B:	12.1	12.2	12.3	12.4
	C:	13.1	13.2	13.3	13.4
	D:	14.1	14.2	14.3	14.4

Upon RETURN from PDOT, S contains:

```
S =  5.Ø
     9.Ø
    13.Ø
    17.Ø
    21.Ø
    25.Ø
    29.Ø
    33.Ø
    37.Ø
    41.Ø
    45.Ø
    49.Ø
    53.Ø
    57.Ø
```

```
S(1+(j-1)*I+(k-1)*J) = (c*S(1+(j-1)*I+(k-1)*J)

 + SUM[s*A((i-1)*IA+1)*V(INDEX(i)+j-1,k); i=1,NA]; k=1,M)

                j = 1,NP
```

where

```
c =  1.Ø if IACC  = Ø
c =  Ø.Ø if IACC <> Ø

s =  1.Ø if IFUN  = Ø
s = -1.Ø if IFUN <> Ø
```

and V(*,1:M) are the M vectors loaded by PILOAD.

If NA is less than one, then

```
S(i,j) = Ø.Ø; i = 1 to M, j = 1 to NP
```

Summary of error conditions:

IERR = Ø	Normal return.
IERR = 1	Insufficient MAX vector memories to hold the number of vectors designated by M. Each MAX module can hold 8 vectors.
IERR = 2	NA, NP, or M is less than or equal to zero. Each of these parameters must be positive.
IERR = 3	Insufficient TMRAM space available. There must be enough TMRAM to hold 17 words starting at the TMRAM workspace base address ITMA.
IERR = 4	ITMA less than 8192. ITMA must be greater than or equal to 8192.

EXAMPLE: Given a system with one available MAX module and 16K TMRAM that has been initialized by PILOAD as follows:

```
**********                                           **********
*        *                                           *        *
* PILOAD *       --- PARALLEL LOAD FOR PIDOT ---     * PILOAD *
*        *                                           *        *
**********                                           **********
```

PURPOSE: To load vectors from Main Memory into the MAX vector memories in preparation for calls to PIDOT.

CALL FORMAT: CALL PILOAD (A, I, J, N, M, ITMA, IPTR, IERR)

PARAMETERS:

- A = Floating-point input array of vectors.
- I = Integer input element stride for vectors in A.
- J = Integer input element stride between vectors in A.
- N = Integer input number of elements per vector in A.
- M = Integer input number of vectors to transfer.
- ITMA = Integer input TMRAM workspace base address.
- IPTR = Integer input offset into the MAX vector memories to begin accessing vector elements. This parameter is different than the ISTART parameter in PLOADD.
- IERR = Integer output error flag.

DESCRIPTION: PILOAD loads the vectors contained in A into the MAX vector memories in preparation for calls to PIDOT. MAX vector memory location zero is reserved for PIDOT, therefore requiring the IPTR parameter to be greater than zero. PILOAD also sets up the MAX configuration table in TMRAM.

Summary of Error Conditions:

IERR = Ø Normal return.

IERR = 1 Insufficient number of MAX vector memories to hold the number of vectors designated by M. Each MAX module can hold 8 vectors.

IERR = 2 Vector length too large. N+IPTR must be less than or equal to 2Ø47.

IERR = 3 One or more of IPTR, N, or M is less than or equal to zero. Each of these parameters must be positive.

MAX module #2

Vector memory	A:	x.x	9.1	9.2	9.3	9.4
	B:	x.x	1Ø.1	1Ø.2	1Ø.3	1Ø.4
	C:	x.x	11.1	11.2	11.3	11.4
	D:	x.x	12.1	12.2	12.3	12.4
	E:	x.x	13.1	13.2	13.3	13.4
	F:	x.x	14.1	14.2	14.3	14.4

TMRAM

```
TM (8192):      MAX
     .
     .          Configuration
     .
TM (82Ø8):      Table
```

IERR = 2	Vector length too large. N+ISTART-1 must be less than or equal to 2047.
IERR = 3	One or more of ISTART, N, or M is less than or equal to zero. Each of these parameters must be positive.
IERR = 4	Insufficient TMRAM space available. There must be enough TMRAM available to hold 4*(N+ISTART-1) + 17 words, starting at the TMRAM workspace base address, ITMA.
IERR = 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

EXAMPLE: Given a system with two available MAX modules and 16K TMRAM and the following input parameters to PLDCD:

```
N      = 3
M      = 9
I      = 18
J      = 2
ITMA   = 8192
ISTART = 1

A = (1.1,1.2)   (1.3,1.4)   (1.5,1.6)
    (2.1,2.2)   (2.3,2.4)   (2.5,2.6)
    (3.1,3.2)   (3.3,3.4)   (3.5,3.6)
    (4.1,4.2)   (4.3,4.4)   (4.5,4.6)
    (5.1,5.2)   (5.3,5.4)   (5.5,5.6)
    (6.1,6.2)   (6.3,6.4)   (6.5,6.6)
    (7.1,7.2)   (7.3,7.4)   (7.5,7.6)
    (8.1,8.2)   (8.3,8.4)   (8.5,8.6)
    (9.1,9.2)   (9.3,9.4)   (9.5,9.6)
```

Note that in this example, A is a two-dimensional complex matrix whose elements are stored in column major order.

The vectors are loaded into the MAX vector memories as real and imaginary pairs as well with the real part of the first number loaded into memory A and the imaginary part in vector vector memory B. The next number will be loaded into vector memories C (real part) and D (imaginary part), etc.

IERR = 2	Vector length too large. N+ISTART-1 must be less than or equal to 2047.
IERR = 3	One or more of ISTART, N, or M is less than or equal to zero. Each of these parameters must be positive.
IERR = 4	Insufficient TMRAM space available. There must be enough TMRAM available to hold 4*(N+ISTART-1) + 17 words, starting at the TMRAM workspace base address, ITMA.
IERR = 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

EXAMPLE: Given a system with two available MAX modules and 16K TMRAM and the following input parameters to PLOADD:

```
N      =    4
M      =   14
I      =   14
J      =    1
ITMA   = 8192
ISTART =    1

A =  1.1  1.2  1.3  1.4
     2.1  2.2  2.3  2.4
     3.1  3.2  3.3  3.4
     4.1  4.2  4.3  4.4
     5.1  5.2  5.3  5.4
     6.1  6.2  6.3  6.4
     7.1  7.2  7.3  7.4
     8.1  8.2  8.3  8.4
     9.1  9.2  9.3  9.4
    10.1 10.2 10.3 10.4
    11.1 11.2 11.3 11.4
    12.1 12.2 12.3 12.4
    13.1 13.2 13.3 13.4
    14.1 14.2 14.3 14.4
```

Note that in this example, A is a two-dimensional array whose elements are stored in column major order.

```
**********                                              **********
*        *                                              *        *
* PLOADV *         --- PARALLEL LOAD FOR ---            * PLOADV *
*        *              PVSMA AND PVMSA                 *        *
**********                                              **********
```

PARALLEL LOAD FOR PVSMA AND PVMSA

PURPOSE: To load vectors from Main Memory into TMRAM and the MAX vector memories in preparation for calls to PVSMA or PVMSA.

CALL FORMAT: CALL PLOADV(A, I, J, N, M, IBANK, ITMA, ISTART, IERR)

PARAMETERS:

```
A      = Floating-point input array of vectors.
I      = Integer input element stride for vectors in A.
J      = Integer input element stride between
           vectors in A.
N      = Integer input number of elements per vector
           in A.
M      = Integer input number of vectors to transfer.
IBANK  = Integer input TMRAM region and bank of MAX
           vector memories to load.
           IBANK =  Ø:  Load first TMRAM region and MAX
                        vector memories A,B,C,D.
           IBANK <> Ø:  Load second TMRAM region and
                        MAX vector memories E,F,G,H.
ITMA   = Integer input TMRAM workspace base address.
ISTART = Integer input starting index into the TMRAM
           region and the MAX vector memories to begin
           loading vector elements.  The first location
           of the TMRAM region and the MAX vector
           memories has an index of one.
IERR   = Integer output error flag.
```

DESCRIPTION: PLOADV loads the vectors contained in A into TMRAM and the MAX vector memories in preparation for calls to PVSMA or PVMSA. TMRAM is loaded first, so that if M equals 1, then only TMRAM is loaded. The remaining vectors are loaded into the MAX vector memories. PLOADV also sets up the MAX configuration table in the high end of TMRAM. The rest of the TMRAM workspace is partitioned into two regions. The first region corresponds functionally to MAX vector memories A,B,C,D, while the second region corresponds functionally to MAX vector memories E,F,G,H.

Note that in this example, A is a two-dimensional array whose elements are stored in column major order.

Upon RETURN from PLOADV, the MAX modules and TMRAM contain:

MAX module #1

```
Vector memory E:  3.1  3.2  3.3  3.4
              F:  5.1  5.2  5.3  5.4
              G:  7.1  7.2  7.3  7.4
              H:  9.1  9.2  9.3  9.4
```

MAX module #2

```
Vector memory E: 11.1 11.2 11.3 11.4
              F: 13.1 13.2 13.3 13.4
```

TMRAM

```
TM (8192):  x.x     First
     .
     .              TMRAM
     .
TM(16374):  x.x     Region
--------------------------------
TM(16375):  1.1
TM(16376):  1.2
TM(16377):  1.3     Second
TM(16378):  1.4
TM(16379):  x.x     TMRAM
     .
     .              Region
     .
TM(24558):  x.x
--------------------------------
TM(24559):          MAX
     .
     .              Configuration
     .
TM(24575):          Table
```

EXAMPLE:

INPUT:

Let the input matrix A be:

1.00	-1.00	-1.00	1.00	1.00	1.00	1.00	9.99
-2.00	0.00	1.00	-1.00	-1.00	-1.00	1.00	9.99
1.00	-1.00	0.00	1.00	1.00	1.00	1.00	9.99
1.00	1.00	1.00	0.00	1.00	1.00	-1.00	9.99
-1.00	1.00	-1.00	-2.00	0.00	1.00	1.00	9.99
1.00	-1.00	1.00	1.00	-1.00	0.00	1.00	9.99
0.00	1.00	1.00	-1.00	1.00	1.00	0.00	9.99
9.99	9.99	9.99	9.99	9.99	9.99	9.99	9.99

LA = 8

N = 7

OUTPUT:

Output matrix A:

-0.50	0.00	1.00	-1.00	-1.00	-1.00	1.00	9.99
-0.50	-1.00	-0.50	0.50	0.50	0.50	1.50	9.99
0.50	-1.00	-0.50	-1.00	1.00	2.00	2.00	9.99
-0.50	1.00	-1.00	-1.00	-1.00	1.00	2.00	9.99
0.00	-1.00	-0.25	0.75	0.40	1.25	0.50	9.99
-0.50	-1.00	-0.50	0.50	0.80	2.00	0.60	9.99
-0.50	1.00	-0.50	0.50	0.40	0.00	-5.00	9.99
9.99	9.99	9.99	9.99	9.99	9.99	9.99	9.99

IPVT = 2 2 5 6 7 6 7

INFO = 0

The pivot vector IPVT contains the row interchange information that was generated by PLUFAC while performing partial pivoting.

EXAMPLE:

INPUT:

Let the factored input matrix A be:

-Ø.5Ø	Ø.ØØ	1.ØØ	-1.ØØ	-1.ØØ	-1.ØØ	1.ØØ	9.99
-Ø.5Ø	-1.ØØ	-Ø.5Ø	Ø.5Ø	Ø.5Ø	Ø.5Ø	1.5Ø	9.99
Ø.5Ø	-1.ØØ	-Ø.5Ø	-1.ØØ	1.ØØ	2.ØØ	2.ØØ	9.99
-Ø.5Ø	1.ØØ	-1.ØØ	-1.ØØ	-1.ØØ	1.ØØ	2.ØØ	9.99
Ø.ØØ	-1.ØØ	-Ø.25	Ø.75	Ø.4Ø	1.25	Ø.5Ø	9.99
-Ø.5Ø	-1.ØØ	-Ø.5Ø	Ø.5Ø	Ø.8Ø	2.ØØ	Ø.6Ø	9.99
-Ø.5Ø	1.ØØ	-Ø.5Ø	Ø.5Ø	Ø.4Ø	Ø.ØØ	-5.ØØ	9.99
9.99	9.99	9.99	9.99	9.99	9.99	9.99	9.99

LA = 8

X = 3.ØØ 6.ØØ -3.ØØ 9.ØØ -3.ØØ 1.ØØ -1.ØØ 9.99

LX = 8

N = 7

M = 1

IPVT = 2 2 5 6 7 6 7

OUTPUT:

Solution matrix X:

-8.ØØ 75.ØØ -6.ØØ 56.ØØ 4.ØØ -18.ØØ 38.ØØ 9.99

where

```
s = +1.Ø        ( when IFUN =  Ø )
  = -1.Ø        ( when IFUN <> Ø )
```

TMRAM is used by this routine to hold up to 4 vectors of length MIN(NA,2Ø47), and also to hold a compressed MAX module configuration table of up to 17 words. The routine checks the amount of TMRAM on the system and returns an error code (see Summary of Error Conditions below) if the above requirement is not met.

Implementation Note:

Vector lengths are not restricted to the length of the MAX vector memories. When the vector length exceeds this length (i.e., NA > 2Ø47), partial dot products are calculated in the MAX and TMRAM and are accumulated on the M64/14Ø or M64/145.

Summary of Error Conditions:

IERR = Ø — Normal return.

IERR = 1 — One or more of LA, LB, LC, MC, NC, or NA is less than or equal to zero. Each of these parameters must be positive.

IERR = 2 — ITMA less than 8192. ITMA must be greater than or equal to 8192.

IERR = 3 — Insufficient TMRAM scratch space. See DESCRIPTION section for details of TMRAM requirements.

EXAMPLE:

INPUT:

```
A:   1.Ø  1.Ø  1.Ø  1.Ø
     2.Ø  2.Ø  2.Ø  2.Ø
     3.Ø  3.Ø  3.Ø  3.Ø
     4.Ø  4.Ø  4.Ø  4.Ø
     5.Ø  5.Ø  5.Ø  5.Ø
     Ø.Ø  Ø.Ø  Ø.Ø  Ø.Ø

B:   1.Ø  2.Ø  3.Ø  4.Ø
     1.Ø  2.Ø  3.Ø  4.Ø
     1.Ø  2.Ø  3.Ø  4.Ø
     1.Ø  2.Ø  3.Ø  4.Ø
```

```
***********                                          ***********
*         *                                          *         *
*  PMOVE  *          ---- PARALLEL MOVE ----         *  PMOVE  *
*         *                                          *         *
***********                                          ***********
```

PURPOSE: To move a row of elements across the MAX modules and to move a single element in TM.

CALL FORMAT: CALL PMOVE(A, IROW1, IROW2, ISTEP, N, IBANK, ITMA, IERR)

PARAMETERS:

A	= Floating-point output matrix.
IROW1	= Integer input element row index.
IROW2	= Integer input element row index.
ISTEP	= Integer input element stride for A.
N	= Integer input number of elements to move.
IBANK	= Integer input bank switch (Ø or 1).
ITMA	= Integer input base address of the TMRAM workspace.
IERR	= Integer output error flag.

DESCRIPTION: PSWAP moves the element in TM indexed by IROW2 out to Main Memory (matrix A) and moves the element indexed by IROW1 to the location indexed by IROW2. The elements indexed by IROW2 in the MAX vector memories are also moved out to Main Memory (matrix A) and the elements indexed by IROW1 across the MAX vector memories are moved to the location indexed by IROW2.

If IBANK = Ø, then only elements in vector memories A, B, C, and D of the MAX vector memories are moved. If IBANK = 1 then only elements in vector memories E, F, G, and H are moved.

This routine can be used in matrix factorization.

Summary of error conditions:

IERR = Ø	Normal return.
IERR = 1	Insufficient MAX vector memories and TMRAM to hold the number of elements designated by N. Each MAX module can hold four vectors. TMRAM can hold one vector.
IERR = 3	One or more of IROW1, IROW2, or N is less than or equal to zero. Each of these parameters must be positive.
IERR = 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

```
A =  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Upon RETURN from PMOVE, A contains:

```
A =  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
     5.1  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9
     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

TMRAM contains:

```
TM (8192):  1.1
TM (8193):  2.1
TM (8194):  3.1
TM (8195):  4.1
TM (8196):  2.1
TM (8197):  6.1
    .
    .
    .
---------------------------
TM(24559):  MAX
    .
    .       Configuration
    .
TM(24575):  Table
```

The MAX vector memories contain:

MAX Module #1

```
Vector Memory: A = 1.2  2.2  3.2  4.2  2.2  6.2
               B = 1.3  2.3  3.3  4.3  2.3  6.3
               C = 1.4  2.4  3.4  4.4  2.4  6.4
               D = 1.5  2.5  3.5  4.5  2.5  6.5
```

MAX module #2

```
Vector Memory: A = 1.6  2.6  3.6  4.6  2.6  6.6
               B = 1.7  2.7  3.7  4.7  2.7  6.7
               C = 1.8  2.8  3.8  4.8  2.8  6.8
               D = 1.9  2.9  3.9  4.9  2.9  6.9
```

INFO=K indicates that the K-th diagonal element became Ø.Ø. This does not cause the routine to return, but indicates that a divide by Ø occurs if the matrix-solving routine SGESL is called with this output. If more that one diagonal element becomes Ø.Ø, then INFO is set to the last one to do so.

The original input matrix A is overwritten by the factored matrix.

For further information, see the LINPACK routine by the same name. Dongarra, J.J., Bunch, J.R., Moler, C.B., and Stewart, G.W. LINPACK User's Guide. Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1979 Pa., 1979.

EXAMPLE:

INPUT:

Let the input matrix A be:

1.ØØ	-1.ØØ	-1.ØØ	1.ØØ	1.ØØ	1.ØØ	1.ØØ	9.99
-2.ØØ	Ø.ØØ	1.ØØ	-1.ØØ	-1.ØØ	-1.ØØ	1.ØØ	9.99
1.ØØ	-1.ØØ	Ø.ØØ	1.ØØ	1.ØØ	1.ØØ	1.ØØ	9.99
1.ØØ	1.ØØ	1.ØØ	Ø.ØØ	1.ØØ	1.ØØ	-1.ØØ	9.99
-1.ØØ	1.ØØ	-1.ØØ	-2.ØØ	Ø.ØØ	1.ØØ	1.ØØ	9.99
1.ØØ	-1.ØØ	1.ØØ	1.ØØ	-1.ØØ	Ø.ØØ	1.ØØ	9.99
Ø.ØØ	1.ØØ	1.ØØ	-1.ØØ	1.ØØ	1.ØØ	Ø.ØØ	9.99
9.99	9.99	9.99	9.99	9.99	9.99	9.99	9.99

LDA = 8

N = 7

OUTPUT:

Output matrix A:

-2.ØØ	Ø.ØØ	1.ØØ	-1.ØØ	-1.ØØ	-1.ØØ	1.ØØ	9.99
Ø.5Ø	-1.ØØ	-Ø.5Ø	Ø.5Ø	Ø.5Ø	Ø.5Ø	1.5Ø	9.99
Ø.5Ø	-1.ØØ	-2.ØØ	-1.ØØ	1.ØØ	2.ØØ	2.ØØ	9.99
Ø.5Ø	1.ØØ	Ø.5Ø	-1.ØØ	-1.ØØ	1.ØØ	2.ØØ	9.99
-Ø.5Ø	1.ØØ	Ø.5Ø	-Ø.5Ø	2.5Ø	1.25	Ø.5Ø	9.99
Ø.5Ø	-1.ØØ	1.ØØ	-Ø.5Ø	-Ø.8Ø	Ø.5Ø	Ø.6Ø	9.99
Ø.ØØ	1.ØØ	Ø.25	-Ø.75	-Ø.4Ø	Ø.ØØ	-Ø.2Ø	9.99
9.99	9.99	9.99	9.99	9.99	9.99	9.99	9.99

IPVT = 2 2 5 6 7 6 7

INFO = Ø

In terms of the input parameters, the computation performed by PTSLVK can be described by the following equation, where V(I,J) represents element I in the J-th vector memory:

```
V(IEND,J) = [ V(IEND,J) + s*T(1)*V(IBEG,J) +
              s*T(1+I)*V(IBEG+1,J) + ... +
              T(I*(IEND-IBEG-1)+1)*V(IEND-1,J) ] * Q

where    s = +1.Ø             (when IFUN =  Ø)
           = -1.Ø             (when IFUN <> Ø)

and      Q = 1.Ø                   (if IUD =  Ø)
           = T(I*(IEND-IBEG)+1) (if IUD <> Ø)
```

J ranges from 1 to the number of vectors available. For example, with N MAX modules, 8*N+4 values of V(IEND,J) would be computed on each call to PTSLVK.

Implementation note:

When the user's data structure is in the form of a 2-D FORTRAN array, the higher level routine PTSOLV should be called to solve for the entire solution matrix X with a single call. PTSLVK may also be called directly by the user to handle matrix data structures that are not in this form.

Summary of error conditions:

IERR = Ø   Normal return.

IERR = 1   Insufficient MAX vector memories and TMRAM to hold the number of vectors designated by NV. Each MAX module can hold eight vectors. TMRAM can hold four vectors.

IERR = 2   Invalid vector specification. (IEND-IBEG) must be greater than or equal to zero, and less than 2Ø48.

IERR = 3   IBEG or IEND M is less than or equal to zero. Both of these parameters must be positive.

IERR = 4   Insufficient TMRAM space available. There must be enough TMRAM available to hold 4*(IEND-IBEG+1) + 17 words, starting at the TMRAM workspace base address, ITMA. Although PTSLVK does not load TMRAM, it does check for consistency between IEND, IBEG, and ITMA.

```
TM(24559):      MAX
     .
     .          Configuration
     .
TM(24575):      Table
```

Given the following input parameters to PTSLVK:

```
I      =    1
IBEG   =    1
IEND   =    4
NV     =   14
IUD    =    1
IFUN   =    1
ITMA   = 8192

T =  1.Ø  1.Ø  1.Ø -1.Ø
```

Upon RETURN from PTSLVK, the changed vector memories and their values would be:

<u>MAX module #1</u>

		Element Number 4
Vector memory	A:	6.2
	B:	8.2
	C:	1Ø.2
	D:	12.2
	E:	14.2
	F:	16.2
	G:	18.2
	H:	2Ø.2

<u>MAX module #2</u>

		Element Number 4
Vector memory	A:	2Ø.2
	B:	22.2
	C:	24.2
	D:	26.2

<u>TMRAM</u>

```
TM (82Ø4):  2.2
TM (82Ø5):  4.2
```

DESCRIPTION: PTSOLV should be used to solve for the unknown matrix X in the matrix equations TX = B or XT = B (where T is upper or lower triangular), whenever the matrices are in the form of a 2-D FORTRAN array. For other data structures, routine PTSLVK can be used in conjunction with PLOADD and PUNLDD.

The matrix B is first loaded into the MAX and TMRAM vector memories, and is overwritten and reused as the forward elimination or back substitution process continues (termed growing the solution matrix).

In terms of the input parameters, the computation performed by PTSOLV can be described by the equations below, which applies to solving TX = B when T is lower triangular. V(I,J) represents element I in the J-th vector memory. Also, T(I,I) has been reciprocated when IUD<>Ø.

```
V(I,J) = [ V(I,J) +
           s*T(I,1)*V(1,J) + s*T(I,2)*V(2,J) + ... +
           s*T(I,I-1)*V(I-1,J) ] * Q

where    s = +1.Ø         (when IFUN =  Ø)
           = -1.Ø         (when IFUN <> Ø)

and      Q = 1            (if IUD =  Ø)
           = T(I,I)       (if IUD <> Ø)

and      I = 1,2,...,N
         J = 1,2,...,(8*NMAXM+4),
```

where NMAXM = number of MAX modules available. When the solution is completed, matrix V is copied from MAX and TMRAM memory to the solution matrix X.

Summary of error conditions:

IERR = Ø   Normal return.

IERR = 1   One or more of N, LT, LB, or LX is less than or equal to zero. Each of these parameters must be positive.

IERR = 2   ITMA less than 8192. ITMA must be greater than or equal to 8192.

IERR = 3   Insufficient TMRAM space available. There must be enough TMRAM available to hold 4*N + 17 words, starting at the TMRAM workspace base address, ITMA.

```
**********                                                  **********
*        *                                                  *        *
* PUNLDD *        --- PARALLEL UNLOAD FOR PTSLVK ---        * PUNLDD *
*        *                                                  *        *
**********                                                  **********
```

PURPOSE: To unload a set of vectors from TMRAM and the MAX vector memories after calls to PTSLVK.

CALL FORMAT: CALL PUNLDD (A, I, J, N, M, ITMA, ISTART, IERR)

PARAMETERS:

A = Floating-point output matrix into which TMRAM and the MAX vector memories are unloaded.

I = Integer input element stride for vectors in A.

J = Integer input element stride between vectors in A.

N = Integer input number of elements per vector.

M = Integer input number of vectors to unload.

ITMA = Integer input base address of the TMRAM workspace.

ISTART = Integer input starting index into the TMRAM workspace and the MAX vector memories to begin loading vectors. The first location of the TMRAM workspace and the MAX vector memories has an index value equal to one.

IERR = Integer output error flag. See "Summary of Error Conditions" below.

DESCRIPTION: This routine performs the reverse of routine PLOADD, unloading the vectors contained in TMRAM and the MAX vector memories into A. Vectors are unloaded with the constraints that TMRAM is unloaded first, and the MAX is unloaded with multiples of four vectors. PUNLDD assumes that PLOADD has set up the MAX configuration table in the high end of TMRAM.

Summary of Error Conditions:

IERR = Ø Normal return.

IERR = 1 Insufficient TMRAM and MAX vector memories to hold the number of vectors designated by M. Each MAX module can hold eight vectors. TMRAM can hold four vectors.

IERR = 2 Vector length too large. N+ISTART-1 must be less than or equal to 2Ø47.

IERR = 3 One or more of ISTART, N, or M is less than or equal to zero. Each of these parameters must be positive.

```
                xx.x
                 1.2
                 2.2
          .     xx.x
          .     xx.x
          .      1.3
                 2.3
                xx.x
                xx.x
                 1.4
TM (8205):       2.4
TM (8206):      xx.x
          .
          .
          .
TM(24558):      xx.x
------------------------------
TM(24559):           MAX
          .
          .          Configuration
          .
TM(24575):           Table
```

Given the following input parameters to PUNLDD:

```
N      =    4
M      =   14
I      =   14
J      =    1
ITMA   = 8192
ISTART =    1
```

Upon RETURN from PUNLDD, Main Memory contains:

```
A =  1.1  1.2  1.3  1.4
     2.1  2.2  2.3  2.4
     3.1  3.2  3.3  3.4
     4.1  4.2  4.3  4.4
     5.1  5.2  5.3  5.4
     6.1  6.2  6.3  6.4
     7.1  7.2  7.3  7.4
     8.1  8.2  8.3  8.4
     9.1  9.2  9.3  9.4
    10.1 10.2 10.3 10.4
    11.1 11.2 11.3 11.4
    12.1 12.2 12.3 12.4
    13.1 13.2 13.3 13.4
    14.1 14.2 14.3 14.4
```

Note that in this example, A is a two-dimensional array whose elements are stored in column major order.

IERR	= 1	Insufficient MAX vector memories and TMRAM to hold the number of vectors designated by M. Each MAX module can hold four vectors. TMRAM can hold one vector.
IERR	= 2	Vector length too large. N+ISTART-1 must be less than or equal to 2Ø47.
IERR	= 3	One or more of ISTART, N, or M is less than or equal to zero. Each of these parameters must be positive.
IERR	= 4	Insufficient TMRAM space available. There must be enough TMRAM available to hold 2*(N+ISTART-1) + 17 words, starting at the TMRAM workspace base address, ITMA. Although PUNLDV does not load TMRAM, it does check for consistency between N, ISTART, and ITMA.
IERR	= 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

EXAMPLE: Given a system with two available MAX modules and 16K TMRAM that contains the following:

MAX module #1

```
Vector memory A:  Ø.9  Ø.8  Ø.7  Ø.6
              B:  1.8  1.6  1.4  1.2
              C:  2.7  2.4  2.1  1.8
              D:  3.6  3.2  2.8  2.4
```

MAX module #2

```
Vector memory A:  4.5  4.Ø  3.5  3.Ø
              B:  5.4  4.8  4.2  3.6
```

TMRAM

```
TM (8192):  Ø.Ø
TM (8193):  Ø.Ø
TM (8194):  Ø.Ø     First
TM (8195):  Ø.Ø
TM (8196):  x.x     TMRAM
     .
     .              Region
     .
TM(16374):  x.x
```

```
**********                                      **********
*        *                                      *        *
* PVMSA  *         --- PARALLEL VMSA ---        * PVMSA  *
*        *                                      *        *
**********                                      **********
```

PURPOSE: To compute the vector multiply and scalar add (VMSA) of a single vector with a set of vectors residing in TMRAM and the MAX vector memories.

CALL FORMAT:
```
CALL PVMSA(B, K, S, J, N, M, IBANK, ITMA, ISTART,
           IFUN, IERR)
```

PARAMETERS:
```
B      = Floating-point input vector.
K      = Integer input element stride for B.
S      = Floating-point input array of scalars.
J      = Integer input element stride for S.
N      = Integer input number of elements per vector.
M      = Integer input number of VMSA's to perform.
IBANK  = Integer input TMRAM region and bank of MAX
         vector memories containing the input set of
         vectors.  Integer output TMRAM region and
         bank of MAX vector memories containing the
         output set of vectors.
         IBANK =  Ø:  Reference the first TMRAM
                      region and MAX vector
                      memories A,B,C,D.
         IBANK <> Ø:  Reference the second TMRAM
                      region and MAX vector memories
                      E,F,G,H.
ITMA   = Integer input TMRAM workspace base address
         from the most recent call to PLOADV.
ISTART = Integer input starting index into the
         TMRAM region and the MAX vector memories
         to begin accessing/storing vector elements.
         The first location of the TMRAM region and
         the MAX vector memories has an index of one.
IFUN   = Integer input addition/subtraction flag.
         IFUN =  Ø:  Use addition.
         IFUN <> Ø:  Use subtraction.
IERR   = Integer output error flag.
```

DESCRIPTION: PVMSA computes the M VMSA's of the vector B with the set of M vectors residing in TMRAM and the MAX vector memories, using the elements of S as the M scalar values. The results are stored in the other region of TMRAM and bank of MAX vector memories as indicated by toggling the value of IBANK.

EXAMPLE: Given a system with two available MAX modules and 16K TMRAM that contains the following:

MAX module #1

```
    Vector memory E:  3.1  3.2  3.3  3.4
                  F:  5.1  5.2  5.3  5.4
                  G:  7.1  7.2  7.3  7.4
                  H:  9.1  9.2  9.3  9.4
```

MAX module #2

```
    Vector memory E: 11.1 11.2 11.3 11.4
                  F: 13.1 13.2 13.3 13.4
```

TMRAM

```
    TM (8192):  x.x     First
         .
         .              TMRAM
         .
    TM(16374):  x.x     Region
    ---------------------------------
    TM(16375):  1.1
    TM(16376):  1.2
    TM(16377):  1.3     Second
    TM(16378):  1.4
    TM(16379):  x.x     TMRAM
         .
         .              Region
         .
    TM(24558):  x.x
    ---------------------------------
    TM(24559):          MAX
         .
         .              Configuration
         .
    TM(24575):          Table
```

Given the following input parameters to PVMSA:

```
    B =  -1.1  -1.2  -1.3  -1.4

    K = 1

    S =   1.Ø
          2.Ø
          3.Ø
          4.Ø
          5.Ø
          6.Ø
          7.Ø
```

TM(24559):	MAX
.	
.	Configuration
.	
TM(24575):	Table

```
IF(IBANK .EQ. Ø) IBANK = 1
ELSE IBANK = Ø

V(ISTART+i,j) = W(ISTART+i,j) +s*B(i+1)*S(j);

                i = Ø,N-1;
                j = 1,M
```

where

```
s =  1.Ø if IFUN =  Ø
s = -1.Ø if IFUN <> Ø
```

and W(*,1:M) are the input set of vectors and V(*,1:M) are the output set of vectors.

PUNLDV is called by the user to retrieve the resulting vectors V(*,1:M), when appropriate.

Summary of error conditions:

IERR = Ø	Normal return.
IERR = 1	Insufficient MAX vector memories and TMRAM to hold the number of vectors designated by M. Each MAX module can hold four vectors. TMRAM can hold one vector.
IERR = 2	Vector length too large. N+ISTART-1 must be less than or equal to 2Ø47.
IERR = 3	One or more of ISTART, N, or M is less than or equal to zero. Each of these parameters must be positive.
IERR = 4	Insufficient TMRAM space available. There must be enough TMRAM available to hold 2*(N+ISTART-1) + 17 words, starting at the TMRAM workspace base address, ITMA. Although PVSMA does not load TMRAM, it does check for consistency between N, ISTART, and ITMA.
IERR = 5	ITMA less than 8192. ITMA must be greater than or equal to 8192.

```
J      =    1
N      =    4
M      =    7
IBANK  =    1
ITMA   = 8192
ISTART =    1
IFUN   =    Ø
```

Upon RETURN from PVSMA, IBANK is equal to zero, and the MAX modules and TMRAM contain:

MAX module #1

```
Vector memory A:  Ø.9  Ø.8  Ø.7  Ø.6
              B:  1.8  1.6  1.4  1.2
              C:  2.7  2.4  2.1  1.8
              D:  3.6  3.2  2.8  2.4

              E:  3.1  3.2  3.3  3.4
              F:  5.1  5.2  5.3  5.4
              G:  7.1  7.2  7.3  7.4
              H:  9.1  9.2  9.3  9.4
```

MAX module #2

```
Vector memory A:  4.5  4.Ø  3.5  3.Ø
              B:  5.4  4.8  4.2  3.6

              E: 11.1 11.2 11.3 11.4
              F: 13.1 13.2 13.3 13.4
```

TMRAM

```
TM (8192):  Ø.Ø
TM (8193):  Ø.Ø
TM (8194):  Ø.Ø     First
TM (8195):  Ø.Ø
TM (8196):  x.x     TMRAM
     .
     .              Partition
     .
TM(16374):  x.x
-----------------------------------
TM(16375):  1.1
TM(16376):  1.2
TM(16377):  1.3     Second
TM(16378):  1.4
TM(16379):  x.x     TMRAM
     .
     .              Partition
     .
TM(24558):  x.x
-----------------------------------
```

K.4 MATRIX ORIENTED MAX ROUTINES

The Matrix Oriented MAX routines emphasize matrix processing and provide for expanded utilization of the MAX system.

K.4.1 Overview

The Matrix Oriented MAX routines were written to enhance the basic functionality of the MAX modules. These routines are more flexible than the Basic MAX routines in terms of data management in the MAX system. They are also more efficient in performing matrix-matrix operations, particularly for small matrices.

There are two major differences between the Matrix Oriented MAX routines and the Basic MAX routines, resulting in increased performance and improved user interfaces. First, the Matrix Oriented MAX routines accept submatrices as input. Second, they present a familiar, consistent model of the MAX vector memories.

A minor difference between the Basic MAX routines and the Matrix Oriented MAX routines is that the Matrix Oriented MAX routines check for the correct number of parameters and exit with no action if the check fails. If the parameter check succeeds, the IERR flag will be set to a code upon exit from the routine. An incorrect number of parameters can be detected by setting IERR to an unused error code, e.g., 999, before calling a Matrix Oriented MAX routine. If IERR is unchanged when the routine exits, then there are an incorrect number of parameters.

The configuration table used by the Matrix Oriented MAX routines is used to reference the available MAX modules. Because the Matrix Oriented MAX routines permit operations with individual vector memories, the configuration table is more extensive than the one used by the Basic MAX routines. The table is always placed into the 254 highest addressable locations in TMRAM regardless of the base address of the TMRAM workspace. Hence, for 16K TMRAM systems, the table will be situated in locations 24322-24575. Similarly, for 32K TMRAM systems, the table will be situated in locations 40706-40959. The rest of the TMRAM workspace is used to store input/output vectors.

K.4.1.1 Submatrix Input

All floating-point data passed to the Matrix Oriented MAX routines can be organized as submatrices. This results in improved performance for small matrices. With a single call to a Matrix Oriented MAX routine, computations can be performed on an entire submatrix.

Each submatrix is defined by a starting element, an intra-vector element stride, an inter-vector element stride, the number of elements per vector, and the number of vectors. A single vector is a degenerate case of a submatrix, where the number of vectors is one. A single element is also a degenerate submatrix, where both the number of

If VECMEM is interpreted as a matrix of column vectors, then IVM is the starting row, IVN is the starting column, NEV is the number of rows, and NVB is the number of columns as shown in Figure K-2.

```
+---+---+---+---+---+---+---+---+---+---+---+---+--
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   | x | x | x | x |   |   |   |   |   |   |  <-- IVM
|   |   | x | x | x | x |   |   |   |   |   |   |
|   |   | x | x | x | x |   |   |   |   |   |   |
|   |   | x | x | x | x |   |   |   |   |   |   |
|   |   | x | x | x | x |   |   |   |   |   |   |
|   |   | x | x | x | x |   |   |   |   |   |   |  <-- IVM+NEV-1
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |

          ^           ^
          |           |

         IVN      IVN+NVB-1
```

Figure K-2 MAX Vector Memory Submatrix

There are several advantages to using this model with the Matrix Oriented MAX routines.

- The user is insulated from the hardware details of the machine and can use familiar concepts to visualize the operations; thus the routines are easy to use.

- All the routines permit operations with a single element, a single vector, or a submatrix in the vector memories; thus they are general.

- In most cases, results from one routine can be used as input to another routine without moving data in the vector memories; thus the data management is consistent.

K.4.2.1 Real Vector Mapping

A set of real vectors is mapped one-to-one to the vector memories. The first four vectors are mapped to the A, B, C, D vector memories of the lowest addressed MAX module. The second four vectors are mapped to the A, B, C, D vector memories of the next lowest addressed MAX module and so on, until there are no more available MAX modules. The next two vectors are mapped to the A and B vector memories simulated in TMRAM. The next four vectors are mapped to the E, F, G, H vector memories of the lowest addressed MAX module. The next four vectors are mapped to the E, F, G, H vector memories of the next lowest addressed MAX module and so on, until as before, there are no more available MAX modules. The last two vectors are mapped to the C and D vector memories simulated in TMRAM.

The vector memories simulated in TMRAM begin at the address ITMA, and are interleaved as shown in Figure K-3.

```
   TMRAM        Address

+----------+
|   A(1)   |    ITMA
+----------+
|   B(1)   |    ITMA+1
+----------+
|   C(1)   |    ITMA+2
+----------+
|   D(1)   |    ITMA+3
+----------+
|   A(2)   |    ITMA+4
+----------+
|   B(2)   |    ITMA+5
+----------+
|   C(2)   |    ITMA+6
+----------+
|   D(2)   |    ITMA+7
+----------+
|          |
     .            .
     .            .
     .            .
```

Figure K-3 TMRAM Simulated Vector Memories

K.4.2.2 Complex Vector Mapping

Mapping a set of complex vectors to the MAX vector memories is similar to the real vector mapping, except that a pair of vector memories is needed for each complex vector. The real part of each complex vector is mapped to the first vector memory of a pair. The imaginary part of a complex vector is mapped to the second vector memory of a pair.

The first two vectors in the set are mapped to the A, B, C, D vector memories of the lowest addressed MAX module. The second two vectors are mapped to the A, B, C, D vector memories of the next lowest addressed MAX module, and so on, until there are no more available MAX modules. The next vector is mapped to the A and B vector memories simulated in TMRAM. The next two vectors are mapped to the E, F, G, H vector memories of the lowest addressed MAX module. The next two vectors are mapped to the E, F, G, H vector memories of the next lowest addressed MAX module, and so on, until as before, there are no more available MAX modules. The last vector is mapped to the C and D vector memories simulated in TMRAM.

K.4.3 Examples

The following examples all involve real data. The applicable complex counterparts are similar. The major difference between the complex cases and the real is that the complex case has one-half the available vector memories as the real case.

Assume for the purpose of these examples that:

- There is one available MAX module. From Section K.2, the number of vector memories is given by

  ```
  NVEC = 8 * 1 + 4 = 12
  ```

  As in Section K.4.1.2, the vector memories will be modeled as a matrix of column vectors represented by a FORTRAN matrix VECMEM declared as follows:

  ```
  REAL VECMEM(2048,NVEC)
  ```

  Note again that VECMEM is never actually declared in any of the code and is used solely to model operations performed with the MAX vector memories.

- The matrix B, denoted by [B], is declared as

  ```
  REAL B (3, 4)
  ```

  and is initialized as follows:

  ```
  B:  1.1   1.2   1.3   1.4
      2.1   2.2   2.3   2.4
      3.1   3.2   3.3   3.4
  ```

- The matrix A, denoted by [A], is declared as

  ```
  REAL A (2, 3)
  ```

  and is initialized as follows:

  ```
  A:  11.1  11.2  11.3
      12.1  12.2  12.3
  ```

- The matrix C, denoted by [C], is declared as

  ```
  REAL C (2, 4)
  ```

The examples in Sections K.4.3.1 through K.4.3.6 illustrate how various data structures can be moved to and from the MAX vector memories. The examples in Sections K.4.3.7 and K.4.3.8 illustrate how dot products can be performed between various data structures. The examples in Sections K.4.3.9 and K.4.3.10 illustrate how VSMA's can be performed between various data structures.

K.4.3.1 Matrix Load

[B] can be loaded into the vector memories by a single call to MLOAD. If input parameters to MLOAD are set up as follows:

```
IBE   = 1
IBV   = 3
IVM   = 1
IVN   = 1
NEV   = 3
NVB   = 4
ITMA  = 8192
```

then the sequence

```
      IERR = 999
      CALL MLOAD (B, IBE, IBV, IVM, IVN, NEV, NVB, ITMA, IERR)
      IF (IERR .NE. Ø) THEN
        WRITE (6, 1ØØØ) IERR
 1ØØØ   FORMAT (1X, 'MLOAD ERROR = ',I4)
      ENDIF
```

will load [B] by columns into the vector starting at the first row and first column of VECMEM, and will also check to make sure that MLOAD did not detect an error. After this sequence the vector memories will contain the following:

MAX vector memories (VECMEM):

```
1.1  1.2  1.3  1.4  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
2.1  2.2  2.3  2.4  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
3.1  3.2  3.3  3.4  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

K.4.3.3 Single Column Load

A single column of [B] can be loaded into a vector memory by a single call to MLOAD. To load column 2 into vector memory 3, the parameters are set up as follows:

```
IBE   = 1
IBV   = 3
IVM   = 1
IVN   = 3
NEV   = 3
NVB   = 1
ITMA  = 8192
```

The sequence

```
      IERR = 999
      CALL MLOAD (B(1,2), IBE, IBV, IVM, IVN, NEV, NVB, ITMA, IERR)
      IF (IERR .NE. Ø) THEN
         WRITE (6, 1ØØØ) IERR
 1ØØØ    FORMAT (1X, 'MLOAD ERROR = ',I4)
      ENDIF
```

leaves the vector memories as follows:

```
MAX vector memories (VECMEM):

x.x  x.x  1.2  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  2.2  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  3.2  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

K.4.3.5 Single Element Load

A single element of [B] can be loaded into an element of a vector memory by a call to MLOAD. To load B(1,2) into the vector memories at row 4 and column 3, the parameters are set up as follows:

```
IBE   = 1
IBV   = 1
IVM   = 4
IVN   = 3
NEV   = 1
NVB   = 1
ITMA  = 8192
```

The sequence

```
         IERR = 999
         CALL MLOAD (B(1,2), IBE, IBV, IVM, IVN, NEV, NVB, ITMA, IERR)
         IF (IERR .NE. Ø) THEN
            WRITE (6, 1ØØØ) IERR
    1ØØØ    FORMAT (1X, 'MLOAD ERROR = ',I4)
         ENDIF
```

leaves the vector memories as follows:

MAX vector memories (VECMEM):

```
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  1.2  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
 .    .    .    .    .    .    .    .    .    .    .    .
x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

K.4.3.7 Matrix Multiplication: [A]*[B] (Dot Products)

To illustrate the flexibility of the Matrix Oriented MAX routines, two examples of matrix multiplication using dot product operations are given below.

Matrix multiplication using dot products can be performed using the MAX in two different ways: either load [B] into the vector memories or load [A] into the vector memories. The preferred method depends on the dimensions of the matrices.

The matrix loaded should maximize three parameters: reuse, vector length, and fit. Reuse is the number of times each element of data loaded into a vector memory is used in subsequent computations. Vector length is the number of elements used per vector operation. Fit is a measure of the average number of vector memories used per vector operation.

Note that by setting the strides and counts appropriately, [A]T*[B] or [A]*[B]T or [A]T*[B]T can also be performed.

The first example loads columns of [B] into the vector memories. The sequence

```
      IBE   = 1
      IBV   = 3
      IVM   = 1
      IVN   = 1
      NEV   = 3
      NVB   = 4
      ITMA  = 8192
      IERR  = 999
      CALL MLOAD (B, IBE, IBV, IVM, IVN, NEV, NVB, ITMA, IERR)
      IF (IERR .NE. Ø) THEN
         WRITE (6,1ØØØ) IERR
 1ØØØ    FORMAT (1X, 'MLOAD ERROR = ',I4)
      ELSE
         IAE  = 2
         IAV  = 1
         ICE  = 2
         ICV  = 1
         NVA  = 2
         IFUN = Ø
         IERR = 999
         CALL MDOT (A, IAE, IAV, IVM, IVN, C, ICE, ICV,
                    NEV, NVA, NVB, IFUN, IERR)
         IF (IERR .NE. Ø) THEN
            WRITE (6,1ØØ1) IERR
 1ØØ1       FORMAT (1X, 'MDOT ERROR = ',I4)
         ENDIF
      ENDIF
```

K.4.3.8 Vector Dot Product

By restricting the vector count of one of the matrices to one, the dot products between that vector and the other matrix can be calculated. This mode of operation is similar to the functionality of the Basic MAX routines. This example performs the dot products between the second row of [A] and all the columns of [B]. Note that by simply changing the element strides for [A], a column of [A] could be used instead. The sequence

```
      IBE   = 1
      IBV   = 3
      IVM   = 1
      IVN   = 1
      NEV   = 3
      NVB   = 4
      ITMA  = 8192
      IERR  = 999
      CALL MLOAD (B, IBE, IBV, IVM, IVN, NEV, NVB, ITMA, IERR)
      IF (IERR .NE. Ø) THEN
         WRITE (6,1ØØØ) IERR
1ØØØ     FORMAT (1X, 'MLOAD ERROR = ',I4)
      ELSE
         IAE  = 2
         IAV  = 1
         ICE  = 2
         ICV  = 1
         NVA  = 1
         IFUN = Ø
         IERR = 999
         CALL MDOT (A(2,1), IAE, IAV, IVM, IVN, C, ICE, ICV,
                   NEV, NVA, NVB, IFUN, IERR)
         IF (IERR .NE. Ø) THEN
            WRITE (6,1ØØ1) IERR
 1ØØ1       FORMAT (1X, 'MDOT ERROR = ',I4)
         ENDIF
      ENDIF
```

writes the results into [C] as follows:

```
C:  77.Ø6  8Ø.72  84.38  88.Ø4
     x.x    x.x    x.x    x.x
```

The first example clears the appropriate submatrix of the vector memories, uses [B] as the scalars, and columns of [A] as the vectors for VSMA operations.

The sequence

```
        IVM  = 1
        IVN  = 1
        NEV  = 2
        NVB  = 4
        ITMA = 8192
        IERR = 999
        CALL MLOAD (Ø.Ø, Ø, Ø, IVM, IVN, NEV, NVB, ITMA, IERR)
        IF (IERR .NE. Ø) THEN
           WRITE (6,1ØØØ) IERR
   1ØØØ    FORMAT (1X, 'MLOAD ERROR = ',I4)
        ELSE
           IAE  = 1
           IAV  = 2
           IBE  = 3
           IBV  = 1
           NVA  = 3
           IFUN = Ø
           IERR = 999
           CALL MVSMA (A, IAE, IAV, IVM, IVN, B, IBE, IBV,
                     NEV, NVA, NVB, IFUN, IERR)
           IF (IERR .NE. Ø) THEN
              WRITE (6,1ØØ1) IERR
    1ØØ1      FORMAT (1X, 'MVSMA ERROR = ',I4)
           ELSE
              ICE  =   1
              ICV  =   2
              IERR = 999
              CALL MUNLD (IVM, IVN, C, ICE, ICV, NEV, NVB, IERR)
              IF (IERR .NE. Ø) THEN
                 WRITE (6,1ØØ2) IERR
    1ØØ2         FORMAT (1X, 'MUNLD ERROR = ',I4)
              ENDIF
           ENDIF
        ENDIF
```

writes the results into [C] as follows:

```
        C:  7Ø.76  74.12  77.48  8Ø.84
            77.Ø6  8Ø.72  84.38  88.Ø4
```

Notice the call to MVSMA changes the value of IVN, meaning that the column of VECMEM containing the results is not the same as the column of VECMEM containing the inputs. Since MUNLD can unload from any column, it receives the modified IVN value for copying results out to C.

K.4.3.1Ø Vector VSMA's

Just as the dot product routines can be used to compute vector-matrix dot products, the VSMA/VMSA routines can also be used to compute VSMA/VMSA vector-matrix operations. This example uses the second column of [A] as the scalars, the third row of [B] as the vector, and [C] as the matrix. Assume that [C] is initialized as follows:

```
        C:  5.1   5.2   5.3   5.4
            6.1   6.2   6.3   6.4
```

The sequence

```
        ICE  = 1
        ICV  = 2
        IVM  = 1
        IVN  = 1
        NEV  = 2
        NVB  = 4
        ITMA = 8192
        IERR = 999
        CALL MLOAD (C, ICE, ICV, IVM, IVN, NEV, NVB, ITMA, IERR)
        IF (IERR .NE. Ø) THEN
           WRITE (6,1ØØØ) IERR
    1ØØØ   FORMAT (1X, 'MLOAD ERROR = ',I4)
        ELSE
           IAE  = 1
           IAV  = 2
           IBE  = 3
           IBV  = 1
           NVA  = 1
           IFUN = Ø
           IERR = 999
           CALL MVSMA (A(1,2), IAE, IAV, IVM, IVN, B(3,1), IBE, IBV,
                       NEV, NVA, NVB, IFUN, IERR)
           IF (IERR .NE. Ø) THEN
              WRITE (6,1ØØ1) IERR
    1ØØ1      FORMAT (1X, 'MVSMA ERROR = ',I4)
           ELSE
              IERR = 999
              CALL MUNLD (IVM, IVN, C, ICE, ICV, NEV, NVB, IERR)
              IF (IERR .NE. Ø) THEN
                 WRITE (6,1ØØ2) IERR
    1ØØ2         FORMAT (1X, 'MUNLD ERROR = ',I4)
              ENDIF
           ENDIF
        ENDIF
```

writes the results into [C] as follows:

```
        C:  39.82  41.Ø4  42.26  43.48
            43.92  45.24  46.56  47.88
```

MATRIX ORIENTED MAX ROUTINES

associated with A and B. The operation of CMDOT can be conveniently described if the set of MAX vector memories is considered to be a complex matrix VECMEM, and A and C are also considered to be complex matrices. Using this matrix convention, CMDOT performs the matrix multiplication

```
C  =  q * C  +  r * A * VECMEM
```

where IAE, IAV, IVM, IVN, ICE, ICV, NEV, NVA, and NVB allow the user to select subsets of A, VECMEM, and C as appropriate.

To illustrate the flexibility of the data structures that can be associated with the data, suppose A and C are one-dimensional complex arrays, and that VECMEM is a complex matrix VECMEM(2Ø48,4*NMAX+2) where NMAX is the number of available MAX modules. In this case, the computations performed by CMDOT can be described in FORTRAN by:

```
   DO 3Ø i = 1, NVA
     DO 2Ø j = 1, NVB
       TEMP = CMPLX(Ø.Ø,Ø.Ø)
       DO 1Ø k = 1, NEV
         TEMP  =  TEMP  +  r * A((i-1)*IAV+(k-1)*IAE+1) *
  +                             VECMEM(IVM+k-1,IVN+j-1)
1Ø     CONTINUE
       C((i-1)*ICE+(j-1)*ICV+1)  =  TEMP  +
  +                      q * C((i-1)*ICE+(j-1)*ICV+1)
2Ø   CONTINUE
3Ø CONTINUE
```

Care should be taken if A and C overlap. There are no checks that ensure the integrity of A is maintained, so values of A could be overwritten before they are used in subsequent calculations.

Summary of error conditions:

IERR = -3	NVA <= Ø.
IERR = -2	NVB <= Ø.
IERR = -1	NEV <= Ø. C is cleared when IFUN = Ø or IFUN = 1.
IERR = Ø	No error occurred. Normal completion.
IERR = 1	No TMRAM on the system.
IERR = 4	IVM <= Ø.
IERR = 5	IVM + NEV - 1 greater than 2Ø47.

```
IVM =  3
IVN =  2
NVB =  5

Input C:

  (1.Ø,1.Ø)
  (2.Ø,2.Ø)
  (3.Ø,3.Ø)
  (4.Ø,4.Ø)
  (5.Ø,5.Ø)
  (1.Ø,1.Ø)
  (1.Ø,1.Ø)
  (2.Ø,2.Ø)
  (3.Ø,3.Ø)
  (4.Ø,4.Ø)
  (1.Ø,1.Ø)
  (1.Ø,1.Ø)
  (1.Ø,1.Ø)
  (2.Ø,2.Ø)
  (3.Ø,3.Ø)

ICE  =   2
ICV  =  1Ø

NEV  =   4

IFUN =   3
IERR = 999

Upon return from CMDOT, C contains:

  ( 5.Ø, -23.Ø)
  ( 7.Ø, -25.Ø)
  (1Ø.Ø, -32.Ø)
  (14.Ø, -46.Ø)
  (19.Ø, -69.Ø)
  ( 5.Ø, -39.Ø)
  ( 6.Ø, -46.Ø)
  ( 9.Ø, -61.Ø)
  (13.Ø, -87.Ø)
  (18.Ø,-126.Ø)
  ( 5.Ø, -55.Ø)
  ( 6.Ø, -66.Ø)
  ( 8.Ø, -9Ø.Ø)
  (12.Ø,-128.Ø)
  (17.Ø,-183.Ø)

IERR = Ø
```

To illustrate the flexibility of the data structures that can be associated with the data, suppose B is a one-dimensional complex array, and that VECMEM is a complex matrix VECMEM(2048,4*NMAX+2) where NMAX is the number of available MAX modules. In this case, the operations performed by CMLOAD can be described in FORTRAN by:

```
   DO 20 i = 1, NVB
     DO 10 j = 1, NEV
       VECMEM(IVM+j-1,IVN+i-1) = B((i-1)*IBV+(j-1)*IBE+1)
10   CONTINUE
20 CONTINUE
```

Summary of error conditions:

IERR = -2	NVB <= 0.
IERR = -1	NEV <= 0.
IERR = 0	No error occurred. Normal completion.
IERR = 1	No TMRAM on the system.
IERR = 2	ITMA <= 8191.
IERR = 3	ITMA > LASTTM-8192-254-4*(IVM+NEV-1) where LASTTM is the highest valid TMRAM address. Refer to Section K.2.
IERR = 4	IVM <= 0.
IERR = 5	IVM + NEV - 1 greater than 2048.
IERR = 6	IVN <= 0.
IERR = 7	IVN + NVB - 1 greater than NMAX*4 + 2 where NMAX is the number of available MAX modules. Refer to Section K.2.

If there are too many or too few formal parameters, then IERR is left unchanged.

For more information on the Matrix Oriented MAX routines, refer to Section K.4.

```
**********                                          **********
*        *                                          *        *
* CMUNLD *       ---- COMPLEX MATRIX UNLOAD ----    * CMUNLD *
*        *                                          *        *
**********                                          **********
```

PURPOSE: To unload a matrix of complex vectors from the MAX vector memories into Main Memory.

CALL FORMAT: CALL CMUNLD (IVM, IVN, B, IBE, IBV, NEV, NVB, IERR)

PARAMETERS:

```
IVM  = Integer input starting element in the MAX
       vector memories.
IVN  = Integer input starting vector in the MAX
       vector memories.
B    = Floating-point output matrix of complex
       vectors in Main Memory.
IBE  = Integer input element stride for vectors in B.
IBV  = Integer input element stride between
       vectors in B.
NEV  = Integer input number of complex elements
       per vector.
NVB  = Integer input number of complex vectors.
IERR = Integer input/output error flag.  See
       DESCRIPTION for a list of error conditions.
```

DESCRIPTION: CMUNLD unloads the NVB vectors contained in the MAX vector memories to Main Memory defined by B, IBE, IBV, and NEV. An arbitrary data structure can be associated with B. The operation of CMUNLD can be conveniently described if the set of MAX vector memories is considered to be a matrix VECMEM and B is also considered to be a matrix. Using this matrix convention, CMUNLD performs the matrix transfer

    B = VECMEM

where IBE, IBV, IVM, IVN, NVB, and NEV allow the user to select subsets of B and VECMEM as appropriate.
To illustrate the flexibility of the data structures that can be associated with the data, suppose B is a one-dimensional array, and that VECMEM is a matrix VECMEM(2Ø48,8*NMAX+4) where NMAX is the number of available MAX modules. In this case, the operations performed by CMUNLD can be described in FORTRAN by:

```
   DO 2Ø i = 1, NVB
     DO 1Ø j = 1, NEV
      B((i-1)*IBV+(j-1)*IBE+1) = VECMEM(IVM+j-1,IVN+i-1)
1Ø   CONTINUE
2Ø CONTINUE
```

Summary of error conditions:

```
IERR = -2    NVB <= Ø.
IERR = -1    NEV <= Ø.
IERR =  Ø    No error occurred.  Normal completion.
```

```
**********                                                **********
*        *                                                *        *
* CMVSMA *    ---- COMPLEX VECTOR MULTIPLY SCALAR ADD ----  * CMVSMA *
*        *                                                *        *
**********                                                **********
```

```
PURPOSE:      To perform complex vector scalar multiply add (CVSMA)
              operations between a matrix of vectors in Main Memory,
              a matrix of vectors in the MAX vector memories, and
              a matrix of scalars in Main Memory.

CALL FORMAT: CALL CMVSMA (A, IAE, IAV, IVM, IVN,  C, ICE, ICV,
                          NEV, NVA, NVB, IFUN, IERR)

PARAMETERS:  A    = Floating-point input matrix of complex vectors
                    in Main Memory.
             IAE  = Integer input element stride for vectors in A.
             IAV  = Integer input element stride between vectors
                    in A.
             IVM  = Integer input starting element in the MAX
                    vector memories.
             IVN  = Integer input/output starting vector for the
                    input/output vectors in the MAX vector memories.
             C    = Floating-point input matrix of scalars in Main
                    Memory.
             ICE  = Integer input element stride for C.
             ICV  = Integer input element stride between vectors
                    in C.
             NEV  = Integer input CVSMA length.
             NVA  = Integer input number of vectors in A to be used.
             NVB  = Integer input number of vectors in the MAX
                    vector memories to be used as input.  Also
                    the number of scalars per vector of C to be used.
             IFUN = Integer input function flag.
                    IFUN = Ø:  r =  1.Ø
                    IFUN = 1:  r = -1.Ø
                    Note that IFUN is a bit-mapped function flag:
                    IFUN = 2 is equivalent to IFUN = Ø, IFUN = 3
                    is equivalent to IFUN = 1, etc.
                    See DESCRIPTION for the usage of r.
             IERR = Integer input/output error flag.
                    See DESCRIPTION for a list of error conditions.

DESCRIPTION: CMVSMA performs complex VSMA operations of length NEV
             between the NVA vectors in Main Memory defined by A, IAE,
             and IAV with the NVB vectors in the MAX vector memories
             defined by IVM and IVN, using the elements of C in Main
             Memory defined by ICE and ICV as the scale factors.  The
             output of the VSMA operations is written into the MAX
             vector memories.
             A system with NMAX available MAX modules has
                 NVEC = 4 * NMAX + 2
```

```
        1Ø     CONTINUE
        2Ø   CONTINUE
        3Ø CONTINUE
        Summary of error conditions:
            IERR = -3    NVA <= Ø.
            IERR = -2    NVB <= Ø.
            IERR = -1    NEV <= Ø.  C is cleared.
            IERR =  1    No TMRAM on the system.
            IERR =  Ø    No error occurred.  Normal completion.
            IERR =  4    IVM <= Ø.
            IERR =  5    IVM + NEV - 1 greater than 2Ø48.
            IERR =  6    IVN <= Ø.
            IERR =  7    IVN + NVB - 1 greater than NMAX*4.
        If there are too many or too few formal parameters then
        IERR is unchanged and no other action is taken.
        For more information on the Matrix Oriented MAX routines,
        refer to Section K.4.
```

```
EXAMPLE: Assume one available MAX module and that the data in
         A and C is stored in column major order (normal FORTRAN).
         Perform the complex VSMA operations of the rows of a
         submatrix of A with a subset of the MAX vector memories,
         using the scalars contained in the columns of a submatrix
         of C.
         Input matrix A:
           (1.Ø,1.Ø) (1.Ø,1.Ø) (1.Ø,1.Ø) (1.Ø,1.Ø)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (2.Ø,2.Ø) (1.Ø,1.Ø) (1.Ø,1.Ø) (1.Ø,1.Ø)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
         IAE  = 12
         IAV  =  4
         NVA  =  2
         Input matrix C:
           (1.Ø,Ø.Ø) (x.x,x.x) (2.Ø,Ø.Ø) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (3.Ø,Ø.Ø) (x.x,x.x) (1.Ø,Ø.Ø) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
         ICE  =  6
         ICV  = 28
         MAX vector memories (VECMEM):
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (1.Ø,1.Ø) (2.Ø,2.Ø) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (1.Ø,1.Ø) (1.Ø,1.Ø) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (1.Ø,1.Ø) (1.Ø,1.Ø) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (1.Ø,1.Ø) (1.Ø,1.Ø) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
           (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x) (x.x,x.x)
               .         .         .         .         .         .
               .         .         .         .         .         .
```

```
**********                                            **********
*        *                                            *        *
*  MDOT  *          --- MATRIX DOT PRODUCT ---        *  MDOT  *
*        *                                            *        *
**********                                            **********
```

PURPOSE: To perform dot products between a matrix of vectors in Main Memory and a matrix of vectors in the MAX vector memories.

CALL FORMAT:

```
CALL MDOT(A, IAE, IAV, IVM, IVN, C, ICE, ICV, NEV,
          NVA, NVB, IFUN, IERR)
```

PARAMETERS:

```
A    = Floating-point input matrix of vectors in
       Main Memory.
IAE  = Integer input element stride for vectors in A.
IAV  = Integer input element stride between
       vectors in A.
IVM  = Integer input starting element in the MAX
       vector memories.
IVN  = Integer input starting vector in the MAX
       vector memories.
C    = Floating-point output matrix of results.
ICE  = Integer input element stride for vectors in C.
ICV  = Integer input element stride between
       vectors in C.
NEV  = Integer input dot product length.
NVA  = Integer input number of vectors in A to be used.
NVB  = Integer input number of vectors in the MAX
       vector memories to be used.
IFUN = Integer input function flag.
       IFUN = Ø:  r =  1.Ø,  q =  Ø.Ø
       IFUN = 1:  r = -1.Ø,  q =  Ø.Ø
       IFUN = 2:  r =  1.Ø,  q =  1.Ø
       IFUN = 3:  r = -1.Ø,  q =  1.Ø
       Note that IFUN is a bit-mapped function flag:
       IFUN = 4 is equivalent to IFUN = Ø; IFUN = 5
       is equivalent to IFUN = 1, etc.
       See DESCRIPTION for the usage of q and r.
IERR = Integer input/output error flag.
       See DESCRIPTION for a list of error conditions.
```

DESCRIPTION: MDOT performs dot products of length NEV between the NVA vectors in Main Memory defined by A, IAE, and IAV with the NVB vectors in the MAX vector memories defined by IVM and IVN. The NVA*NVB results are written to locations in Main Memory defined by C, ICE, and ICV.

Specifically, for each vector in A, MDOT performs the NVB dot products between that vector and the NVB vectors in the MAX vector memories, and then writes the NVB results into C. Arbitrary data structures can be

IERR = 7 IVN + NVB - 1 greater than NMAX*8 + 4 where NMAX is the number of available MAX modules. Refer to Section K.2.

If there are too many or too few formal parameters, then IERR is left unchanged.

For more information on the Matrix Oriented MAX routines, refer to Appendix K.4.

EXAMPLE: Assume one available MAX module, that the MAX vector memories (VECMEM) have been loaded by MLOAD as indicated, and that the data in A is stored in column major order (normal FORTRAN). Multiply the transpose of a submatrix of A by a subset of the MAX vector memories, negate the result and accumulate to C.

Input matrix A:

1.Ø	x.x	x.x	3.Ø	x.x	x.x	5.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
2.Ø	x.x	x.x	4.Ø	x.x	x.x	6.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
2.Ø	x.x	x.x	4.Ø	x.x	x.x	6.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
3.Ø	x.x	x.x	5.Ø	x.x	x.x	7.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
3.Ø	x.x	x.x	5.Ø	x.x	x.x	7.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
4.Ø	x.x	x.x	6.Ø	x.x	x.x	8.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
4.Ø	x.x	x.x	6.Ø	x.x	x.x	8.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
5.Ø	x.x	x.x	7.Ø	x.x	x.x	9.Ø	x.x	x.x
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x

IAE = 2
IAV = 48
NVA = 3

Upon return from MDOT, C contains:

```
 -23.Ø
 -25.Ø
 -32.Ø
 -46.Ø
 -69.Ø
 -39.Ø
 -46.Ø
 -61.Ø
 -87.Ø
-126.Ø
 -55.Ø
 -66.Ø
 -9Ø.Ø
-128.Ø
-183.Ø
```

IERR = Ø

VECMEM(2Ø48,8*NMAX+4) where NMAX is the number of available MAX modules. In this case, the operations performed by MLOAD can be described in FORTRAN by:

```
   DO 2Ø i = 1, NVB
     DO 1Ø j = 1, NEV
       VECMEM(IVM+j-1,IVN+i-1) = B((i-1)*IBV+(j-1)*IBE+1)
1Ø   CONTINUE
2Ø CONTINUE
```

Summary of error conditions:

IERR = -2	NVB <= Ø.
IERR = -1	NEV <= Ø.
IERR = Ø	No error occurred. Normal completion.
IERR = 1	No TMRAM on the system.
IERR = 2	ITMA <= 8191.
IERR = 3	ITMA > LASTTM-8192-254-4*(IVM+NEV-1) where LASTTM is the highest valid TMRAM address. Refer to Section K.2.
IERR = 4	IVM <= Ø.
IERR = 5	IVM + NEV - 1 greater than 2Ø48.
IERR = 6	IVN <= Ø.
IERR = 7	IVN + NVB - 1 greater than NMAX*8 + 4 where NMAX is the number of available MAX modules. Refer to Section K.2.

If there are too many or too few formal parameters, then IERR is left unchanged.

For more information on the Matrix Oriented MAX routines, refer to Section K.4.

EXAMPLE: Assume one available MAX module and that the data in B is stored in column major order (normal FORTRAN). Load the rows of a submatrix of B into a subset of the MAX vector memories.

```
**********                                            **********
*        *                                            *        *
* MUNLD  *           --- MATRIX UNLOAD ---            * MUNLD  *
*        *                                            *        *
**********                                            **********
```

PURPOSE: To unload a matrix of vectors from the MAX vector memories into Main Memory.

CALL FORMAT: CALL MUNLD(IVM, IVN, B, IBE, IBV, NEV, NVB, IERR)

PARAMETERS:
- IVM = Integer input starting element in the MAX vector memories.
- IVN = Integer input starting vector in the MAX vector memories.
- B = Floating-point output matrix of vectors in Main Memory.
- IBE = Integer input element stride for vectors in B.
- IBV = Integer input element stride between vectors in B.
- NEV = Integer input number of elements per vector.
- NVB = Integer input number of vectors.
- IERR = Integer input/output error flag. See DESCRIPTION for a list of error conditions.

DESCRIPTION: MUNLD unloads the NVB vectors contained in the MAX vector memories to Main Memory defined by B, IBE, IBV, and NEV. An arbitrary data structure can be associated with B. The operation of MUNLD can be conveniently described if the set of MAX vector memories is considered to be a matrix VECMEM, and, B is also considered to be a matrix. Using this matrix convention, MLOAD performs the matrix transfer

B = VECMEM

where IBE, IBV, IVM, IVN, NVB, and NEV allow the user to select subsets of B and VECMEM as appropriate.

To illustrate the flexibility of the data structures that can be associated with the data, suppose B is a one-dimensional array, and that VECMEM is a matrix VECMEM(2Ø48,8*NMAX+4) where NMAX is the number of available MAX modules. In this case, the operations performed by MUNLD can be described in FORTRAN by:

```
   DO 2Ø i = 1, NVB
     DO 1Ø j = 1, NEV
       B((i-1)*IBV+(j-1)*IBE+1) = VECMEM(IVM+j-1,IVN+i-1)
1Ø   CONTINUE
2Ø CONTINUE
```

MAX vector memories (VECMEM):

```
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  2.Ø  3.Ø  4.Ø  5.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  2.Ø  3.Ø  4.Ø  5.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  2.Ø  3.Ø  4.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  2.Ø  3.Ø  4.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  1.Ø  2.Ø  3.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  1.Ø  2.Ø  3.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  1.Ø  1.Ø  2.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  1.Ø  1.Ø  1.Ø  1.Ø  2.Ø  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

Upon return from MUNLD, B contains:

```
  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø, 1.Ø
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  2.Ø  2.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø  1.Ø
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  3.Ø  3.Ø  2.Ø  2.Ø  1.Ø  1.Ø  1.Ø  1.Ø
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  4.Ø  4.Ø  3.Ø  3.Ø  2.Ø  2.Ø  1.Ø  1.Ø
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  5.Ø  5.Ø  4.Ø  4.Ø  3.Ø  3.Ø  2.Ø  2.Ø
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

IERR = Ø

A system with NMAX available MAX modules has

```
NVEC = 8 * NMAX + 4
```

MAX vector memories, numbered from 1 to NVEC, which are partitioned into two banks of NVEC/2 vector memories each. Vector memories 1 through NVEC/2 make up one bank, and vector memories NVEC/2+1 through NVEC make up the other. The two banks of vector memories can be thought of as complementary, where vector memories 1 and NVEC/2+1 are complements, 2 and NVEC/2+2 are complements, and so forth.

Vector memories NVEC/2 and NVEC are not accessible for VMSA operations and hence are not used.

For each vector in A, MVMSA performs the NVB VMSA operations between that vector and the NVB vectors residing in one bank of the MAX vector memories, using the scale factors contained in C. The NVB resultant vectors are written into the complementary bank of MAX vector memories. IVN is toggled to point to the starting vector memory in the output bank.

```
IVN = MOD(IVN+NVEC/2, NVEC)
```

The NVB resultant vectors then become the input for the VMSA operations with the next vector of A.

Note that due to the parallel operation of the MAX, all NVEC/2 - 1 vector memories in the output bank will be overwritten. The unused NVEC/2 - 1 - NVB vector memories will contain extraneous results. Refer to EXAMPLE for more details.

IERR = 5 IVM + NEV - 1 greater than 2Ø48.

IERR = 6 IVN <= Ø.

IERR = 7 NVB > NVEC/2 - 1 or
MOD(IVN,NVEC/2) + NVB - 1 > NVEC/2 - 1
or
IVN = NVEC/2 or
IVN = NVEC
where NVEC = 8*NMAX+4 and NMAX is the number of available MAX modules. Refer to Section K.2.

If there are too many or too few formal parameters, then IERR is left unchanged.

For more information on the Matrix Oriented MAX routines, refer to Section K.4.

EXAMPLE: Assume one available MAX module and that the data in A and C is stored in column major order (normal FORTRAN). Perform the VMSA operations of the rows of a submatrix of A with a subset of the MAX vector memories using the scalars contained in the columns of a submatrix of C.

Input matrix A:

1.Ø	x.x	1.Ø	x.x	1.Ø	x.x	1.Ø	x.x	1.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
1.Ø	x.x	2.Ø	x.x	3.Ø	x.x	4.Ø	x.x	5.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x
1.Ø	x.x	2.Ø	x.x	3.Ø	x.x	2.Ø	x.x	1.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x	x.x

IAE = 12
IAV = 2
NVA = 3

Input matrix C:

1.Ø	x.x	x.x	2.Ø	x.x	x.x	1.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x
2.Ø	x.x	x.x	1.Ø	x.x	x.x	Ø.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x
3.Ø	x.x	x.x	Ø.Ø	x.x	x.x	2.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x
4.Ø	x.x	x.x	3.Ø	x.x	x.x	3.Ø
x.x	x.x	x.x	x.x	x.x	x.x	x.x

ICE = 2
ICV = 24

```
**********                                               **********
*        *                                               *        *
* MVSMA  *   --- MATRIX VECTOR SCALAR MULTIPLY ADD ---   * MVSMA  *
*        *                                               *        *
**********                                               **********
```

PURPOSE: To perform vector scalar multiply add (VSMA) operations between a matrix of vectors and a matrix of scalars in Main Memory and a matrix of vectors in the MAX vector memories.

CALL FORMAT:
```
CALL MVSMA(A, IAE, IAV, IVM, IVN, C, ICE, ICV,
                NEV, NVA, NVB, IFUN, IERR)
```

PARAMETERS:

```
A    = Floating-point input matrix of vectors in
       Main memory.
IAE  = Integer input element stride for vectors in A.
IAV  = Integer input element stride between
        vectors in A.
IVM  = Integer input starting element in the MAX
       vector memories.
IVN  = Integer input/output starting vector for the
       input/output vectors in the MAX vector memories.
C    = Floating-point input matrix of scalars.
ICE  = Integer input element stride for C.
ICV  = Integer input element stride between
       vectors in C.
NEV  = Integer input VSMA length.
NVA  = Integer input number of vectors in A to be used.
NVB  = Integer input number of vectors in the MAX
       vector memories to be used as input.  Also
       the number of scalars per vector of C to be used.
IFUN = Integer input function flag.
       IFUN = Ø:  r =  1.Ø
       IFUN = 1:  r = -1.Ø
       Note that IFUN is a bit-mapped function flag:
       IFUN = 2 is equivalent to IFUN = Ø, IFUN = 3
       is equivalent to IFUN = 1, etc.
       See DESCRIPTION for the usage of r.
IERR = Integer input/output error flag.
       See DESCRIPTION for a list of error conditions.
```

DESCRIPTION: MVSMA performs VSMA operations of length NEV between the NVA vectors in Main Memory defined by A, IAE, and IAV with the NVB vectors in the MAX vector memories defined by IVM and IVN, using the elements of C in Main Memory defined by ICE and ICV as the scale factors. The output of the VSMA operations is written into the MAX vector memories.

The operation of MVSMA can be conveniently described if the set of MAX vector memories is considered to be a matrix VECMEM(2Ø48,NVEC). Arbitrary data structures can be associated with A and C. For simplicity, assume that A and C are also matrices, dimensioned A(IAV,NVA) and C(ICV,NVA), respectively. Further, assume that IAE and ICE are both equal to one. Given these assumptions, the computations performed by MVSMA can be described in FORTRAN by:

```
   DO 3Ø i = 1, NVA
     IVO = MOD(IVN+NVEC/2, NVEC)
     DO 2Ø j = 1, NVB
       DO 1Ø k = 1, NEV
         VECMEM(IVM+k-1,IVO+j-1) = r * C(j,i) * A(k,i) +
  +                                   VECMEM(IVM+k-1,IVN+j-1)
1Ø     CONTINUE
2Ø   CONTINUE
     IVN = IVO
3Ø CONTINUE
```

To illustrate the generality and the flexibility of the data structures that can be associated with the data, suppose that A and C are one-dimensional arrays. In this case, the operations performed by MVSMA can be described in FORTRAN by:

```
   DO 3Ø i = 1, NVA
     IVO = MOD(IVN+NVEC/2, NVEC)
     DO 2Ø j = 1, NVB
       DO 1Ø k = 1, NEV
         VECMEM(IVM+k-1,IVO+j-1) = r *
  +                              C((j-1)*ICE+(i-1)*ICV+1) *
  +                              A((k-1)*IAE+(i-1)*IAV+1) +
  +                              VECMEM(IVM+k-1,IVN+j-1)
1Ø     CONTINUE
2Ø   CONTINUE
     IVN = IVO
3Ø CONTINUE
```

Summary of error conditions:

IERR = -3	NVA <= Ø.
IERR = -2	NVB <= Ø.
IERR = -1	NEV <= Ø.
IERR = Ø	No error occurred. Normal completion.
IERR = 1	No TMRAM on the system.
IERR = 4	IVM <= Ø.

MAX vector memories (VECMEM):

```
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  1.0  2.0  3.0  4.0  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  1.0  1.0  2.0  3.0  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  1.0  1.0  1.0  2.0  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  1.0  1.0  1.0  1.0  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  1.0  1.0  1.0  1.0  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

```
IVN  =   2
IVM  =   3
NVB  =   4

NEV  =   5

IFUN =   0
IERR = 999
```

Upon return from MVSMA, the MAX vector memories (VECMEM) contain:

```
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
  y.y  4.0  5.0  6.0 11.0  x.x  y.y  5.0  5.0  8.0 14.0  x.x
  y.y  6.0  5.0  5.0 13.0  x.x  y.y  8.0  5.0  9.0 19.0  x.x
  y.y  8.0  6.0  4.0 15.0  x.x  y.y 11.0  6.0 10.0 24.0  x.x
  y.y 10.0  7.0  4.0 17.0  x.x  y.y 12.0  7.0  8.0 23.0  x.x
  y.y 12.0  8.0  4.0 20.0  x.x  y.y 13.0  8.0  6.0 23.0  x.x
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
   .    .    .    .    .    .    .    .    .    .    .    .
  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x  x.x
```

```
IERR = 0
IVN  = 8
```

Note that vector memories 1 and 7 contain extraneous results, and that vector memories 6 and 12 remain unchanged because they are not accessible for VSMA operations.

APPENDIX L

MAX ROUTINES IN ALPHABETICAL ORDER

APMATH64/MAX KEY WORD INDEX

This index of APMATH64/MAX routines is sorted by key words that appear in each routine title. Each title can contain more than one key word. The key words are listed alphabetically to the right of the gap running down the center of each page.

To use the key word index, locate a key word that is representative of the desired APMATH64/MAX function. Applicable APMATH64/MAX routine names and titles can be found on the same line with each occurrence of the key word. The routine name appears in brackets ([ ]). The routine title immediately follows the routine name and continues on the other side of the gap when necessary. The ellipsis (...) is placed directly after the last word in the title if the line wraps around. The page where a particular routine is documented can be found in Appendix L.

Please detach cards along perforations.

READER'S COMMENT FORM

Your comments will help us improve the quality and usefulness of our publications. Please fill out and return this form. (The mailing address is on the back.)

Title of document: ______________________________

Your Name and Title: ______________________ Date: __________

Firm: ______________________ Department: ______________

Address: ______________________________

City: ____________ State: ____________ Zip Code: __________

Telephone Number: (_____) ____________ Extension: __________

I used this manual. . .

- ☐ as an introduction to the subject
- ☐ as an aid for advanced training
- ☐ to instruct a class
- ☐ to learn operating procedures
- ☐ as a reference manual
- ☐ other ______________

____

I found this material. . .

	Yes	No
accurate	☐	☐
complete	☐	☐
written clearly	☐	☐
well illustrated	☐	☐
well indexed	☐	☐

Please indicate below, listing the pages, any errors you found in the manual. Also indicate if you would have liked more information about a certain subject.

ARRAY

ARRAY is an independent society of people who use FPS products. Membership is free and includes a quarterly newsletter. There is an annual conference, as well as other activities. If you are interested in becoming an ARRAY member, please fill out and return this form. (The mailing address is on the back.)

Your Name and Title: ______________________ Date: __________

Firm: ______________________ Department: ______________

Address: ______________________________

City: ____________ State: ____________ Zip Code: __________

Telephone Number: (_____) ____________ Extension: __________

FLOATING POINT
SYSTEMS, INC.

P.O. Box 23489, Portland, Oregon 97223
Tel: 503/641-3151
Telex: 360470 FLOATPOIN BEAV
Telex: 4742018 FLPT UI