N87-27240

BUBBLE VECTOR IN AUTOMATIC MERGING

P. R. PAMIDI and T. G. BUTLER
RPK Corporation BUTLER ANALYSES

SUMMARY

This paper shows that it is within the capability of the DMAP language to build a set of vectors that can grow incrementally to be applied automatically and economically within a DMAP loop that serve to append sub-matrices that are generated within a loop to a core matrix. The method of constructing such vectors is explained. In an appendix are four DMAP packets for applying these techniques in different ways.

PURPOSE

The objective is to build a set of partitioning vectors that will grow incrementally to be applied automatically in a DMAP loop without analyst intervention for the purpose of appending the sub-matrices, formed in the loop, to the product matrix accumulated from preceding passes through the loop; i.e.

$$\lfloor\, 0\ 1 \,\rfloor^T$$

$$\lfloor\, 0\ 0\ 1 \,\rfloor^T$$

$$\lfloor\, 0\ 0\ 0\ 1 \,\rfloor^T$$

$$\vdots$$

$$\lfloor\, 0\ 0\ \ldots .0\ 0\ 1 \,\rfloor^T$$

DILEMMA

The DMAP language in NASTRAN does not have a module that will generate vectors of arbitrary size. No matrix mathematical operations will increase the order of a matrix as demonstrated here.

When two matrices are added or subtracted, the order of the result stays the same:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2x2} \left(\pm\right) \begin{bmatrix} m & n \\ p & q \end{bmatrix}_{2x2} = \begin{bmatrix} (a \pm m)(b \pm n) \\ (c \pm p)(d \pm q) \end{bmatrix}_{2x2}$$

If two matrices are multiplied, the order of the result will not exceed the bounding order of the multiplicands:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2x2} x \begin{bmatrix} m & n \\ p & q \end{bmatrix}_{2x2} = \begin{bmatrix} (am + bp)(an + bq) \\ (cm + dp)(cn + dq) \end{bmatrix}_{2x2}$$

If a matrix is squared, the order of the result stays the same:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^2_{2x2} = \begin{bmatrix} (a^2 + bc)(ab + bd) \\ (ca + dc)(cb + d^2) \end{bmatrix}_{2x2}$$

Merging does produce a matrix of expanded order <u>if one has at hand a partitioning vector, of the order desired, to monitor the merging.</u> This is tantamount to saying that you can expand the order of a matrix if you already have one around which is of the size that you want. A scheme is therefore needed which employs a sequence of DMAP utility modules to provide an incrementally expanding partitioning vector. However, no single module available in NASTRAN will continually increase the order of a partitioning vector as desired here. If you let this notion

soak in for a time, then the inspiration percolates to the front of your brain to say "start with something big and extract the object of required order from it." Our ideas at this point are still a long way away from the final algorithm, but this is the seminal idea. After a number of crude tries, the algorithm shown here finally took shape.

NOTATION

A vector is a column matrix : $[V] = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{Bmatrix}$

The transpose of a vector is a row matrix:

$$[V]^T = \lfloor a_1 \; a_2 \; a_3 \; a_4 \cdots\cdots\cdots \rfloor$$

To save space in the text, vectors will be written as transposes of row matrices:

$$\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{Bmatrix} = \lfloor a_1 \; a_2 \; a_3 \; a_4 \; \cdots\cdots \; a_n \rfloor^T$$

INITIAL CONSIDERATIONS

The design of the DMAP ALTER packet will hinge on the range of certain parameters.

1. The size of the core matrix to which sub-matrices are to be appended. (Number of starting columns or rows)

2. The position at which sub-matrices are to be appended; i.e. Pre or Post.

3. The size of the final matrix after all sub-matrices have been appended (Final number of columns or rows).

4. The number of columns or rows in the sub-matrix generated in each pass through the DMAP loop (Increment).

SCHEME FOR APPENDING A SUB-MATRIX IN EACH PASS THROUGH A LOOP

The nub of the idea is to operate with a series of vectors of order larger than that of the matrix to be built in the DMAP loop. The final MONITOR vector, that will be used for the actual merge operation, will be extracted from the SOURCE. The technique which is used in the intermediate operations for managing the incremental growth of the MONITOR vector is where the real interest lies. This will be elaborated in detail, but as an introduction suffice it to say that there are two SOURCE vectors which are companions such that one has the increment at the tip of the vector and the other has the increment at the base of the vector. A BUBBLE vector is created whose length remains invariant, but whose increment moves inward from its starting position in a sea of zeroes. The BUBBLE vector leaves a trail of its previous positions as an accumulation of the increments into a vector called CLUSTER. The overall length of the CLUSTER vector remains invariant as the collection of increments increases. CLUSTER is used to partition the MONITOR from one of the SOURCE vectors.

$$\{SOURCE\} \xrightarrow{CLUSTER} \{MONITOR\}$$

The steps in managing these vectors follow. For example, in the case of post-appending a single column in each pass, the SOURCE vector would have leading zeroes and a one in the last position, so a logical name would be BAS.

$$\lfloor 0000.....0000000.....00000000000.....0001 \rfloor^T$$

Use this SOURCE vector as as a replenishable reservoir from which to take slices of increasing length to form the MONITOR vector as the product matrix grows from successive appending loops.

Several examples of these slices follow:

$$\lfloor\ 01\ \rfloor^T$$

Slice taken to create a MONITOR vector to merge a single vector to the right hand side of a core consisting of a single vector, resulting in a product matrix of 2 columns.

$$\lfloor\ 00000001\ \rfloor^T$$

Slice taken to create a MONITOR vector to merge a single vector to the right hand side of a 7-col core matrix, resulting in a product matrix of 8 columns.

$$\lfloor\ 000000000000001\ \rfloor^T$$

Slice taken to create a MONITOR vector to merge a single vector to the right hand side of a 14-col core matrix, resulting in a product matrix of 15 columns.

The CLUSTER vectors needed to do the slicing from the SOURCE would be of the same order as the SOURCE, but would have a cluster of increments equal to the number of columns in the product matrix after the end of a pass; i.e. the three CLUSTER vectors that would be used to slice the three MONITOR vectors shown above from the SOURCE would be respectively:

$$\lfloor\ 000000.....0000000.....00000000000.....0011\ \rfloor^T$$

$$\lfloor\ 000000.....0000000.....00000000000.....11111111\ \rfloor^T$$

$$\lfloor\ 000000.....0000000.....00000000000.....111111111111111\ \rfloor^T$$

It is evident from the above three examples that the number of ones in the CLUSTER vector grows by an increment after each pass (in this example the increment is one). The needs of

the algorithm will be met by saving only the latest CLUSTER vector and devising some way of adding another increment in the next interior position. For example, in building the CLUSTER for the pass after the product has grown to order eight, one could add a vector to the "8" CLUSTER vector, which is of the same order as the "8" CLUSTER vector, but has zeroes everywhere except for a one in the 9th-from-the-end interior position.

```
⌊ 000000.....0000000.....00000000.....00011111111 ⌋T
                                          ↓
                                          ↓
                          +               ↓
                                          ↓
⌊ 000000.....0000000.....00000000000.....00100000000 ⌋T
                                          ↓
                                          ↓
                          =               ↓
                                          ↓
⌊ 000000.....0000000.....00000000000.....111111111 ⌋T
```

Well, here is a situation that can be used to evolve pass by pass. It is a matter of moving the "1" in the intermediate matrix to the next-interior-position . All well and good. But how does one do that? In a vulgar analogy, just bite off a zero from the front and spit it onto the end. The tools are already at hand to do just that. The two companion SOURCE vectors are of the right order. There is a "1" on the front of one of the two companions to do the biting off of a zero from the front. The other of the pair has a "1" on the end to do the spitting of a zero onto the end. This intermediate vector is saved from pass to pass. Conceptually the "1" keeps travelling to the interior on successive passes. It is analogous to a bubble moving in a liquid-filled tube; thus the name BUBBLE vector.

Using this scheme, our needs will be met;

1. The MONITOR vector grows by an increment on each pass to do the merging of an increment to the product. The MONITOR is discarded after each pass; and is regenerated on a succeeding pass.
2. The CLUSTER vector augments its cluster of "1's" by an increment on each pass to do its job of slicing off a new MONITOR vector from the SOURCE vector. It is saved in its augmented form after each pass.
3. The BUBBLE vector shifts its set of ones to the interior by an increment on each pass to do its job of adding an increment to the CLUSTER. It is saved after being shifted in each pass.
4. The pair of SOURCE vectors remains unchanged during the whole operation.

REQUIREMENTS

The analyst is to supply simple initial bulk data and should be free of having to intervene in making entries in the DMAP code, once the author has incorporated the BUBBLE scheme into a DMAP loop. It should be easy for the author to use. It should be economical on storage space.

ALGORITHM FOR APPENDING

Now to build an algorithm to meet the stipulated requirements, we will consider these requirements one by one.

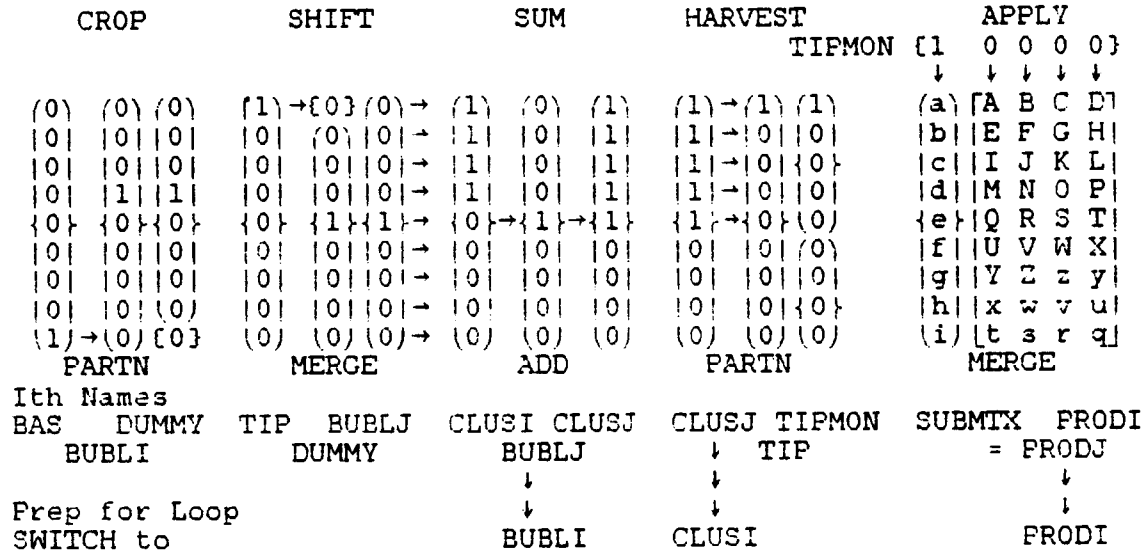
Specification: Simple initial bulk data.

The user supplies 2 SOURCE vectors. Because they are sparse, it will take a minimum of 2 cards for each vector for a total minimum of four cards. This is simple. He knows how big "big" is from the nature of the problem that he is analyzing. He knows the size of the sub-matrix to be appended so he can set the vectors' increments.

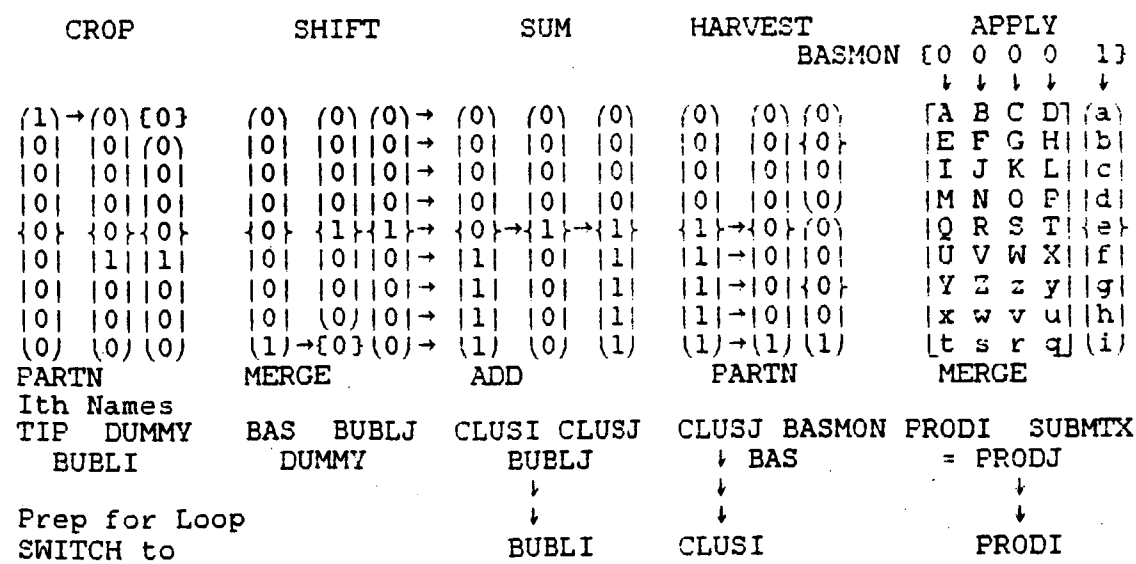
Specification: Easy for the author.

To illustrate the ease of concept, examples of the Ith step of a loop (I = 4) will be used to demonstrate how simple it is to shift from pre or post appending or from column or row appending. The operations can be characterized as CROP, SHIFT, SUM, HARVEST, AND APPLY. The first example shows pre-appending of columns and the second example shows post-appending of columns.

Pre-appending



Post-appending



Specification: Economical on storage space

The two source vectors will remain in storage for the whole time that the loop is in operation. Their length may be in the thousands, but the vectors are stored in packed form so the storage will consist of the header, the increment of 1's, and the packing index for the position of nonzero entries.

The CLUSTER will be the same length as the SOURCE vectors and will have a cluster of "1's" instead of a small increment of 1's, but the content will be mostly zeroes in packed form; so its storage requirements will be low until its cluster grows to the final size of the PRODUCT matrix at the end of the loop.

The BUBBLE vectors will be the same length as the SOURCE vectors and will have a single non-zero increment like the SOURCE vectors so their storage will be exactly the same as the SOURCE vectors.

The MONITOR vectors will exist for the duration of a merge within a loop. Their brief storage requirements are exactly the same as the SOURCE vectors, because they will differ from the source only in the order number in the header.

ALGORITHM FOR POST APPENDING A MATRIX

Organize a pair of SOURCE vectors of large order; one with leading zeroes before the increment of "1's"; and the other with trailing zeroes after the increment of "1's". Call these respectively BAS and TIP.

BAS $\lfloor 0\ 0\ 0\ 0\ 0\ \ldots\ldots 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \rfloor^T$

The # of 1's is governed by the sub-matrix order/pass; i.e. increment.

TIP $\lfloor 1\ 1\ 0\ 0\ 0\ 0\ 0\ \ldots\ldots 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \rfloor^T$

The # of 1's is governed by the increment/pass.

Find out the order of the core matrix prior to entering the loop. Use PARAML with operator = TRAILER and RECONO = trailer position # 1 corresponding to the number of columns in the core matrix. Call the parameter value that is returned by the name CORCOL for the number of core columns. But, if the sub-matrix is to be augmented by rows, set RECONO = trailer position #2 corresponding to the number of rows; then call the returned parameter by the name COROW for the number of core rows. The remainder of the explanation for this algorithm will be based upon the logic of post appending columns.

```
PARAML      COREMTX//*TRAILER*/1/CORCOL $
EQUIV       COREMTX,PRODI/TRUE $
```

Move the bubble to that CORCOL position in unity increments with these preparatory steps.

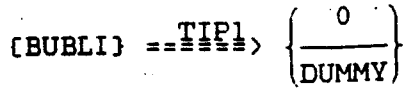
Set up a small loop to prepare the vectors BUBLI and CLUSI ahead of the main loop. Companion source vectors (SOURCE1) with increments of unity called BAS1 and TIP1 are supplied by the user with DMI cards in the bulk data. Make copies of BAS1 to act as starting configurations for CLUSI and BUBLI for molding into their loop configurations.

```
COPY        BAS1/CLUSI/0 $ For the case of post-appending.
COPY        BAS1/BUBLI/0 $
```

Set the number of movements of the bubble that are needed to prepare BUBLI.

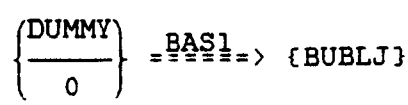
```
PARAM       //*SUB*/SHIFT/CORCOL/1 $

LABEL       BUBTOP $
FILE        BUBLI= SAVE/ CLUSI = SAVE/ PRODI = SAVE $
```

Partition off a zero from the starting vector of BUBLI with TIP1.



PARTN BUBLI, ,TIP1/DUMMY,,,/+7 $

Merge a zero onto the truncated stub of the BUBBLE vector with BAS1



MERGE DUMMY, , , , ,BAS1/BUBLJ/+7 $

The increment is now moved interior-ward in the BUBBLE vector. Combine the shifted BUBBLE vector with the starting vector of CLUSI.

{CLUSI} + {BUBLJ} = {CLUSJ}

ADD CLUSI,BUBLJ/CLUSJ/ $

Reset the internal loop names to the starting assignments in order to prepare for the next pass.

SWITCH BUBLJ,BUBLI/ /-1 $
SWITCH CLUSJ,CLUSI/ /-1 $

Return to the starting label if the loop count (internally monitored by NASTRAN) is less than the prescribed value of SHIFT.

REPT BUBTOP,SHIFT $

Now the positions of the ones in both BUBLI and CLUSI matches the size of the CORE matrix.

Set up the loop for post appending an incremental sub-matrix onto the CORE matrix in each pass.

LABEL CORTOP $

Keep shifting the increment (which can now be different from unity depending on the need of the sub-matrix in the loop) in the BUBBLE vector once per pass (with BAS and TIP instead of BAS1 and TIP1) now that it has been set by the preparatory packet. Start by truncating an increment of leading zeroes.

$$\{BUBLI\} \xrightarrow{TIP} \left\{\begin{matrix} 0 \\ \hline DUMMY \end{matrix}\right\}$$

```
PARTN       BUBLI, ,TIP/DUMMY,,,/+7 $
```

Merge on an increment of trailing zeroes to push the bubble into the interior.

$$\left\{\begin{matrix} DUMMY \\ \hline 0 \end{matrix}\right\} \xrightarrow{BAS} \{BUBLJ\}$$

```
MERGE       DUMMY, , , , ,BAS/BUBLJ/+7 $
```

Combine the shifted BUBBLE vector with the CLUSTER vector "CLUSI".

$$\{CLUSI\} + \{BUBLJ\} = \{CLUSJ\}$$

```
ADD         CLUSI,BUBLJ/CLUSJ/ $
```

Partition off the MONITOR vector to the size of the PRODUCT matrix being built in this pass.

$$\{BAS\} \xrightarrow{CLUSJ} \left\{\begin{matrix} 0 \\ \hline MNTRI \end{matrix}\right\}$$

```
PARTN       BAS, ,CLUSJ/,MNTRI, , /+7 $
```

The desired merging vector has been produced and is ready to be applied to the job of joining the current sub-matrix to the current PRODUCT matrix.

```
$$MOCK  FUNCT'L MODULE "MTXOPN" PRODUCES THE SUB-MATRIX "SUBMTX"
        MTXOPN      DBIN/SUBMTX/ $
$$  PRODUCT MATRIX "PRODI" IS AVAILABLE FROM A PREVIOUS PASS.


        [PRODI : SUBMTX] ==MNTRI==> [PRODJ]

        MERGE       PRODI, ,SUBMTX, ,MNTRI, /PRODJ/ +7 $
```

Prepare for the next pass by resetting the matrix names to their initial assignments.

```
        SWITCH      PRODJ,PRODI/ /-1 $
        SWITCH      CLUSJ,CLUSI/ /-1 $
        SWITCH      BUBLJ,BUBLI/ /-1 $
```

Apply a test to see how far the merging has proceeded with respect to the requirements of the matrix operations. Set the value of a threshold to negative when the loop is done so that a conditional jump takes the OSCAR outside the loop.

```
$$MOCK  FUNCT'L MODULE "TESTOPN" PROVIDES A TEST LOGIC FOR LOOPS.
        TESTOPN     VARY, , /DBOUT/THRESHLD $
        COND        OUTSIDE,THRESHLD $
```

Loop back to the top of the main loop so long as the value of THRESHLD remains positive. When it turns negative the conditional jump takes the sequence of operations beyond the loop.

```
        REPT        CORTOP,## $ ## VALUE IS LARGER THAN THAT
$$                  EXPECTED FOR  THRESHOLD
        LABEL       OUTSIDE $ PRODI IS PASSED OUTSIDE THE LOOP WITH
$$                       SUB-MATRICES FROM ALL LOOPS MERGED ONTO IT
```

This completes the explanation for the algorithm following the example of post-appending any sized sub-matrix by columns. In the appendix, this algorithm is coded for the four permutations of:

1. Post-append by columns.
2. Pre-append by columns.
3. Post append by rows.
4. Pre-append by rows.

CONCLUSION

It has been shown that it is within the capability of the DMAP language to build a set of vectors that can grow incrementally to be applied automatically and economically within a DMAP loop to serve to append sub-matrices that are generated within a loop to a product matrix.

APPENDIX FOR FOUR PATHS OF THE BUBBLE ALGORITHM

```
$POST APPEND COLUMNS.

PARAML      COREMTX/ /*TRAILER*/ 1 /CORCOL $
COPY        BAS1  /CLUSI/ 0 $
COPY        BAS1  /BUBLI/ 0 $
PARAM       //*SUB*/SHIFT/CORCOL/ 1 $
LABEL       BUBTOP $
FILE        BUBLI = SAVE/ CLUSI = SAVE/ PRODI = SAVE $
PARTN       BUBLI, ,  TIP1 /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,BAS1   /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
SWITCH      BUBLJ,BUBLI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
REPT        BUBTOP,SHIFT $
LABEL       CORTOP $
PARTN       BUBLI, ,TIP    /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,BAS    /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
PARTN       BAS    , ,CLUSJ/ ,MNTRI, , /+7 $
MTXOPN      DBIN/MODE/ $
MERGE       PRODI,     ,MODE, ,MNTRI, /PRODJ/+7 $
SWITCH      PRODJ,PRODI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
SWITCH      BUBLJ,BUBLI/ /-1 $
TSTOPN      VARY, , /DBOUT/THRESHLD $
COND        OUTSIDE,THRESHLD $
REPT        CORTOP,NMBR $
LABEL       OUTSIDE $
```

```
$PRE APPEND COLUMNS.

PARAML      COREMTX/ /*TRAILER*/1/CORCOL $
COPY        TIP1  /CLUSI/ 0 $
COPY        TIP1  /BUBLI/ 0 $
PARAM       //*SUB*/SHIFT/CORCOL/ 1 $
LABEL       BUBTOP $
FILE        BUBLI = SAVE/ CLUSI = SAVE/ PRODI = SAVE $
PARTN       BUBLI, ,  BAS1 /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,TIP1   /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
SWITCH      BUBLJ,BUBLI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
REPT        BUBTOP,SHIFT $
LABEL       CORTOP $
PARTN       BUBLI, ,BAS    /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,TIP    /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
PARTN       TIP    , ,CLUSJ/ ,MNTRI, , /+7 $
MTXOPN      DBIN/MODE/ $
MERGE       PRODI,     ,MODE, ,MNTRI, /PRODJ/+7 $
SWITCH      PRODJ,PRODI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
SWITCH      BUBLJ,BUBLI/ /-1 $
TSTOPN      VARY, , /DBOUT/THRESHLD $
COND        OUTSIDE,THRESHLD $
REPT        CORTOP,NMBR $
LABEL       OUTSIDE $
```

```
$POST APPEND ROWS.

PARAML      COREMTX/ /*TRAILER*/2/COROW $
COPY        BAS1  /CLUSI/ 0 $
COPY        BAS1  /BUBLI/ 0 $
PARAM       //*SUB*/SHIFT/COROW/ 1 $
LABEL       BUBTOP $
FILE        BUBLI = SAVE/ CLUSI = SAVE/ PRODI = SAVE $
PARTN       BUBLI, ,  TIP1 /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,BAS1   /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
SWITCH      BUBLJ,BUBLI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
REPT        BUBTOP,SHIFT $
LABEL       CORTOP $
PARTN       BUBLI, ,TIP    /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,BAS    /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
PARTN       BAS    , ,CLUSJ/ ,MNTRI, , /+7 $
MTXOPN      DBIN/MODE/ $
MERGE       PRODI,MODE    , , , ,MNTRI    /PRODJ/+7 $
SWITCH      PRODJ,PRODI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
SWITCH      BUBLJ,BUBLI/ /-1 $
TSTOPN      VARY, , /DBOUT/THRESHLD $
COND        OUTSIDE,THRESHLD $
REPT        CORTOP,NMBR $
LABEL       OUTSIDE $
```

```
$PRE APPEND ROWS.

PARAML      COREMTX/ /*TRAILER*/2/COROW $
COPY        TIP1  /CLUSI/ 0 $
COPY        TIP1  /BUBLI/ 0 $
PARAM       //*SUB*/SHIFT/COROW/ 1 $
LABEL       BUBTOP $
FILE        BUBLI = SAVE/ CLUSI = SAVE/ PRODI = SAVE $
PARTN       BUBLI, ,  BAS1 /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,TIP1   /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
SWITCH      BUBLJ,BUBLI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
REPT        BUBTOP,SHIFT $
LABEL       CORTOP $
PARTN       BUBLI, ,BAS    /DUMMY, , , /+7 $
MERGE       DUMMY, , , , ,TIP    /BUBLJ/+7 $
ADD         CLUSI,BUBLJ/CLUSJ/ $
PARTN       TIP    , ,CLUSJ/ ,MNTRI, , /+7 $
MTXOPN      DBIN/MODE/ $
MERGE       PRODI,MODE, , , ,MNTRI    /PRODJ/+7 $
SWITCH      PRODJ,PRODI/ /-1 $
SWITCH      CLUSJ,CLUSI/ /-1 $
SWITCH      BUBLJ,BUBLI/ /-1 $
TSTOPN      VARY, , /DBOUT/THRESHLD $
COND        OUTSIDE,THRESHLD $
REPT        CORTOP,NMBR $
LABEL       OUTSIDE $
```