```
C RCS file, release, date & time of last delta, author, state, [and locker]
C $Header: /project/yoj/arc/CCTM/src/hadv/yamo/hppm.F,v 1.3 2011/10/21 16:11:20 yoj Exp $

C what(1) key, module and SID; SCCS file; date and time of last delta:
C %W% %P% %G% %U%

C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
      SUBROUTINE HPPM ( NI, NJ, CON, VEL, DT, DS, ORI )

C-----------------------------------------------------------------------
C Function
C   This is the one-dimensional implementation of piecewise parabolic
C   method.  Variable grid spacing is allowed. The scheme is positive
C   definite and monotonic. It is conservative, and causes small
C   numerical diffusion.

C   A piecewise continuous parabola is used as the intepolation polynomial.
C   The slope of the parabola at cell edges are computed from a cumulative
C   function of the advected quantity.  These slopes are further modified
C   so that the interpolation function is monotone. For more detailed
C   information see:

C   Colella, P., and P. L. Woodward, (1984), "The Piecewise Parabolic
C   Method (PPM) for Gas-Dynamical Simulations," J. Comput. Phys. 54,
C   174-201.

C   The concentrations at boundary cells (i.e., at 1 and NI) are not
C   computed here.  They should be updated according to the boundary
C   conditions.

C   The following definitions are used:

C              |---------------> Positive direction
C
C  -->|Boundary|<-----------------Main Grid----------------->|Boundary|<--
C
C     |<------>|<------>|       ~|<------>|~       |<------>|<------>|
C       CON(0)   CON(1)            CON(i)            CON(n)  CON(n+1)
C
```

```
C     VEL(1)-->|        VEL(i)-->|        |-->VEL(i+1)      |-->VEL(n+1)
C
C      FP(0)-->|       FP(i-1)-->|        |-->FP(i)         |-->FP(n)
C
C      FM(1)<--|         FM(i)<--|        |<--FM(i+1)       |<--FM(n+1)
C
C                            -->| DS(i)  |<--

C-----------------------------------------------------------------------

C Revision History:

C   20 April, 1993 by M. Talat Odman at NCSC:
C   Created based on Colella and Woodward (1984)

C   15 Sept., 1993 by Daewon Byun at EPA:
C   Original code obtained from Phillip Colella at Berkeley

C   29 Nov.,  1993 by M. Talat Odman at NCSC:
C   Found no difference from original code

C   05 Oct.,  1993 by M. Talat Odman at NCSC:
C   Modified for EDSS archive, made discontinuity capturing an option

C   Sep 97 Jeff
C   Aug 98 - Jeff - optimize for mesh coefficients

C   David Wong - Sep. 1998
C     -- parallelized the code
C     -- Expanded the one-level nested loop which involves either with row or
C        column, into a three-level nested loop with layers and species.
C        Corresponding arrays' dimensions were adjusted accordingly
C   Jeff - optimize for mesh coefficients
C
C   David Wong - 1/8/99
C     -- BARRIER is removed
C
C   David Wong - 1/12/99
C     -- inside BNDY_HI_PE conditional code segment, NI is changed to MY_NI
C
C   David Wong - 1/12/99
C     -- change se_loop_index argument list
C     -- add new subroutine call to determine lo and hi boundary processor

C   22 Nov 00 J.Young: PE_COMM2E -> Dave Wong's f90 stenex COMM
C                      PE_COMM3E -> Dave Wong's f90 stenex COMM

C   23 Feb 01 J.Young: allocatable arrays ...
C                      Since F90 does not preserve dummy argument array
C                      indices, CONI( 1:NI+2,, ) is copied into local array
C                      CON( 0:NI+1,, ).
C                      The caller of HPPM dimensions the actual argument,
C                      as CON( -NTHIK+1:MY_NCOLS+NTHIK,, ).

C   3 Sep 01 David Wong
C     -- use "dynamic" data structure instead of F90 ALLOCATE statement to
C        avoid memory fragmentation which eventually leads to not enough
C        contigous memory (F90 bug?)
C   24 Mar 04 G.Hammond: moved all mpi communication to caller
```

```
119
120 C   06/16/04 by Peter Percell & Daewon Byun at UH-IMAQS:
121 C     - Fixed bug in using fluxes in non-uniform grids to update concentrations
122
123 C   14 Feb 05 J.Young: fix DS dimension bug
124 C   11 Oct 05 J.Young: re-dimension lattice arrays to one
125 C    1 Nov 06 J.Young: Following Glenn Hammond, moved all communication
126 C   out of HPPM; using "swap_sandia" communication in caller; update only
127 C   local values in the CGRID array within a time step, discarding previous
128 C   ghost values.
129 C    1 May 07 J.Young: Following Peter Percell, eliminate CONI,DSI using interface
130 C   specification in caller
131 C   11 May 09 J.Young: Simplify - remove STEEPEN option (never used); assume constant
132 C                      cell widths, DS( i )
133 C   11 May 10 D.Wong: Change local dynamic arrays: make allocatable to enable proper
134 C                     PGI compiliation; fix a max first dimension
135 C   16 Feb 11 S.Roselle: replaced I/O API include files with UTILIO_DEFN
136
137 C-----------------------------------------------------------------------
138
139       USE HGRD_DEFN
140       USE UTILIO_DEFN
141 #ifdef parallel
142       USE SE_MODULES          ! stenex (using SE_UTIL_MODULE)
143 #else
144       USE NOOP_MODULES        ! stenex (using NOOP_UTIL_MODULE)
145 #endif
146
147       IMPLICIT NONE
148
149 C Includes:
150
151 ! #ifdef parallel
152       INTEGER, PARAMETER :: SWP = 3
153       INTEGER, PARAMETER :: X1 = 1
154       INTEGER, PARAMETER :: X2 = 2
155       INTEGER, PARAMETER :: X3 = 3
156 ! #else
157 !     INTEGER, PARAMETER :: SWP = 1
158 !     INTEGER, PARAMETER :: X1 = 0
159 !     INTEGER, PARAMETER :: X2 = 0
160 !     INTEGER, PARAMETER :: X3 = 0
161 ! #endif
162
163 C Arguments:
164
165       INTEGER, INTENT( IN )    :: NI, NJ      ! number of zones (cells)
166       REAL,    INTENT( INOUT ) :: CON( 1-SWP:,1: ) ! conc's in the zones (cells)
167       REAL,    INTENT( IN )    :: VEL( : )    ! velocities at zone (cell) boundaries
168       REAL,    INTENT( IN )    :: DT          ! time step
169       REAL,    INTENT( IN )    :: DS          ! distance between zone (cell) boundaries
170       CHARACTER, INTENT( IN )  :: ORI         ! orientation of advection ('C'-x or 'R'-y)
171
172 C Parameters:
173
174       REAL, PARAMETER :: TWO3RDS = 2.0 / 3.0
175       REAL, PARAMETER :: SIXTH   = 1.0 / 6.0
176
177 C Local variables:
```

```
178
179       CHARACTER, SAVE :: FIRSTORI = ' '   ! for test if Col or Row orientation change
180       LOGICAL, SAVE :: FIRSTIME = .TRUE.
181
182       INTEGER, SAVE :: NSPCS
183
184 !     REAL :: FM    (    1:NI+1,   SIZE( CON,2 ) ) ! outflux from left or bottom of cell
185 !     REAL :: FP    (    0:NI,     SIZE( CON,2 ) ) ! outflux from right or top of cell
186
187 !     REAL :: CM    ( 1-X1:NI+X1+1,SIZE( CON,2 ) ) ! zone R.H. trial intercept
188 !     REAL :: CL    ( 1-X1:NI+X1 )                 ! zone L.H. intercept
189 !     REAL :: CR    ( 1-X1:NI+X1 )                 ! zone R.H. intercept
190 !     REAL :: DC    ( 0-X1:NI+X1+1,SIZE( CON,2 ) ) ! CR - CL
191 !     REAL :: C6    ( 1-X1:NI+X1 )                 ! coefficient of second-order term
192
193       REAL, ALLOCATABLE, SAVE :: FM( :,: ) ! outflux from left or bottom of cell
194       REAL, ALLOCATABLE, SAVE :: FP( :,: ) ! outflux from right or top of cell
195
196       REAL, ALLOCATABLE, SAVE :: CM( :,: ) ! zone R.H. trial intercept
197       REAL, ALLOCATABLE, SAVE :: CL( : )   ! zone L.H. intercept
198       REAL, ALLOCATABLE, SAVE :: CR( : )   ! zone R.H. intercept
199       REAL, ALLOCATABLE, SAVE :: DC( :,: ) ! CR - CL
200       REAL, ALLOCATABLE, SAVE :: C6( : )   ! coefficient of second-order term
201       REAL C0, C1
202
203       LOGICAL, SAVE :: BNDY_LO_PE, BNDY_HI_PE
204
205       CHARACTER( 96 ) :: XMSG = ' '
206       CHARACTER( 16 ) :: PNAME = 'HPPM'
207
208       REAL X, Y                 ! Courant number
209       INTEGER NMX, ASTAT
210
211       INTEGER I, S              ! loop indices
212
213 C-----------------------------------------------------------------------
214
215       IF ( FIRSTIME ) THEN
216          FIRSTIME = .FALSE.
217
218          NMX = MAX( NI,NJ )
219          NSPCS = SIZE ( CON,2 )
220          ALLOCATE( FM(     1:NMX+1,    NSPCS ),
221      &             FP(     0:NMX,      NSPCS ),
222      &             CM(  1-X1:NMX+X1+1, NSPCS ),
223      &             CL(  1-X1:NMX+X1 ),
224      &             CR(  1-X1:NMX+X1 ),
225      &             DC(  0-X1:NMX+X1+1, NSPCS ),
226      &             C6(  1-X1:NMX+X1 ), STAT = ASTAT )
227          IF ( ASTAT .NE. 0 ) THEN
228             XMSG = '*** Error allocating FM, FP, CM, CL, CR, DC, or C6'
229             CALL M3EXIT ( PNAME, 0, 0, XMSG, XSTAT1 )
230          END IF
231
232       END IF   ! Firstime
233
234       IF ( ORI .NE. FIRSTORI ) THEN
235          FIRSTORI = ORI
236          CALL SUBST_HI_LO_BND_PE ( ORI, BNDY_LO_PE, BNDY_HI_PE )
```

```
      END IF   ! FIRSTORI

C Set all fluxes to zero. Either positive or negative flux will remain zero
C depending on the sign of the velocity.

      FM( 1:NI+1,: ) = 0.0
      FP( 0:NI,: ) = 0.0

! #ifndef parallel
C If PE near bottom or left domain boundary...
C Zeroth order polynomial at the boundary cells
C First order polynomial at the next cells, no monotonicity constraint needed
!     IF ( BNDY_LO_PE ) THEN
!        DO S = 1, NSPCS
!           CM( 1,S ) = CON( 1,S )
!           CM( 2,S ) = 0.5 * ( CON( 1,S ) + CON( 2,S ) )
!        END DO
!     END IF

C If PE near top or right domain boundary...
C Zeroth order polynomial at the boundary cells
C First order polynomial at the next cells, no monotonicity constraint needed
!     IF ( BNDY_HI_PE ) THEN
!        DO S = 1, NSPCS
!           CM( NI+1,S ) = CON( NI,S )
!           CM( NI,S )   = 0.5 * ( CON( NI,S ) + CON( NI-1,S ) )
!        END DO
!     END IF
! #endif

C Second order polynomial inside the domain

      DO S = 1, NSPCS
         DO I = 2 - X3, NI + X3 - 1

C Compute average slope in the i'th zone

C Equation (1.7)
            C0 = CON( I,S )   - CON( I-1,S )
            C1 = CON( I+1,S ) - CON( I,S )
            DC( I,S ) = 0.5 * ( C0 + C1 )

C Guarantee that CM lies between CON(I) and CON(I+1) - monotonicity constraint

C Equation (1.8)
            IF ( C0 * C1 .GT. 0.0 ) THEN
               DC( I,S ) = SIGN( 1.0, DC( I,S ) )
     &                   * MIN(      ABS( DC( I,S ) ),
     &                         2.0 * ABS( C0 ),
     &                         2.0 * ABS( C1 ) )
            ELSE
               DC( I,S ) = 0.0
            END IF

         END DO   ! I

C Equation (1.6)
         DO I = 3 - X3, NI + X3 - 1
            CM( I,S ) = 0.5 * ( CON( I,S ) + CON( I-1,S ) )
```

```
296      &                - SIXTH * ( DC( I,S ) - DC( I-1,S ) )
297           END DO
298
299        END DO   ! S
300
301  C Generate piecewise parabolic distributions
302
303        DO S = 1, NSPCS
304
305           DO I = 1 - X1, NI + X1
306
307  C Equation (1.15)
308              CR( I ) = CM( I+1,S )
309              CL( I ) = CM( I,S )
310
311  C Monotonicity
312
313              IF ( ( CR( I ) - CON( I,S ) )
314      &          * ( CON( I,S ) - CL( I ) ) .GT. 0.0 ) THEN
315
316  C Temporary computation of DC and C6
317                 DC( I,S ) = CR( I ) - CL( I )
318                 C6( I ) = 6.0 * ( CON( I,S ) - 0.5 * ( CL( I ) + CR( I ) ) )
319
320  C overshoot cases - Equation (1.10)
321                 IF ( DC( I,S ) * C6( I ) .GT.
322      &               DC( I,S ) * DC( I,S ) ) THEN
323                    CL( I ) = 3.0 * CON( I,S ) - 2.0 * CR( I )
324                 ELSE IF ( -DC( I,S ) * DC( I,S ) .GT.
325      &                     DC( I,S ) * C6( I ) ) THEN
326                    CR( I ) = 3.0 * CON( I,S ) - 2.0 * CL( I )
327                 END IF
328
329              ELSE                    ! Local extremum: Interpolation
330                                      ! function is set to be a constant
331                 CL( I ) = CON( I,S )
332                 CR( I ) = CL( I )
333
334              END IF
335
336              DC( I,S ) = CR( I ) - CL( I )      ! Equation (1.5)
337              C6( I ) = 6.0 * ( CON( I,S ) - 0.5 * ( CL( I ) + CR( I ) ) )
338
339           END DO   ! I
340
341  C Compute fluxes from the parabolic distribution as in Equation (1.12)
342
343  ! #ifdef parallel
344  !        I = 0
345  !        IF ( VEL( I+1 ) .GT. 0.0 ) THEN
346  !           Y = VEL( I+1 ) * DT
347  !           X = Y / DS
348  !           FP( I,S ) = Y * ( CR( I ) - 0.5 * X * ( DC( I,S )
349  !    &                - C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
350  !        END IF
351  ! #endif
352
353  !        IF ( BNDY_LO_PE ) THEN
354              I = 0
```

```
355             IF ( VEL( I+1 ) .GT. 0.0 ) THEN
356                Y = VEL( I+1 ) * DT
357                X = Y / DS
358                FP( I,S ) = Y * ( CR( I ) - 0.5 * X * ( DC( I,S )
359      &                     - C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
360             END IF
361 !         END IF
362
363          DO I = 1, NI
364
365 C function for mass leaving interval I at lower face (I-1/2)
366 C = length of segment leaving * integral average concentration in that segment
367             IF ( VEL( I ) .LT. 0.0 ) THEN
368                Y = -VEL( I ) * DT
369                X = Y / DS
370                FM( I,S ) = Y * ( CL( I ) + 0.5 * X * ( DC( I,S )
371      &                     + C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
372             END IF
373
374 C function for mass leaving interval I at upper face (I+1/2)
375             IF ( VEL( I+1 ) .GT. 0.0 ) THEN
376                Y = VEL( I+1 ) * DT
377                X = Y / DS
378                FP( I,S ) = Y * ( CR( I ) - 0.5 * X * ( DC( I,S )
379      &                     - C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
380             END IF
381
382          END DO   ! I
383
384 ! #ifdef parallel
385 !         I = NI + 1
386 !         IF ( VEL( I ) .LT. 0.0 ) THEN
387 !            Y = -VEL( I ) * DT
388 !            X = Y / DS
389 !            FM( I,S ) = Y * ( CL( I ) + 0.5 * X * ( DC( I,S )
390 !     &                  + C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
391 !         END IF
392 ! #endif
393 !         IF ( BNDY_HI_PE ) THEN
394             I = NI + 1
395             IF ( VEL( I ) .LT. 0.0 ) THEN
396                Y = -VEL( I ) * DT
397                X = Y / DS
398                FM( I,S ) = Y * ( CL( I ) + 0.5 * X * ( DC( I,S )
399      &                     + C6( I ) * ( 1.0 - TWO3RDS * X ) ) )
400             END IF
401 !         END IF
402
403       END DO   ! S
404
405 C Compute fluxes from boundary cells
406
407 C If PE near top or left boundary...
408       IF ( BNDY_LO_PE ) THEN
409          IF ( VEL( 1 ) .GT. 0.0 ) THEN
410             Y = VEL( 1 ) * DT
411             DO S = 1, NSPCS
412                FP( 0,S ) = Y * CON( 0,S )
413             END DO
```

```
414           END IF
415         END IF
416
417   C If PE near bottom or right boundary...
418         IF ( BNDY_HI_PE ) THEN
419           IF ( VEL( NI+1 ) .LT. 0.0 ) THEN
420             Y = -VEL( NI+1 ) * DT
421             DO S = 1, NSPCS
422               FM( NI+1,S ) = Y * CON( NI+1,S )
423             END DO
424           END IF
425         END IF
426
427   C Update concentrations as in Equation (1.13)
428         DO S = 1, NSPCS
429           DO I = 1, NI
430             CON( I,S ) = CON( I,S )
431        &                 + ( FP( I-1,S ) - FP( I,S ) + FM( I+1,S ) - FM( I,S ) ) / DS
432           END DO
433         END DO
434
435         RETURN
436         END
437
```